ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DISTRIBUIDO BASADO EN SOA Y MVC PARA DESPLIEGUE DE INFORMACIÓN MÉDICA DE PACIENTES

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

LUIS EDUARDO ASANZA ALVIA

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

Quito, septiembre 2017

DECLARACIÓN

Yo, Luis Eduardo Asanza Alvia, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Luis Eduardo Asanza Alvia

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Luis Eduardo Asanza Alvia, bajo mi supervisión.

Raúl David Mejía Navarrete, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

AGRADECIMIENTOS

A mi madre por su apoyo ilimitado e incondicional, y por enseñarme siempre a tener la fortaleza de salir adelante a pesar de las dificultades.

A mi padre, que contribuyó con mi formación académica y profesional.

A mi tutor David, por su gran ayuda y colaboración en varios momentos de la vida universitaria y durante el proceso de desarrollo de este trabajo de titulación.

A Visonex, que me brindó la oportunidad de ser parte de su equipo, y que cada día me ofrece nuevas oportunidades profesionales.

A mis familiares y amigos que nunca se negaron a tenderme la mano cuando lo necesité, sobre todo en los momentos de mayor dificultad, en especial a mis tíos Ronald, Letty y Mirtha.

A mis profesores de la Escuela Politécnica Nacional, quienes me inspiraron a seguir los principios de la ética profesional.

DEDICATORIA

A mi abuelita, María Esther Novillo, quién dedicó muchos años de su vida a la docencia, y quién siempre me inculcó la importancia de estudiar desde que era muy pequeño.

A mis sobrinos: Elian, Brady, Alan, Ashley y Johan, espero que sigan los consejos de mi abuelita.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE SEGMENTOS DE CÓDIGO

RESUMEN

En este documento se presenta un resumen del diseño, implementación y pruebas del sistema para despliegue de información médica de pacientes. El documento está organizado en 4 capítulos.

En el capítulo 1 se presenta un resumen de algunos conceptos de la arquitectura de software y de los estilos arquitecturales en capas y SOA. Se presentan además algunos patrones de diseño y se revisan las tecnologías usadas durante el desarrollo de este proyecto. Posteriormente se expone un resumen de la metodología SCRUM y de los tipos de pruebas a realizarse.

En el capítulo 2 se presentan los requerimientos del sistema y se resume el diseño de cada uno de los componentes del mismo: bases de datos, capa de acceso a datos, capa de negocio, capa de servicios, capa de presentación, capa de autenticación y autorización y capa de *logs*. Posteriormente se detalla el desarrollo del sistema usando la metodología SCRUM.

En el capítulo 3 se presentan los resultados de las pruebas unitarias y de orden superior realizadas al sistema, así como un resumen de los errores más importantes encontrados en el desarrollo.

En el capítulo 4 se presentan las conclusiones y recomendaciones producto de este trabajo.

Finalmente, como anexos se incluyen el código del sistema, el manual de usuario del mismo y el glosario.

PRESENTACIÓN

En los tiempos actuales es imprescindible que las organizaciones dispongan de *software* para el manejo del negocio. Sin importar que tipos de servicio se ofrecen, una organización usa las aplicaciones de *software* con varios propósitos, entre ellos, mejorar la eficiencia de sus procesos, mejorar la calidad de sus servicios o mejorar la atención a sus clientes.

Las organizaciones que prestan servicios de salud no son la excepción ante esta realidad, ya que requieren tener aplicaciones de software que permita a los médicos tener acceso a la información médica de pacientes, a través de interfaces de usuario o través de reportes, para facilitar su trabajo dentro de la organización.

El manejo de registros médicos en papel ya es cosa del pasado. En la actualidad se debe manejar registros electrónicos que aseguren la integridad y confidencialidad de los datos. Además, estos deben poder ser accedidos desde diferentes lugares geográficos, y, sobre todo, la información médica de cada paciente debe estar centralizada en una base de datos para facilitar la administración de la misma.

El presente trabajo de titulación presenta un resumen del diseño e implementación de un sistema que permitirá desplegar información médica de pacientes y generar reportes. El sistema propuesto usa servicios web Windows Communication Foundation (WCF) para exponer información médica proveniente de una base de datos SQL Server y la tecnología MVC.NET de Microsoft para la creación de las interfaces de usuario. La aplicación web posee una interfaz de usuario que permite administrar usuarios del sistema y además implementa un mecanismo de autenticación y autorización de usuarios del sistema, a través de roles de usuario. Adicionalmente, la aplicación web posee varias interfaces de usuario donde se despliega la información médica. El sistema también permite generar reportes en formato PDF, XLSX y HTML, a través de la herramienta DoodleReport. Finalmente, el sistema incluye la generación de *logs* ante eventos específicos durante la ejecución del sistema.

CAPÍTULO 1. MARCO TEÓRICO

1.1 ARQUITECTURA DE SOFTWARE

1.1.1 INTRODUCCIÓN

Previo al desarrollo de una aplicación de software, se debe definir la estructura de la misma. La arquitectura de software busca definir dicha estructura para que la misma sea capaz de cumplir con los requerimientos operacionales, y además persigue optimizar ciertos atributos de calidad como el rendimiento y la seguridad en el sistema.

La arquitectura de software tiene grandes implicaciones puesto que “el escoger de manera incorrecta la arquitectura, si llega a cambiar el comportamiento del sistema en base a los requerimientos del negocio, obliga a cambiar una gran porción de código, mientras que, seleccionar la arquitectura adecuada permite controlar que no se dispare un presupuesto especificado y que no se extienda el tiempo planificado de desarrollo de la aplicación” [1].

Dentro de la arquitectura de software no existe un camino único a seguir, es decir, existen varias alternativas que permiten llegar al objetivo de definir una solución estructurada para un sistema, es por eso que en [2] se hace hincapié en que “la arquitectura de software se trata más de poner en una balanza varias opciones y escoger de entre ellas la opción o las opciones que más se adapten a la aplicación”.

1.1.2 DEFINICIÓN

Existen varias definiciones de Arquitectura de Software, por ejemplo, en [3] se establece lo siguiente:

“La arquitectura de software abarca el conjunto de decisiones importantes acerca de la organización de un sistema de software que incluye la selección de los elementos estructurales y sus interfaces mediante el cual el sistema se compone; comportamiento como se especifica en la colaboración entre dichos elementos; composición de estos elementos estructurales y de comportamiento en subsistemas más grandes; y un estilo arquitectónico que guía a esta organización. La arquitectura de software también incluye la funcionalidad, facilidad de uso, la resistencia, el rendimiento, la reutilización, la comprensibilidad, las limitaciones económicas y tecnológicas, ventajas y desventajas y las preocupaciones estéticas”.

Se puede resumir a la arquitectura de software como el proceso de escoger a alto nivel (i.e. dejando a un lado los detalles) los elementos que definen la estructura o estructuras del sistema, y como se relacionan entre sí, basado en un amplio rango de factores como: parámetros de calidad, funcionabilidad, rendimiento, confiabilidad, escalabilidad, entre otros. La arquitectura de software no especifica el uso de un lenguaje de programación en particular.

1.1.3 PRINCIPIOS CLAVES DE DISEÑO EN ARQUITECTURA DE SOFTWARE

Según [3], el pensamiento actual de la arquitectura de software asume que el diseño evolucionará en el tiempo y que no se puede saber con anticipación toda la arquitectura del sistema. El diseño evolucionará durante las etapas de implementación mientras se adquiere más conocimiento y mientras se prueba el funcionamiento de la aplicación en el mundo real. La evolución en el diseño provoca costos. Es por eso que se recomienda los siguientes principios claves que ayudan a crear una arquitectura de software que permitirá minimizar costos y mantenimiento, y que además promueve la usabilidad y extensibilidad:

- Separación de asuntos: Implica dividir la aplicación en distintas funcionalidades y evitar el traslape de ellas.
- Principio de Responsabilidad Única: Cada componente debe ser responsable solo de una funcionalidad específica.
- Principio de Menor Conocimiento: Un objeto o componente no debe conocer sobre los detalles internos de otros objetos o componentes. Mientras menos sepa uno del otro, más desacoplados estarán entre sí.
- Principio “No te repitas” (DRY - *Don’t Repeat Yourself*): Una funcionalidad específica debería ser implementada en un solo componente, y así evitar duplicados.
- Minimizar diseño por adelantado: Significa que solo se debe diseñar lo que es necesario a ese momento. Hay que evitar invertir tiempo en diseñar algo que luego no se va a necesitar.

1.1.4 REQUERIMIENTOS ARQUITECTURALES

Según [4] un requerimiento arquitectural “describe una condición o capacidad a la que un sistema debe cumplir ya sea derivado directamente de las necesidades del usuario, o declarado en un contrato, norma, especificación u otro documento

impuesto formalmente. Un requerimiento arquitectural es cualquier requerimiento que es relevante para la arquitectura.”

La clasificación de los requerimientos está dada por FURPS+, siendo FURPS un acrónimo representado por:

- *Functionality* (Funcionalidad): Representa las principales funcionales del producto. Por ejemplo, desplegar información médica de pacientes o generar reportes.
- *Usability* (Usabilidad): Tiene que ver con la estética y la consistencia de la interfaz de usuario.
- *Reliability* (Confiabilidad): Aquí entran características como el tiempo en el cual el sistema está operativo (disponibilidad), la exactitud con la que se muestran los datos y la habilidad de recuperación ante fallos.
- *Performance* (Rendimiento): Está relacionado con el tiempo de respuesta del sistema.
- *Supportability* (Apoyo): Está relacionado, por ejemplo, con la facilidad con la que se puede probar el sistema, compatibilidad, que facilidad existe para mantener la aplicación, entre otras.

Los 4 últimos requerimientos (usabilidad, confiabilidad, rendimiento y apoyo) juntos se denominan requerimientos no funcionales y de acuerdo con IBM, también se los conoce como requerimientos operacionales.

El + en FURPS+ se usa para identificar categorías adicionales que usualmente representan impedimentos. Estos son: requerimientos de diseño, de implementación, de interfaz, y requerimientos físicos.

1.1.5 ESTILOS ARQUITECTURALES

Un estilo arquitectural es un conjunto de principios que definen a alto nivel un aspecto del sistema (como la estructura) [3] [5]. Es muy común que un sistema use varios estilos arquitecturales, aprovechando así, los beneficios de cada uno de ellos. Uno de los estilos arquitecturales más usados es el estilo arquitectural en capas. A continuación, se muestra una breve descripción de los estilos arquitecturales que se usarán para la construcción del sistema.

1.1.5.1 Estilo arquitectural en capas

Este estilo arquitectural permite estructurar el sistema en capas (dividir el código en secciones lógicas). Este modelo está basado en la separación de responsabilidades [6].

Una capa es una sección del sistema la cual posee una responsabilidad específica. La responsabilidad de una capa es la funcionalidad que implementa.

Usualmente en este modelo, a las capas se les suele llamar niveles (*tier*), sin embargo, Microsoft pone énfasis en que son términos distintos. En [7] se define a una capa como el agrupamiento lógico de funcionalidad en una aplicación mientras que nivel describe la distribución física de funcionalidad en diferentes servidores, redes o ubicaciones remotas.

El estilo arquitectural en capas da solución a varios problemas como los que se mencionan a continuación:

- Un sistema es demasiado complejo para ser comprendido en su totalidad
- Un sistema es difícil de mantener
- Un sistema cuyos elementos no estables no están aislados
- Un sistema cuyos elementos más reusables son difíciles de identificar
- Un sistema que es construido por diferentes equipos de desarrollo, probablemente con diferentes habilidades

Un sistema que implementa el estilo arquitectural en capas usualmente dispone de tres de ellas tal como se muestra en La **Figura 1.1**. Las capas son:

- Presentación
- Negocio (*Business Logic Layer*)
- Acceso a datos (*Data Access Layer*)

En ocasiones un sistema puede disponer de una capa adicional denominada capa de servicios. “Cuando un sistema debe proveer servicios (por ejemplo, servicios web), se suele añadir la capa de servicios” [3].

La capa de servicios se encuentra situada entre la capa de presentación y la capa de negocio tal como lo muestra la **Figura 1.2**.

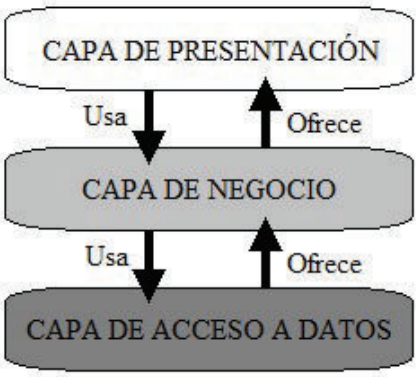


Figura 1.1. Estilo arquitectural de 3 capas

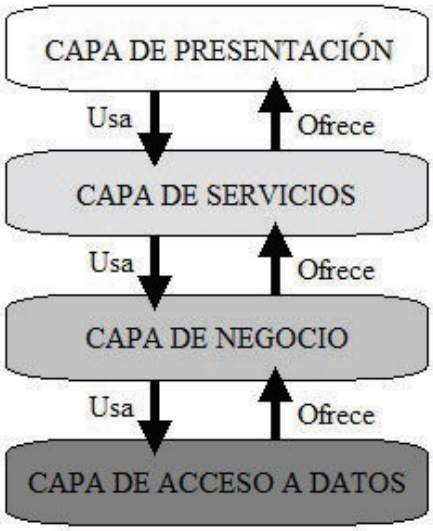


Figura 1.2. Estilo arquitectural de 4 capas

1.1.5.1.1 *Capa de presentación*

La capa de presentación es aquella que incluye la lógica de presentación, y la interfaz de usuario.

La interfaz de usuario contiene los elementos usados para desplegar información al usuario. La interfaz de usuario de una aplicación web se divide en 3 componentes:

- Estructura: La interfaz de usuario se construye con código HTML.
- Comportamiento: Las interfaces pueden ser alteradas en su estructura con código de cliente (como JavaScript). También se puede añadir animaciones o comportamiento del lado del cliente.

- Presentación: La manera en que luce la interfaz se define a través de CSS[1] (*Cascade Style Sheets*).

Además de la interfaz de usuario, a la capa de presentación pertenecen los siguientes elementos [2]:

- Modelo de Datos: La interfaz de usuario se construye en base al modelo de datos.
- Validación básica[2]: Permiten corroborar si la información que ingresa el usuario en la interfaz de usuario es válida antes que esta sea enviada al servidor para su procesamiento.
- Lógica para manejar interacciones del usuario a través de la interfaz: Ante una interacción del usuario, la capa de presentación será la encargada de responder a esta acción.

1.1.5.1.2 Capa de lógica de negocio

La Capa de Lógica de Negocio o BLL (*Bussiness Logic Layer*) "implementa la funcionalidad núcleo del sistema, y encapsula la lógica de negocio relevante" [3].

La lógica de negocio está relacionada con la obtención, procesamiento, transformación y administración de los datos de aplicación. Además, implica la implementación de reglas de negocio y de sus políticas, así como también se asegura de la validez y consistencia de datos.

Las entidades de negocio (también llamadas objetos de negocio) pertenecen a la capa de negocio. Estas encapsulan la lógica de negocio y datos necesarios para representar elementos de la vida real dentro de la aplicación.

1.1.5.1.3 Capa de acceso a datos

La capa de acceso a datos o DAL (*Data Access Layer*) es la encargada de comunicarse con la base de datos, y permite realizar operaciones sobre ella (como

---

[1] CSS es un lenguaje de programación usado para describir la presentación de documentos web.
[2] Validación básica incluye validaciones de campos requeridos, comprobación de tipo de datos, validación de rangos de valores permitidos, entre otros. Existen también validaciones de negocio, las cuales deben ser delegadas a la capa de negocio, sin embargo, eso no implica que no puedan ser implementadas también en la capa de presentación.

crear, leer, actualizar o borrar filas en las tablas de la base). Además, permite el manejo de transacciones y concurrencia. La capa de acceso a datos expone interfaces genéricas para que puedan ser consumidas por la capa de Negocio.

Para facilitar y acelerar el proceso de programación de esta capa, es muy común el uso de un ORM (*Object Relational Mapping*). El ORM es una herramienta que asocia los objetos relacionales de base de datos a objetos de lenguaje de programación, de tal manera que permite [2]:

- Crear entidades basándose en las tablas de la base de datos
- Dar seguimiento a los cambios realizados sobre un objeto en lenguaje de programación y generar el comando SQL correspondiente.
- Realizar operaciones CRUD (*Create Read Update Delete*) sobre la base.
- Manejar transacciones y concurrencia.

ADO.NET *Entity Framework* es una solución de Microsoft para la implementación de un ORM.

1.1.5.1.4 Capa de servicio

Cuando un sistema expone un conjunto de servicios (en este caso servicios WCF[3]) no solo para que sean consumidos por clientes propios de la aplicación, sino también por clientes externos, se debe considerar la posibilidad de tener una capa de Servicio.

La capa de servicio sirve para adaptar la información que viene de la BLL según los requerimientos de la capa de presentación.

En la capa de servicio se puede implementar seguridad para permitir o denegar acceso de clientes.

1.1.5.2 Estilo arquitectural SOA

SOA (*Service Oriented Architecture*) es un concepto definido de varias formas por diferentes organizaciones. Sun Microsystem en 1990 estableció que “SOA es un estilo arquitectural para construir aplicaciones de software que usan servicios

[3] Un servicio WCF es un servicio web implementado a través del *framework* WCF (Windows Communication Foundation) de Microsoft.

disponibles en una red como la web. Promueve el desacoplamiento entre los componentes de software de modo que puedan ser reutilizados" [8].

El W3C[4] también tiene su propia definición de SOA, la cual indica que "es una forma de arquitectura de sistemas distribuidos, donde un sistema distribuido consiste de diversos agentes de software discretos que deben trabajar juntos para realizar algunas tareas. Un agente es un programa que actúa en nombre de una persona u organización" [9].

Tomando en cuenta ambas definiciones se concluye que:

- SOA no está atado a ninguna tecnología de programación.
- SOA se basa en la construcción de servicios disponibles en una red.
- Un servicio en SOA no tiene que ser específicamente un servicio web. Un servicio de Windows que corre dentro de una red de área local es un ejemplo de un servicio que no es de tipo web.
- SOA promueve la reutilización de los servicios.

1.1.5.2.1 Servicios basados en SOA

Tal como sucede con SOA, el término "servicio" también se define de varias maneras por diferentes empresas.

Según Sun Microsystem: "un servicio es una implementación de una funcionalidad de negocio bien definida, y estos pueden ser consumidos por clientes en diferentes aplicaciones o procesos de negocio" [8].

Mientras que el W3C lo define como: "un servicio es un recurso abstracto que representa una capacidad de realizar tareas que representa una funcionalidad coherente desde el punto de vista de una entidad proveedora[5] y de la entidad solicitante[6]".

---

[4] El W3C (*World Wide Web Consortium*) es una comunidad que desarrolla estándares para la web.
[5] La entidad proveedora es una persona o un grupo de personas de la vida real (que probablemente representan a una organización) que posee un agente proveedor (servicio web) el cual realiza acciones en su nombre.
[6] La entidad solicitante es una persona o un grupo de personas de la vida real (que probablemente representan a una organización) que requiere hacer uso de un servicio de una entidad proveedora. Por ejemplo, una organización accede a un servicio web de la entidad proveedora para obtener los tratamientos de un paciente.

Las definiciones presentadas indican que los servicios SOA no son específicos para una plataforma o tecnología de programación. Tampoco se especifica que un servicio debe ser de tipo web. Sin embargo, es muy común que se hable de servicios web cuando se habla de SOA.

1.1.5.2.2 Servicio web SOAP

El W3C *Working Group*, en su documento *Web Services Architecture* define al servicio web como: "un sistema de software diseñado para soportar la interoperabilidad de la interacción máquina-máquina a través de una red. Este tiene una interfaz descrita en un formato procesable por un computador denominado WSDL[7]. Otros sistemas interactúan con el servicio web de una manera prescrita por su descripción usando mensajes SOAP[8], típicamente transportados usando HTTP[9] con una serialización XML[10] en conjunto con otros estándares relacionados a la web". [9]

El W3C *Working Group* identifica dos grandes tipos de servicios web: servicios RESTful y servicios web arbitrarios. Los servicios web arbitrarios exponen interfaces específicas de aplicación, es decir, operaciones arbitrarias definidas por el desarrollador, a diferencia de los servicios RESTful donde los agentes proveen semánticas de interfaz genéricas como: *retrieve*, *create*, *update* y *delete*.

Los servicios SOAP usan protocolos como SOAP y HTTP para el envío de mensajes XML tal como se ve en la **Figura 1.3**.

Tecnología Base

Los servicios usan para su comunicación tecnología basada en XML. XML ofrece un formato de datos estandarizado, flexible y extensible. Al ser estandarizado, ofrece interoperabilidad.

---

[7] WSDL (*Web Services Description Language*) es un formato de tipo XML usado para describir servicios en la red.
[8] SOAP es un protocolo ligero usado para el intercambio de información estructurada en un ambiente distribuido.
[9] HTTP (*Hypertext Transfer Protocol*) es un protocolo de la capa de aplicación usado para la comunicación entre aplicaciones y servicios web.
[10] XML (*Extensible Markup Language*) es un formato de texto estructurado usado para el envío de mensajes a través de la red.

La principal ventaja de usar XML es la estructura que maneja. La gramática de un documento XML está definida en el DTD[11].

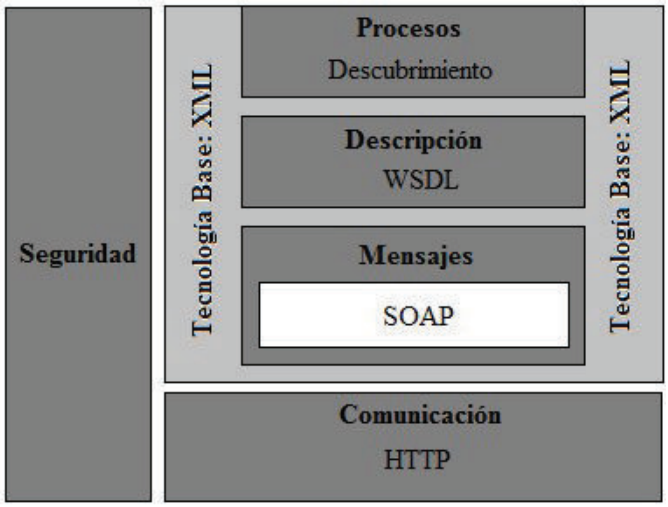


Figura 1.3. Arquitectura de Servicios Web

Mensajes

Los servicios web usan el protocolo de mensajería SOAP para el intercambio de mensajes. “SOAP es un protocolo ligero destinado a intercambiar información estructurada en un entorno descentralizado y distribuido. Utiliza tecnologías XML que proporciona una construcción de mensaje que se puede intercambiar sobre una variedad de protocolos subyacentes (entre ellos HTTP) y que es independiente de cualquier modelo de programación particular”. [10]

Comunicación

Los mensajes SOAP son enviados a través del protocolo HTTP (*Hypertext Transfer Protocol*).

Descripción

El WSDL (*Web Services Description Language*) es un formato XML para describir servicios web. WSDL provee una descripción abstracta del servicio, se enfoca en lo que hace el servicio mas no como lo hace ni donde lo hace [11].

---

[11] DTD (*Document Type Declaration*) es un documento que define la estructura, los elementos y los atributos válidos de un documento XML.

Procesos

Los servicios en SOA deben ser descubribles a través de su URL (*Uniform Resource Locator*).

Seguridad

Los servicios en SOAP deben proveer, de ser necesario, mecanismos de seguridad para que solo sean accedidos por los usuarios que tengan acceso.

1.1.6 PATRONES DE DISEÑO

Un patrón de diseño "describe un problema el cual ocurre por reiteradas ocasiones en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar esta solución un millón de veces, sin nunca hacerlo de la misma manera dos veces" [12].

Un patrón de diseño es distinto a un estilo arquitectural ya que los estilos arquitecturales son descripciones abstractas y no están atados a ningún paradigma de programación específico como el orientado a objetos.

A continuación, se describe algunos patrones que serán usados en el presente Trabajo de Titulación:

1.1.6.1 MVC (Modelo Vista Controlador)

MVC es un patrón de diseño que se suele usar en la capa de presentación en una arquitectura de 4 capas. El objetivo principal de este patrón es la separación de responsabilidades de cada una de las tres piezas que conforman MVC.

El patrón MVC separa la aplicación en tres unidades: Modelo, Vista, y Controlador [13]:

- Modelo: Contiene o representa el dato con el que el usuario trabaja. No es necesario que sean entidades del negocio, ya que pueden ser simplemente modelos de la vista, que representa datos enviados entre las vistas y controladores.
- Vista: Se usa para dibujar la interfaz de usuario, se basan en el modelo.
- Controlador: Procesa peticiones provenientes del cliente (en este caso cliente se refiere al explorador web que accede a la aplicación web).

La **Figura 1.4** muestra la interacción de las unidades de MVC desde que la aplicación web recibe una petición.

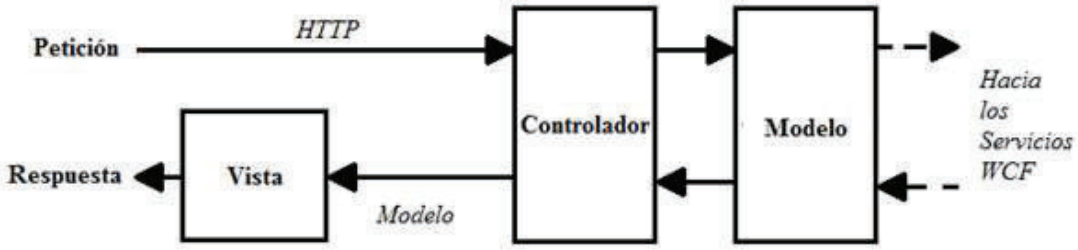


Figura 1.4. Interacciones en una aplicación MVC

ASP.NET MVC es un *framework* de aplicaciones web de Microsoft que usa como base el patrón MVC [14].

1.1.6.2 Fachada

Dentro de la capa de negocio, en caso de que se requiera, se puede implementar el patrón de diseño “Fachada”. Mediante este patrón se puede centralizar varias operaciones en un solo método, como una única transacción, de esa manera externamente no se conocen los detalles de las operaciones que se realizan internamente [15].

1.1.6.3 DTO (Data Transfer Object)

Cuando el sistema expone datos a través de servicios, se recomienda el uso del patrón de diseño DTO (*Data Transfer Object*) para organizar de mejor manera los datos en estructuras unificadas, logrando así minimizar el número de llamadas remotas a los servicios [3].

Otro beneficio de usar DTO es el desacoplamiento de entidades de datos del cliente consumidor de los servicios web, ya que, si llega a cambiar alguna entidad de negocio, la aplicación cliente no tendrá que cambiar.

Sin embargo hay una desventaja respecto al uso de DTO, y es que se requiere de más código para crear las nuevas estructuras y los adaptadores que trasladen la información desde las entidades de negocio hacia el DTO, y si llega a crecer mucho la aplicación, podría crecer en gran medida el número de DTO y adaptadores, por lo que [5] menciona un enfoque mixto, donde ciertos servicios expongan entidades

del negocio para clientes internos, mientras que en otros servicios se use DTO para clientes externos y así ajustarse a sus necesidades.

1.1.6.4 Repositorio

El patrón de diseño repositorio permite aislar los objetos de negocio de los detalles del código necesario para acceder a la base de datos. Para implementarlo se define una interfaz con una colección de operaciones que permiten el acceso a objetos del negocio.

En [12] se indica que el patrón repositorio ayuda a minimizar duplicados en la lógica de consultas a la base de datos.

1.1.6.5 Unit of work (Unidad de trabajo)

El patrón de diseño *unit of work* "mantiene una lista de objetos afectados por una transacción de negocio" que puede afectar a la base de datos y coordina la escritura de los cambios a la misma [12].

Este patrón de diseño permite realizar el seguimiento de las modificaciones que se han realizado sobre los objetos y mantenerlas en memoria. Cuando se da seguimiento a dichas modificaciones, no es necesario realizar peticiones a la base de datos por cada una de ellas.

1.1.7 COMPONENTES TRANSVERSALES

1.1.7.1 Autenticación y autorización

La Autenticación es el proceso de validar las credenciales de un usuario (como el nombre de usuario y su contraseña) contra alguna autoridad [16].

Si las credenciales ingresadas por la entidad son válidas, se considera a esta entidad como autenticada, y podrá acceder a determinados recursos del sistema en base a la autorización.

La Autorización es el proceso por el cual se determina si la entidad tiene permisos para acceder a algún recurso que esté solicitando [17].

El proceso de autorización sucede después del proceso de autenticación. ASP.NET Identity es un *framework* de Microsoft para implementar autenticación y autorización.

1.1.7.2 Manejo de excepciones

Una Excepción (Evento Excepcional) es un evento que ocurre en tiempo de ejecución de un programa, y que indica que algo falló.

Es necesario implementar el manejo de excepciones en un sistema para añadir seguridad y confiabilidad.

Añadir manejo de excepciones permite diagnosticar y resolver errores de aplicación. Así mismo, permite hacer aplicaciones más robustas, ya que son más resistentes a fallos.

1.1.7.3 Logging

Es el proceso de registrar eventos o actividades que se dan en el sistema para poder dar seguimiento, o para identificar posibles errores o ataques del sistema. Los *logs* pueden ser almacenados en archivos de texto, o en registros de base de datos.

A veces la información generada por los *logs* es tan reducida que provoca que un ataque pase desapercibido. En otras ocasiones la información de *logs* es demasiada extensa, que se vuelve inmanejable, o inclusive puede afectar el rendimiento del sistema.

En [18] se sugiere generar *logs* de las siguientes áreas:

- Todos los intentos de autenticación deben ser registrados.
- Accesos a áreas sensibles a la seguridad.
- Áreas donde se requiera estar autenticado.
- Cualquier modificación o borrado de datos críticos.
- Cualquier transacción o tarea en un sistema crítico, que puede servir para auditoría. Por ejemplo, un sistema que maneja información médica.
- Cualquier modificación de los registros de *logs*.

1.1.7.4 Validaciones

El sistema debe implementar un mecanismo de validación de datos para evitar inconsistencia de datos y violaciones de reglas de negocio. Para evitarlo, hay que considerar las siguientes directrices [3]:

- En lo posible definir listas de posibles valores de entrada permitidos, en vez de definir una lista con los valores no permitidos.
- Las validaciones del lado del cliente (JavaScript) no son suficientes para asegurar el sistema. Es necesario implementar también las validaciones en el lado del servidor.
- Centralizar las validaciones en componentes separados que puedan ser reutilizados, inclusive se puede considerar implementar las validaciones en una librería externa.
- Asumir siempre que los valores de entrada son maliciosos. Es necesario validar siempre los valores de entrada y evaluar rangos aceptables, por ejemplo, longitud máxima de cadena de caracteres, rangos numéricos, formatos, tipos, etc.

1.2 TECNOLOGÍAS PARA APLICACIONES WEB

A continuación, se revisarán las tecnologías que se usarán en este proyecto.

1.2.1 PAQUETES NUGET

Es un administrador de paquetes para el desarrollo de software en .NET. Permite crear y subir paquetes *nugets* para que puedan ser instalados por otros desarrolladores.

Un paquete *nuget* es aquel que contiene archivos librerías (de código) o cualquier otro tipo de archivo cliente como CSS[12] o JS[13]. Cuando un paquete *nuget* es instalado, esos archivos son descargados a la solución.

1.2.2 HTML (HYPERTEXT MARKUP LANGUAGE)

HTML es un tipo de lenguaje utilizado para crear páginas web. HTML usa etiquetas (*tags*). La mayoría de las etiquetas poseen su correspondiente etiqueta de terminación, tal es el caso de la etiqueta `<b>` que termina con la etiqueta `</b>`. La etiqueta de cerrado se caracteriza por tener un slash (`/`) dentro de ella. Un ejemplo de etiqueta que no requiere etiqueta de cerrado es el salto de línea `<br>`.

---

[12] Archivo CSS (*Cascading Style Sheets*) es aquel que contiene lenguaje para definir la presentación de documentos HTML.
[13] Archivo JS (JavaScript) es aquel que contiene código en lenguaje JavaScript para añadir comportamiento a los documentos HTML del lado del navegador.

HTML tiene un conjunto de nombres de etiquetas válidos, y todos los navegadores están creados para entenderlo.

La última versión de HTML es HTML5. HTML5 trata de resolver la necesidad de sitios web interactivos donde se pueda reproducir audio y video, y soportar animaciones, con el fin de mejorar la experiencia del usuario al usar la interfaz.

HTML5 pone énfasis en separar estructura, presentación y comportamiento. Las etiquetas HTML proveen estructura, más no presentación.

1.2.2.1 DOM (Document Object Model)

"DOM es una API (*Application Programming Language*) para documentos HTML válidos" [19]. Un documento válido HTML significa que está bien estructurado, y DOM permite acceder y manipular la estructura del documento HTML.

El principal objetivo de DOM es proveer una interfaz de programación estándar que puede ser usado en una amplia variedad de ambientes y aplicaciones. Además, DOM está diseñado para ser usado con cualquier lenguaje de programación [19]. Tal es el caso de JavaScript y jQuery, capaces de usar DOM para acceder al documento HTML.

1.2.3 CSS (CASCADING STYLE SHEETS)

CSS es un lenguaje cuyo objetivo es crear un aspecto consistente a una página web y separar la estructura de la presentación [20].

La manera en que CSS provee presentación a una estructura HTML es definiendo como se muestra cada elemento HTML. CSS provee estilos para definir márgenes, bordes, colores, tipo de fuente, alineación de texto, posiciones, etc. CSS usa selectores para definir sobre que elemento o elementos HTML se les aplicará cierto estilo.

En CSS, un estilo es una regla que describe como dar formato a una parte específica de un documento HTML. Una hoja de estilos es un conjunto de reglas de estilo.

La principal característica de CSS es *Cascading*, lo que implica una sobre escritura de los últimos estilos definidos sobre los estilos previamente definidos. En otras

palabras, el orden en que se definen los estilos, y el orden en que se referencian las hojas de estilo sí importa, ya que los últimos estilos pueden sobrescribir a los primeros.

1.2.4 BOOTSTRAP

"Bootstrap es un *framework* HTML, CSS y JavaScript para el desarrollo de proyectos web responsivos que implementan la filosofía primero-móvil[14]" [21].

Bootstrap es una herramienta que facilita el desarrollo de la interfaz de usuario de las aplicaciones web. La principal ventaja de usar Bootstrap es que permite añadir responsividad a las páginas web, de una manera rápida y simple.

Bootstrap permite implementar distintos controles web.

1.2.5 JAVASCRIPT

JavaScript (JS) es un lenguaje de codificación ampliamente utilizado para añadir comportamiento a los documentos HTML y hacerlos más interactivos. Por ejemplo, con JavaScript se puede incluir animaciones, añadir acciones a los botones, ocultar o mostrar objetos del documento HTML, etc. Además, con JavaScript es posible hacer una petición al servidor web asincrónicamente.

A JS se lo conoce como un lenguaje de programación del lado del cliente, debido a que no se ejecuta en el servidor web, sino que lo hace completamente en el cliente web.

JavaScript es un lenguaje sin tipo. Al declarar una variable, no es necesario especificar su tipo, siempre se la declarará como `var`. "El valor contenido en la variable podrá ser un objeto, un valor primitivo (indefinido, `null`, booleano, número o cadena de caracteres) o una función (en JS, una función es un objeto que puede ser llamado, y una función dentro de un objeto, se lo conoce como método)" [20].

1.2.6 JQUERY

Es una librería de JavaScript que permite, entre varias cosas, la manipulación de los elementos de un documento HTML de una manera sencilla con una sintaxis fácil

---

[14] *Mobile First* (Primero-móvil) es una estrategia de codificación, que se enfoca en diseñar las aplicaciones web primero para dispositivos móviles antes de ser diseñadas para un dispositivo de escritorio como una PC.

de entender. Además, permite crear manejadores de eventos (un evento muy común ocurre cuando un usuario hace clic sobre un botón, a este se lo denomina evento clic, y jQuery es capaz de detectar dicho evento para añadir funcionalidad antes de que el evento ocurra), animaciones y peticiones AJAX[15].

La implementación de jQuery es muy simple, basta con añadir las librerías gratuitas a la solución, e incluir las referencias de estas en las páginas web donde se implementará sentencias jQuery.

1.2.7 MVC.NET

Es un *framework* de Microsoft para el desarrollo web. Tiene como base al patrón de diseño MVC para la implementación de aplicaciones web.

Como MVC.NET implementa el patrón de diseño MVC, este ofrece la ventaja de separar responsabilidades en cada una de las unidades que conforman a MVC: Modelo, Vista y Controlador.

MVC.NET permite el uso de *helpers*[16] (generadores de código HTML) que simplifica el proceso de desarrollo en la vista. Así mismo es fácil añadir código jQuery y CSS a las vistas, ya que estas tienen código HTML limpio. Las vistas soportan el uso de sintaxis Razor.

1.2.7.1 Razor

Es un motor de vistas del *framework* de MVC (MVC.NET) que procesa contenido y que busca instrucciones. Usualmente en la vista se ubican instrucciones para insertar contenido dinámico que luego es enviado al explorador web [13].

La sintaxis Razor es simple de programar. El código Razor se puede incrustar en el documento HTML dentro de la vista MVC, y es procesado antes de que el resultado final sea enviado al explorador web.

---

[15] AJAX (*Asynchronous* JavaScript *and XML*) es una tecnología que permite cargar contenido adicional en la página web realizando peticiones al servidor web, sin necesidad de cargar un nuevo documento HTML.

[16] En MVC, *helper* es un generador de código HTML. Los *helper* son implementados en las vistas de MVC usando código de servidor C#. Antes de que una vista MVC sea desplegada en el explorador web, el *helper* es procesado para que devuelva el contenido HTML y lo inserte dentro de la vista, para que finalmente pueda ser desplegado en la interfaz de usuario.

Las vistas que soportan Razor con código C#, tienen la extensión CSHTML.

1.2.8 WCF.NET

Tecnología de Microsoft que provee una manera uniforme de desarrollar aplicaciones distribuidas al proveer un modelo de programación orientado a servicios.

1.2.9 ENTITY FRAMEWORK

ADO.NET *Entity Framework* (EF) es un *framework* desarrollado para la plataforma Microsoft .NET, que permite acceder a datos relacionales usando objetos específicos del dominio de la aplicación.

Entity Framework crea un Modelo de Datos de Entidad que permite codificar contra clases denominadas Entidades, en vez de hacerlo contra esquemas y objetos de base de datos como las tablas [22].

El Modelo de Datos de Entidad consiste de 3 capas separadas [22]:

- Conceptual: Contiene las clases entidades del modelo de datos de entidad. La capa conceptual puede ser modelada con un diseñador o con código.
- Almacenamiento (*Store*): Define las tablas, columnas, asociaciones y tipos de datos que mapean hacia la base de datos.
- Asociación (*Mapping*): Define el mapeo entre la capa conceptual y la capa de almacenamiento. Entre otras cosas, define las propiedades de las clases entidades mapean hacia las columnas en las tablas de la base de datos.

1.2.9.1 Database first (Base de datos primero)

Este es uno de los enfoques que maneja *Entity Framework*, y es ampliamente usado. Implica generar un modelo de aplicación basado en una base de datos existente. Las entidades son auto generadas en base a las tablas, vistas y procedimientos almacenados seleccionados de la base de datos existente.

1.2.9.2 Code first (Código primero)

El enfoque Code First es una utilidad de *Entity Framework* que permite crear manualmente las clases entidades antes de tener construida la base de datos. Una vez creadas las clases y sus asociaciones en código C#, estas pueden ser

fácilmente procesadas para construir una nueva base de datos o actualizar una existente.

1.2.9.3 LINQ (Language-Integrated Query)

En un *framework* de Microsoft que permite entre otras cosas generar consultas a la base de datos, programando contra las clases entidades y sus asociaciones. LINQ permite expresar conceptos de base de datos directamente usando el código C#.

El esfuerzo de crear código para realizar operaciones sobre las bases de datos se ve drásticamente reducido al usar LINQ [22].

1.2.10 ASP.NET IDENTITY

El *framework* ASP.NET Identity de Microsoft provee un sistema de manejo de todo lo correspondiente a la identidad de un usuario para implementar autenticación y autorización. ASP.NET Identity puede ser usado con otros *frameworks* como ASP.NET MVC.

1.2.11 VISUAL STUDIO TEAM SERVICES

Es un producto de Microsoft que permite subir y compartir código usando la nube. Posee varias funcionalidades como manejo de versionamiento, crear un *product backlog*, integración con Visual Studio, etc.

Visual Studio Team Services es gratuita para pequeños equipos de hasta 5 personas con acceso a funcionalidad básica.

1.2.12 MICROSOFT AZURE

Es una plataforma proveedora de servicios informáticos en la nube. Uno de los servicios que ofrece es PAAS (*Platform as a Service*) o plataforma como servicio. Una plataforma como servicio permite instalar aplicaciones web o servicios WCF en la nube de una manera simple. Visual Studio permite instalar herramientas para la publicación de aplicaciones web a través de Microsoft Azure.

1.3 SCRUM

SCRUM es un método para desarrollar productos de una manera ágil. Es ampliamente usado para el desarrollo de software, pero no es su único fin, ya que puede ser implementado en otros tipos de negocio.

SCRUM [23] se define como “un *framework* dentro del cual las personas pueden alcanzar problemas adaptivos complejos, mientras se entregan productos productivamente y creativamente con el valor más alto posible”.

SCRUM consiste en equipos y sus respectivos roles, eventos, artefactos y reglas. Cada parte tiene un propósito específico el cual debe cumplirse para lograr que SCRUM funcione.

SCRUM está basado en un proceso empírico (basado en la práctica y la experiencia). Los 3 pilares sobre la que se sustenta el proceso empírico de SCRUM son:

- Transparencia: Aspectos importantes del proceso deben ser visible para los responsables del resultado.
- Inspección: Los involucrados en SCRUM deben poder inspeccionar el progreso del trabajo para saber si se está llegando al objetivo.
- Adaptación: Durante el proceso las cosas pueden ser cambiadas si es necesario, siempre y cuando esté dentro de los límites permitidos. SCRUM define eventos para la inspección y adaptación.

1.3.1 EL EQUIPO SCRUM

Los miembros del equipo SCRUM son: el dueño del producto (*product owner*), el equipo de desarrollo y el SCRUM master [23]. A continuación, una breve descripción de cada uno de ellos:

- *Product owner*: Es responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo. También es responsable de manejar el *Product Backlog*.
- Equipo de desarrollo: Son los que realizan el trabajo necesario para la terminación de entregables al final del *Sprint*. SCRUM recomienda que el equipo de desarrollo se empodere de la organización y administración de su propio trabajo.
- *Scrum master*: Es responsable de asegurarse que el equipo entiende el proceso Scrum. Además, ayuda a que el equipo SCRUM se adhiera a las prácticas y reglas del proceso SCRUM.

1.3.2 EVENTOS SCRUM

1.3.2.1 Sprint

Sprint es un período de tiempo menor o igual a 1 mes, y de acuerdo con [23] es el corazón de Scrum. Un nuevo *sprint* inicia inmediatamente después de que el previo *sprint* finalizó.

Algunos de los eventos con lo que cuenta el *sprint* son: *sprint planning* (planificación del *sprint*), y *sprint retrospective* (retrospectivo de *sprint*).

1.3.2.2 Sprint planning

Durante este evento se planifica el trabajo a realizarse durante el *sprint* con la ayuda de todo el equipo Scrum. Tiene un máximo de 8 horas para un *sprint* de un mes, y usualmente es menor para un *sprint* de menor tiempo.

Durante el *sprint planning* se resuelve las siguientes preguntas:

- ¿Qué se puede entregar en el incremento[17] resultante del siguiente *sprint*?
- ¿De qué manera se logrará completar el trabajo necesario para entregar el incremento en el siguiente *sprint*?

La meta del *sprint* se da a través de la implementación del *product backlog*, ya que este, indica al equipo de desarrollo el avance del incremento del producto a entregar en el siguiente *sprint*. En la planificación del *sprint* se seleccionan los ítems del *product backlog* que se puedan terminar durante el *sprint*.

1.3.3 ARTEFACTOS SCRUM

1.3.3.1 Product backlog

Es una lista ordenada de todo lo que se pueda necesitar en el producto. También es una fuente de requerimientos para cualquier cambio a aplicarse sobre el producto. El *product owner* es el responsable de su contenido.

El *product backlog* es dinámico, constantemente cambia a medida que van apareciendo nuevas necesidades del producto. El *product backlog* lista todas las

---

[17] Un incremento en SCRUM es la suma de todos los ítems completados durante un *sprint* y el valor de los incrementos de todos los *sprints* anteriores.

funcionalidades, funciones, requerimientos, mejoras y correcciones sobre el producto que se harán en el futuro.

1.3.3.2 Sprint backlog

Es un conjunto de ítems seleccionados para un *sprint*. Los ítems que conforman el *sprint* backlog son tomados del *product backlog*. La selección de los ítems que conforman el *sprint backlog* va de la mano con un plan de entrega de un incremento al final del *sprint*, es decir, se toma los ítems necesarios del *product backlog* siempre y cuando puedan ser terminados en el tiempo que dura un *sprint*.

Durante el *sprint planning* se selecciona los ítems que formarán parte del *sprint backlog*.

1.4 PRUEBAS

1.4.1 PRUEBAS UNITARIAS

De acuerdo con [24], prueba unitaria o prueba de módulo se define como: “el proceso de probar subprogramas individuales, subrutinas, o procedimientos en un programa”.

La prueba unitaria se enfoca en pequeñas unidades del programa y facilita el proceso de encontrar y corregir errores, ya que, si se encuentra un error, se sabrá que este existe dentro de una pequeña unidad de software.

1.4.2 PRUEBAS DE ORDEN SUPERIOR

Las pruebas de orden superior se basan en el siguiente enunciado [24]:

“Un error de software ocurre cuando el programa no hace lo que el usuario final razonablemente espera que haga y cuando hace lo que el usuario final razonablemente espera que no haga”.

Dentro de la clasificación de pruebas de orden superior, están las pruebas de función, de sistema y de aceptación.

1.4.2.1 Pruebas de función

Es el proceso de tratar de encontrar diferencias entre el programa y la especificación externa (descripción del comportamiento del programa desde el punto de vista del usuario final).

1.4.2.2 Pruebas de sistema

Consiste en comparar el sistema con sus objetivos originales. Según [24], las pruebas de sistema suelen ser confundidas con las pruebas de función, pero no son lo mismo. Las pruebas de función consisten en probar la funcionalidad completa del sistema mientras que las pruebas del sistema se enfocan en los objetivos del programa. Sin objetivos, no se puede realizar las pruebas del sistema.

1.4.2.3 Pruebas de aceptación

Es el proceso de comparar el programa con sus requerimientos originales y las necesidades actuales del usuario final. Según [24] este tipo de prueba es realizada por la persona que va a usar el programa o por el usuario final, normalmente no se considera como responsabilidad de quién desarrolló el programa.

CAPÍTULO 2. DISEÑO DEL SISTEMA

2.1 INTRODUCCIÓN

En el presente capítulo se determinarán los requerimientos del sistema, se detallará el diseño del mismo, especificando cada uno de los componentes que lo conforman. Los componentes del sistema son: bases de datos, capa de acceso a datos, capa de negocio, capa de servicios, capa de presentación, capa de autenticación y autorización, y capa de *logs* y manejo de excepciones. Durante el desarrollo del sistema se seguirán los principios de la metodología SCRUM por lo que se especificará el *product backlog* y los *sprints* necesarios.

2.2 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

Los requerimientos propuestos para el sistema a desarrollarse son los siguientes:

- El sistema contará con 3 bases de datos: una para el manejo de información médica, otra para el manejo de usuarios y una tercera para el manejo de *logs*.
- La base de datos para el manejo de información médica será provista por la empresa Visonex Technologies.
- El sistema contará con una aplicación web construida con el *framework* de MVC.NET.
- El sistema permitirá la autenticación de usuarios a través de una de sus interfaces de usuario.
- El sistema desplegará interfaces de usuario basadas en web.
- El sistema permitirá desplegar información médica a través de sus interfaces de usuario.
- El sistema implementará autorización de usuarios a través de roles de usuario.
- El sistema permitirá añadir o quitar roles a los usuarios a través de una de sus interfaces de usuario.
- Las interfaces de usuario del sistema serán responsivas[18].

[18] Diseño web responsivo o adaptable es un enfoque de diseño y desarrollo cuyo objetivo es que la apariencia de las páginas web se adapte al tamaño de la pantalla del dispositivo desde donde son accedidas.

- El sistema tendrá una interfaz de usuario para el registro de usuarios.
- El sistema permitirá generar reportes con información médica en formato PDF.
- El sistema contará con 4 servicios web WCF: uno para acceder a la información médica de pacientes, otro para acceder a la información de médicos, otro para acceder a la información de generación de reportes, y un cuarto para generación de *logs*.
- La aplicación web consumirá los servicios WCF tanto para acceder a la información médica de pacientes, como para acceder a la información de médicos.
- El sistema tendrá un componente desacoplado para la creación de *logs*, el cual se conectará al servicio web de *logs* cada vez que se requiera insertar registros en la base de datos de *logs*.
- El sistema proveerá de un mecanismo de seguridad apropiado para proteger el acceso a los servicios web de información médica de pacientes, información de médicos, e información para generación de reportes.
- El sistema tendrá una página de inicio que podrá ser accedida una vez que un usuario haya iniciado sesión, esta desplegará alertas del usuario.
- Las interfaces de usuario desplegarán un menú de acceso hacia otras interfaces una vez que el usuario haya iniciado sesión.
- El sistema generará *logs* ante varios tipos de eventos que sean considerados de importancia, y serán almacenados en la base de datos de *logs*.
- El sistema almacenará información sobre las excepciones que ocurran durante la ejecución en la base de datos de *logs*.
- Las interfaces de usuario del sistema solo soportarán el idioma inglés.
- Las interfaces de usuario del sistema deberán ser soportadas en al menos dos exploradores web: Google Chrome e Internet Explorer.
- Las bases de datos serán de tipo SQL *server*.

2.3 DISEÑO DEL SISTEMA

En esta sección se presentará de forma resumida el diseño de cada uno de los componentes del Sistema. Primero se presentará el diseño arquitectural que se requiere para cumplir con los requerimientos del sistema, luego se analizará cada

uno de los componentes o capas que conforman el diseño arquitectural, y finalmente se presentará el diseño de interfaces de usuario y de la lógica de presentación.

2.3.1 DISEÑO ARQUITECTURAL

El sistema contará con los siguientes componentes:

- Capa de acceso a datos: Se encargará del manejo de la persistencia de información médica. Esta capa tendrá elementos responsables de proveer acceso a la base de datos con información médica.
- Capa de negocio: Se encargará de la lógica de negocio. Esta capa tendrá los elementos responsables de realizar algún tipo de procesamiento de negocio y la aplicación de las reglas de negocio.
- Capa de servicios: Esta capa tendrá 3 servicios WCF que expondrán la funcionalidad de negocio relacionada al manejo de la información médica.
- Capa de presentación: Se encargará del manejo de la lógica de presentación. Será construida usando el *framework* de MVC.NET la cual será el cliente de los servicios WCF de la capa de servicios. Esta capa tendrá los elementos responsables de proveer mecanismos de comunicación entre los usuarios y el sistema, como son las interfaces de usuario.
- Capa de *logs* y manejo de excepciones: Se encargará de la generación y almacenamiento de *logs* en la base de datos de *logs*. Esta capa incluye la creación de un servicio web para la generación de *logs*, y la funcionalidad necesaria para implementar el guardado de *logs* en la base de datos de *logs*. Se almacenarán *logs* de información, de alertas y de errores del sistema.
- Capa de autenticación y autorización: Se encargará del manejo de la información de autenticación y autorización de usuarios.
- Base de datos con información médica: Tendrá toda la información médica asociada a los pacientes, así como también la información de médicos.
- Base de datos para manejo de autenticación y autorización: Tendrá toda la información pertinente al manejo de usuarios y roles para la implementación de autenticación y autorización en el sistema.
- Base de datos para manejo de *logs*: Aquí se almacenarán todos los *logs* generados desde las diferentes capas o componente del sistema.

La **Figura 2.1** presenta el diagrama arquitectural de capas del sistema y la interacción con cada una de las bases de datos. También se muestra la interacción de las capas transversales de autenticación y autorización, y de generación de *logs* con las capas.

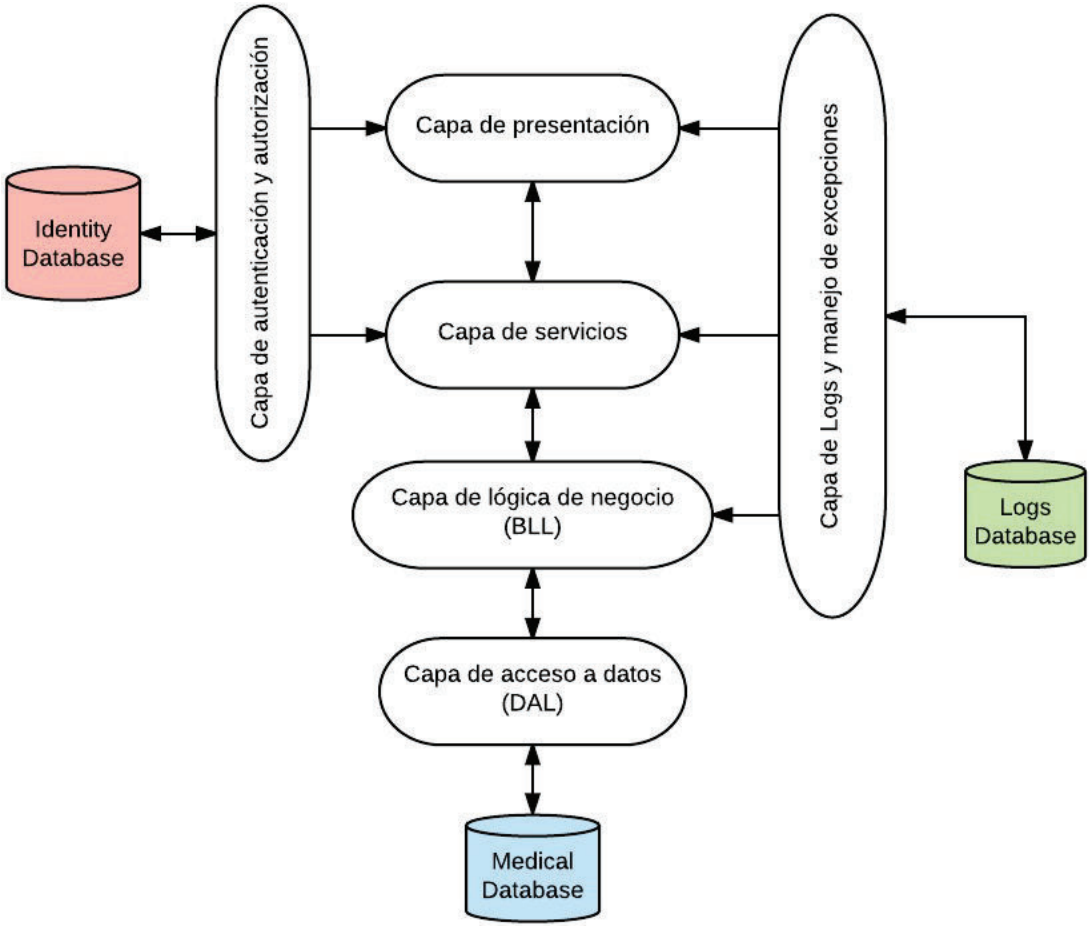


Figura 2.1 Diagrama arquitectural del sistema e interacción de las capas con las bases de datos

2.3.1.1 Análisis de base de datos con información médica

La base de datos con información médica será denominada base de datos médica y como `MedicalDatabase` dentro del sistema.

Las tablas que forman parte de la base de datos médica y una breve descripción del contenido de información de cada una de ellas se muestra en la **Tabla 2.1**. Los nombres de las tablas y de sus campos están en el idioma inglés, así como los datos de prueba provistos por la empresa Visonex Technologies.

Tabla 2.1 Tablas de la base de datos médica

Nombre de tabla	Descripción
`TestPatient`	Contiene los registros de cada paciente. Se almacenan varios datos generales del paciente, como nombres, fecha de nacimiento, información de contacto, entre otros.
`TestPhysician`	Contiene los registros de médicos. Se almacena información general del médico, como nombres, especialidad médica, información de contacto, entre otros.
`Allergy`	Contiene los registros médicos de alergias que tiene cada paciente.
`VascularAccessDevice`	Contiene los registros médicos de accesos vasculares[19] que tiene cada paciente.
`Infection`	Contiene los registros médicos de infecciones[20] que posee cada paciente.
`Immunization`	Contiene los registros médicos de inmunizaciones[21] que posee cada paciente.
`Treatment`	Contiene los registros médicos de los tratamientos que tiene cada paciente.
`Medication`	Contiene los registros de los medicamentos de un paciente.

La tabla `TestPatient` no registra información médica del paciente, pero sí su información general, es decir, información de contacto, edad, nombre, etc. La **Tabla 2.2** muestra una breve descripción de los campos de la tabla `TestPatient`.

[19] Un acceso vascular es aquel que permite el ingreso y salida de sangre del cuerpo humano. Un acceso vascular permite a los doctores realizar los tratamientos de diálisis renal del paciente.
[20] Una infección se refiere a la invasión y multiplicación de microorganismos en un área del cuerpo del paciente.
[21] Inmunización es la acción y efecto de inmunizar, es decir, hacer que el paciente no sea vulnerable ante ciertas enfermedades.

Tabla 2.2 Campos de la tabla `TestPatient`

Columna	Tipo de dato	Descripción
TestPatientID	int	Clave primaria
FirstName	varchar	Primer nombre del paciente
MiddleName	varchar	Segundo nombre del paciente
LastName	varchar	Apellido del paciente
Address1	varchar	Dirección línea 1
Address2	varchar	Dirección línea 2
City	varchar	Ciudad
State	varchar	Estado
ZipCode	varchar	Código postal
Phone	varchar	Teléfono
Email	varchar	Dirección electrónica
BirthDate	smalldatetime	Fecha de nacimiento
TestPatientHeight	varchar	Talla del paciente (pies)
TestPatientHeightInches	varchar	Talla del paciente (pulgadas)
SocialSecurityNumber	varchar	Número de seguridad social
Status	varchar	Estado del paciente
DateFirstDialysis	smalldatetime	Fecha de la primera diálisis
Ethnicity	varchar	Etnia
Race	varchar	Raza
DeathDate	smalldatetime	Fecha de deceso
Sex	varchar	Sexo
MaritalStatus	varchar	Estado civil
FistulaFirstStartDate	datetime	Fecha de la primera *Fistula*
Suffix	varchar	Sufijo
Prefix	varchar	Prefijo
Degree	varchar	Título
AliasNickname	varchar	Alias
PrimaryLanguage	varchar	Lengua primaria
EthnicityReported	varchar	Etnia reportada
CitizenshipStatus	varchar	Estado de la ciudadanía
CitizenshipStatusEffectiveDate	datetime	Fecha efectiva del estado de ciudadanía
AddedDate	datetime	Fecha de creación del registro
EditDate	datetime	Fecha de la última modificación del registro

Columna	Tipo de dato	Descripción
AddedUser	int	ID del usuario quien añadió el registro
EditUser	int	ID del usuario que realizó la última modificación del registro
Labidnumber	varchar	Clave foránea

La tabla TestPhysician registra información general de médicos, es decir, nombres, dirección, información de contacto, entre otros. La **Tabla 2.3** muestra una breve descripción de los campos de la tabla TestPhysician.

La tabla Allergy registra información relacionada a las alergias de los pacientes, por ejemplo, el nombre del medicamento al cual el paciente es alérgico, y el tipo de reacción. La **Tabla 2.4** muestra una breve descripción de los campos de la tabla Allergy.

La tabla VascularAccessDevice contiene registros de los accesos vasculares de los pacientes. Un acceso vascular es un elemento indispensable para la realización de tratamientos de diálisis[22] en pacientes. Los accesos vasculares pueden infectarse con el tiempo, es por ello que estos pueden expirar.

La tabla VascularAccessDevice contiene información relevante respecto a los accesos vasculares que un paciente tiene o ha tenido, por ejemplo, el tipo de acceso vascular, el estado actual del acceso vascular, la ubicación del acceso vascular en el cuerpo, la fecha de inicio del acceso vascular, etc. La **Tabla 2.5** muestra una breve descripción de los campos de la tabla VascularAccessDevice.

La tabla Infection contiene registros de las infecciones que se pueden dar dentro de un acceso vascular de un paciente. Debido a que un acceso vascular es la puerta de entrada hacia el cuerpo humano, es muy común que ahí se produzcan infecciones.

[22] La diálisis renal o tratamiento de diálisis renal es el proceso de retirar la sangre del cuerpo humano, y de una manera artificial, simular el proceso que realizan los riñones sobre la misma, para luego devolverla al torrente sanguíneo del paciente.

La tabla `Infection` registra información relacionada a las infecciones, por ejemplo, el tipo de fuente de la infección, los síntomas, la fecha en que inició la infección, etc. La **Tabla 2.6** muestra una breve descripción de los campos de la tabla `Infection`.

Tabla 2.3 Campos de la tabla `TestPhysician`

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`Lastname`	`varchar`	Apellido del médico
`Middlename`	`varchar`	Segundo nombre del médico
`Firstname`	`varchar`	Primer nombre del médico
`Degree`	`varchar`	Título del médico
`MedicareIDNumber`	`varchar`	Identificador único del médico dentro de la red médica de Estados Unidos
`Address`	`varchar`	Dirección
`City`	`varchar`	Ciudad
`State`	`varchar`	Estado
`ZipCode`	`varchar`	Código postal
`Phone`	`varchar`	Teléfono
`Fax`	`varchar`	Fax
`HomePhone`	`varchar`	Teléfono de casa
`MobilePhone`	`varchar`	Teléfono celular
`Pager`	`varchar`	Número de *pager*[23]
`Email`	`varchar`	Dirección electrónica
`Specialty`	`varchar`	Especialidad
`TestPhysicianSignsOrdersElectronically`	`bit`	¿Puede el médico firmar electrónicamente?
`AddedDate`	`datetime`	Fecha de creación del registro
`EditDate`	`datetime`	Fecha de la última modificación del registro
`AddedUser`	`int`	ID del usuario quien añadió el registro
`EditUser`	`int`	ID del usuario que realizó la última modificación del registro

[23] *Pager* es un dispositivo de comunicación usado por los médicos dentro de hospitales.

Tabla 2.4 Campos de la tabla `Allergy`

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea. Indica el identificador único del paciente quien posee la alergia.
`MedID`	`int`	Clave foránea
`MedicationName`	`varchar`	Nombre del medicamento
`TypeofReaction`	`varchar`	Tipo de reacción alérgica
`StartDate`	`datetime`	Fecha en que inició la alergia
`EndDate`	`datetime`	Fecha en que finalizó la alergia
`DRUG_ID`	`varchar`	Clave foránea
`AddedDate`	`datetime`	Fecha de creación del registro
`EditDate`	`datetime`	Fecha de la última modificación del registro
`AddedUser`	`int`	ID del usuario quien añadió el registro
`EditUser`	`int`	ID del usuario que realizó la última modificación del registro

Tabla 2.5 Campos de la tabla `VascularAccessDevice`

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea. Indica el identificador único del paciente quien posee el acceso vascular.
`VascularAccessType`	`varchar`	Tipo de acceso vascular
`Location`	`varchar`	Ubicación del acceso vascular
`CurrentStatus`	`varchar`	Actual estatus del acceso vascular
`ChronicCatheter`	`bit`	¿El acceso vascular es catéter crónico?
`Comments`	`varchar`	Comentarios
`StartDate`	`smalldatetime`	Fecha de inicio del acceso vascular
`EndDate`	`smalldatetime`	Fecha de finalización del acceso vascular
`AddedDate`	`datetime`	Fecha de creación del registro
`EditDate`	`datetime`	Fecha de la última modificación del registro
`AddedUser`	`int`	ID del usuario quien añadió el registro
`EditUser`	`int`	ID del usuario que realizó la última modificación del registro

Tabla 2.6 Campos de la tabla `Infection`

Columna	Tipo de dato	Descripción
ID	int	Clave primaria
PatientID	int	Clave foránea. Indica el identificador único del paciente quien tuvo la infección.
SuspectedInfectionStartDate	date	Fecha cuando se sospecha inició la infección
ConfirmationDate	date	Fecha cuando se confirmó que existe la infección
InfectionSourceType	varchar	Tipo de fuente de infección
VascularAccessDeviceIDNumber	int	Clave foránea. Indica el identificador único del acceso vascular asociado al registro de infección.
PrimaryLocationOther	varchar	Otra ubicación primaria del acceso vascular asociado a la infección
IdentificationType	varchar	Tipo de identificación usado para determinar la infección
ObservedByUserIDNumber	int	Identificador único del usuario que observó la infección. Clave foránea.
Symptom_NONE	bit	El paciente no tuvo síntomas
Symptom_AccessSite	bit	El paciente tuvo síntomas en el área de acceso vascular
Symptom_Fever_37_8c	bit	El paciente tuvo fiebre mayor a 37.8C
Symptom_Chills	bit	El paciente tuvo resfriado
Symptom_BP	bit	El paciente tuvo presión alta
Symptom_Mental	bit	El paciente tuvo síntomas de problemas mentales
Symptom_Wound	bit	El paciente tuvo una herida
Symptom_Cellulitis	bit	El paciente tuvo síntoma de celulitis
Symptom_Respiratory	bit	El paciente tuvo síntoma de problema respiratorio
Symptom_Other	varchar	El paciente tuvo otro síntoma
Comments	varchar	Comentarios respecto a la infección
EndDate	date	Fecha cuando terminó la infección

Columna	Tipo de dato	Descripción
InfectionRequiredHospitalization	varchar	¿Requirió hospitalización por infección?
HospitalizationDate	date	Fecha de hospitalización
LossOfVascularAccess	varchar	¿Pérdida de acceso vascular?
AccessSiteNHSN_Fistula	bit	Acceso vascular tipo *Fistula*[24]
AccessSiteNHSN_Graft	bit	Acceso vascular tipo *Graft*[25]
AccessSiteNHSN_ TunnCentralLine	bit	Acceso vascular tipo *Tunneled Central Line*[26]
AccessSiteNHSN_ NonTunnCentralLine	bit	Acceso vascular de tipo diferente a *Tunneled Central Line*
AccessSiteNHSN_ OtherAccessDevice	bit	Acceso vascular de otro tipo
AddedDate	datetime	Fecha de creación del registro
EditDate	datetime	Fecha de la última modificación del registro
AddedUser	int	ID del usuario quien añadió el registro
EditUser	int	ID del usuario que realizó la última modificación del registro

La tabla Immunization contiene registros de las inmunizaciones agendadas a un paciente. Una manera de inmunizar a los pacientes es a través de vacunas, y cada orden de vacunación que el paciente recibe por su médico es registrada en esta tabla.

La tabla Immunization registra información relacionada a las inmunizaciones, como el nombre de la misma, la fecha en que el paciente está supuesto a recibirla, la fecha de expiración de la orden de inmunización, etc. La **Tabla 2.7** muestra una breve descripción de los campos de la tabla Immunization.

[24] Técnica que consiste en conectar una arteria con una vena en una extremidad del cuerpo, usado para proveer acceso para realizar diálisis.
[25] Técnica que consiste en insertar un implante bajo la piel que permite conectar la arteria con la vena, y que provee un acceso para realizar diálisis.
[26] Técnica que consiste en la implementación de un catéter, y provee acceso para diálisis.

Tabla 2.7 Campos de la tabla `Immunization`

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea que corresponde al identificador único del paciente a quien se agendó la inmunización.
`Immunization`	`varchar`	Describe la inmunización
`ImmunizationDate`	`smalldatetime`	Fecha en la que se realizó la inmunización
`AdministeredOnDialysis`	`bit`	Fue o no administrada durante diálisis
`Scheduled`	`bit`	La inmunización fue agendada o no por el médico
`Refused`	`bit`	La inmunización fue rechazada o no por el médico
`RunIDNumber`	`int`	Clave foránea a tabla desconocida
`MedicationCode`	`varchar`	Clave foránea a tabla desconocida
`Status`	`varchar`	Estado de la inmunización
`ExpirationDate`	`datetime`	Fecha de expiración de la inmunización
`ImmunizationRoute`	`varchar`	Ruta de inmunización[27]
`TreatmentIDNumber`	`int`	Clave foránea que corresponde al identificador único del tratamiento de un paciente.
`AddedDate`	`datetime`	Fecha de creación del registro
`EditDate`	`datetime`	Fecha de la última modificación del registro
`AddedUser`	`int`	ID del usuario quien añadió el registro
`EditUser`	`int`	ID del usuario que realizó la última modificación del registro

[27] Una ruta de inmunización se refiere a la vía en que el paciente recibe la inmunización, por ejemplo, en el caso de tratarse de una vacuna intramuscular, la ruta de la inmunización es intramuscular o IM.

La tabla `Treatment` contiene registros de los tratamientos de diálisis renal que recibe un paciente y registra información relacionada a dichos tratamientos, como los identificadores únicos de la estación de diálisis y el identificador único de la máquina de diálisis donde el paciente se realiza el tratamiento, además, información del peso del paciente antes y después del tratamiento, valores de los signos vitales antes y después del tratamiento, etc. La **Tabla 2.8** muestra una breve descripción de los campos de la tabla `Treatment`.

La tabla `Medication` contiene registros de los medicamentos prescritos a un paciente. Cada registro contiene información relacionada al medicamento, por ejemplo, el nombre del medicamento, la dosis, la prescripción, fecha de inicio y fin del medicamento, etc. La **Tabla 2.9** muestra una breve descripción de los campos de la tabla `Medication`.

La base de datos médica no posee tablas para almacenar información de usuarios del sistema, ni tampoco información de roles.

Para mantener separada la información médica de la información de autenticación y autorización, se diseñará la base de autenticación y autorización, donde se podrá almacenar información relacionada a los usuarios del sistema, como nombre de usuario, contraseñas, roles, etc. De esta manera, la base de datos médica no requiere ser modificada en su estructura.

En la **Figura 2.2**, la **Figura 2.3** y la **Figura 2.4** se muestra el modelo relacional de la base de datos médica.

Existe una relación de uno a muchos entre la tabla `TestPatient` y las tablas: `Immunization`, `Medication`, `Treatment`, `VascularAccessDevice` y `Allergy`, a través de una clave externa (*foreign key*). Es decir, por cada registro `TestPatient`, podrá existir de uno a muchos registros `Immunization`, `Medication`, `Treatment`, `VascularAccessDevice` y `Allergy` asociados.

Además, la tabla `VascularAccessDevice` tiene una relación de uno a muchos con la tabla `Infection` a través de una clave externa, es decir, por cada registro de acceso vascular, podrá existir de uno a muchos registros de infecciones, esto se debe a que los accesos vasculares son fuentes de infecciones.

Tabla 2.8 Campos de la tabla `Treatment`

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea que corresponde al identificador único del paciente quien se realizó el tratamiento.
`StationID`	`varchar`	Identificador único de la estación de diálisis
`MachineID`	`varchar`	Identificador único de la máquina de diálisis
`OutpatientTreatment`	`tinyint`	Clave foránea de tabla desconocida
`TreatmentStart`	`smalldatetime`	Fecha de inicio del tratamiento
`TreatmentEnd`	`smalldatetime`	Fecha de finalización del tratamiento
`AverageBloodFlowRate`	`decimal`	Caudal de sangre promedio
`StartSittingBP`	`varchar`	Presión sanguínea inicial mientras el paciente está sentado
`StartSittingPulse`	`varchar`	Pulso inicial mientras el paciente está sentado
`EndSittingBP`	`varchar`	Presión sanguínea al final del tratamiento mientras el paciente está sentado
`EndSittingPulse`	`varchar`	Pulso al final del tratamiento mientras el paciente está sentado
`StartStandingBP`	`varchar`	Presión sanguínea inicial mientras el paciente está de pie
`StartStandingPulse`	`varchar`	Pulso inicial mientras el paciente está de pie
`EndStandingBP`	`varchar`	Presión sanguínea al final del tratamiento mientras el paciente está de pie
`EndStandingPulse`	`varchar`	Pulso al final del tratamiento mientras el paciente está de pie
`TreatmentHighBP`	`varchar`	Presión sanguínea alta durante el tratamiento
`TreatmentLowBP`	`varchar`	Presión sanguínea baja durante el tratamiento
`DryWeight`	`decimal`	Peso normal del paciente sin ningún fluido extra

Columna	Tipo de dato	Descripción
LastWeight	decimal	Último peso del paciente
PreWeight	decimal	Peso del paciente antes del tratamiento
PostWeight	decimal	Peso del paciente después del tratamiento
LitersProcessed	decimal	Litros de fluido de diálisis procesado
TimeDialyzed	varchar	Tiempo de diálisis
TimeInByPass	varchar	Tiempo conectado a la máquina de diálisis
TimePumpOff	varchar	Tiempo hasta que la bomba de diálisis fue apagada
FluidRemoved	decimal	Cantidad de fluidos removidos después del tratamiento
TreatmentSequenceNumber	varchar	Número de secuencia del tratamiento[28]
InputFile	varchar	Nombre del archivo que contiene información de configuración para la máquina de diálisis.
UnitID	smallint	Identificador único de la máquina de diálisis.
Location	varchar	Ubicación del acceso vascular
PatientStatus	varchar	Estatus del paciente
DialysateTemp	decimal	Temperatura del fluido de diálisis
PatientTempStart	decimal	Temperatura inicial del paciente
PatientTempEnd	decimal	Temperatura final del paciente
PatientTempHigh	decimal	Temperatura alta del paciente durante el tratamiento
TxSubtype	int	Clave foránea de tabla desconocida
InfectionPresent	bit	¿Existe infección durante el tratamiento?
AddedDate	datetime	Fecha de creación del registro
EditDate	datetime	Fecha de la última modificación del registro
AddedUser	int	ID del usuario quien añadió el registro
EditUser	int	ID del usuario que realizó la última modificación del registro

[28] Una sesión de diálisis renal puede incluir varios tratamientos de diálisis. El número de secuencia se refiere al número del tratamiento dentro la sesión de diálisis renal.

Tabla 2.9 Campos de la tabla `Medication`

Columna	Tipo de dato	Descripción
ID	int	Clave primaria
PatientID	int	Clave foránea que corresponde al identificador único del paciente quien se realizó el tratamiento.
MedID	int	Clave foránea de tabla desconocida
Medication	varchar	Descripción del medicamento
DoseStrength	varchar	Dosis del medicamento
DoseForm	varchar	Forma de dosis del medicamento
Prescription	varchar	Prescripción del medicamento
Hold	bit	¿Medicamento en espera?
StartDate	smalldatetime	Fecha de inicio del medicamento
AdminDuringTreatment	bit	¿Administrar durante tratamiento?
ReasonOnHold	varchar	Razón por la cual el medicamento está en espera
TotalDoses	int	Número total de dosis
NurseComments	varchar	Comentarios de la enfermera
DoseQty	decimal	Cantidad de dosis
DoseUnit	varchar	Unidad de dosis
DoseRoute	varchar	Ruta de la dosis
DoseFreq	varchar	Frecuencia de la dosis
DoseFreqMonday	bit	¿Existe dosis el lunes?
DoseFreqTuesday	bit	¿Existe dosis el martes?
DoseFreqWednesday	bit	¿Existe dosis el miércoles?
DoseFreqThursday	bit	¿Existe dosis el jueves?
DoseFreqFriday	bit	¿Existe dosis el viernes?
DoseFreqSaturday	bit	¿Existe dosis el sábado?
DoseFreqSunday	bit	¿Existe dosis el domingo?
PrescriptionFreeText	varchar	Descripción relacionada a la prescripción
DateDoseLastGiven	datetime	Fecha de la última dosis dada
MonthlyDose	decimal	Dosis mensual
SelfAdmin	bit	¿Paciente se auto administra el medicamento?
ICD10	varchar	Código de diagnóstico médico usado en Estados Unidos
GENPRODUCT_ID	int	Clave foránea a tabla desconocida
DRUG_ID	varchar	Clave foránea a tabla desconocida
AddedDate	datetime	Fecha de creación del registro
EditDate	datetime	Fecha de la última modificación del registro
AddedUser	int	ID del usuario quien añadió el registro
EditUser	int	ID del usuario que realizó la última modificación del registro

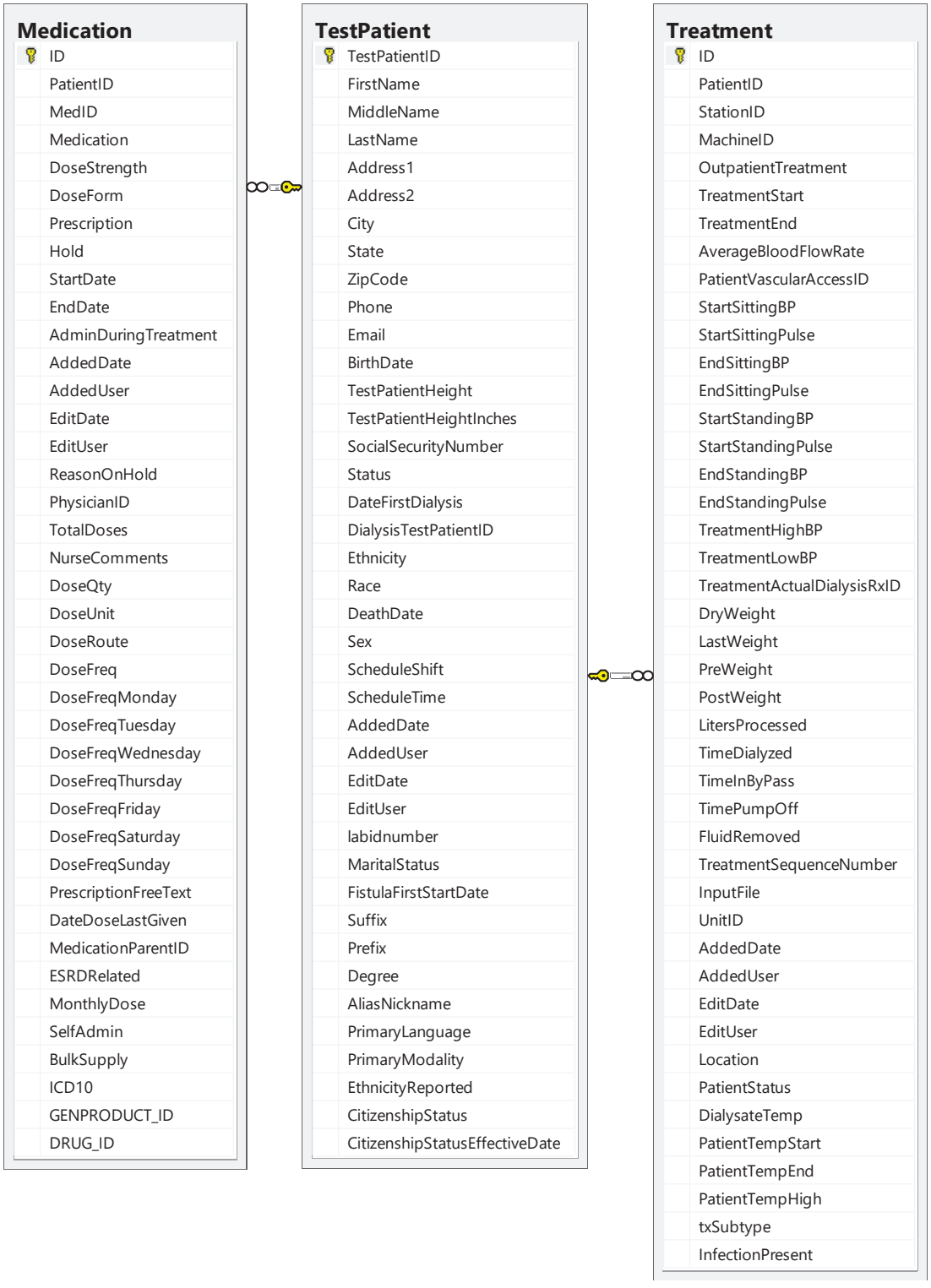


Figura 2.2 Diagrama relacional de base de datos médica parte I

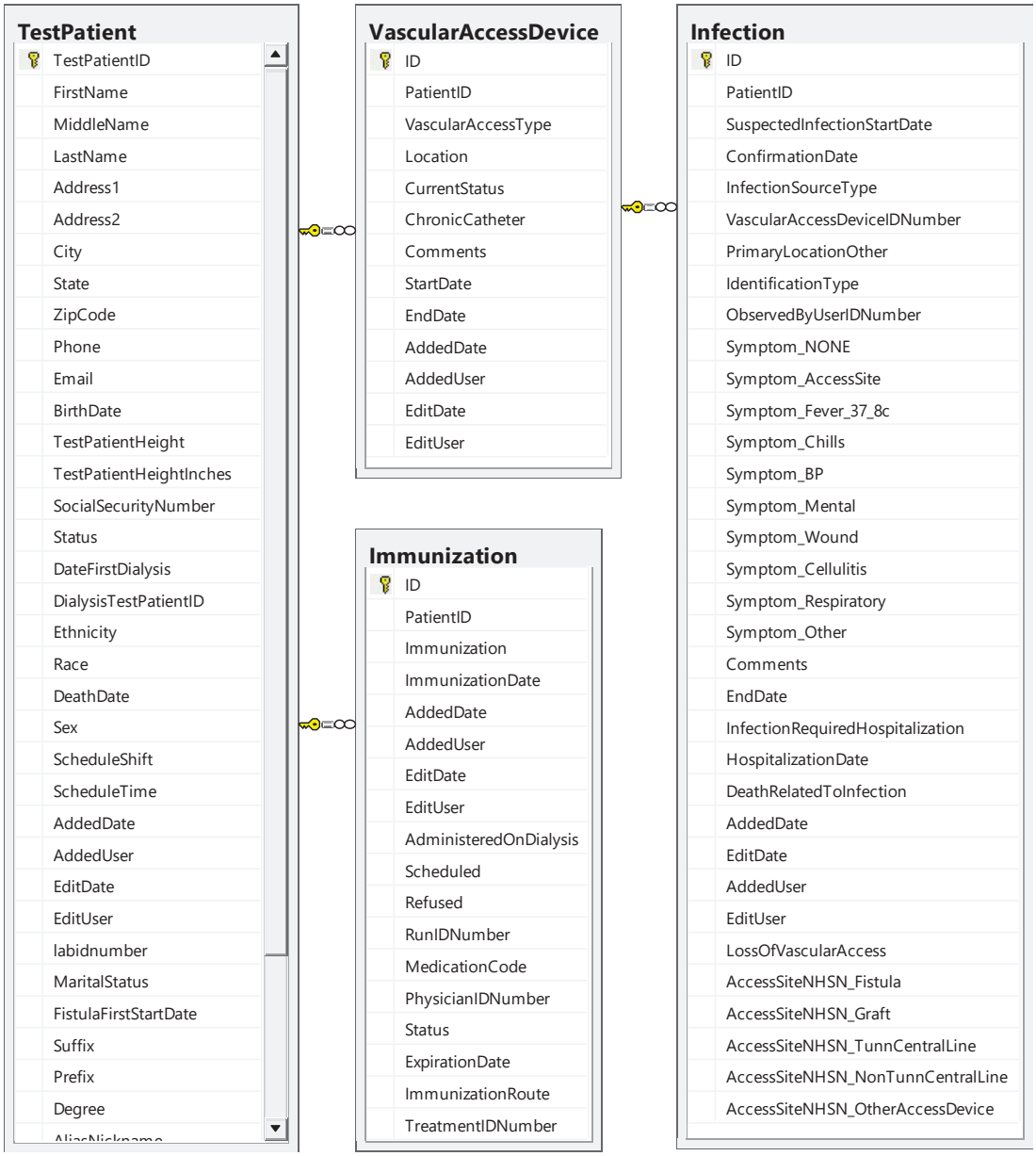


Figura 2.3 Diagrama relacional de base de datos médica parte II

2.3.1.2 Diseño de base de datos de autenticación y autorización

La base de datos de autenticación y autorización permitirá almacenar información de usuarios y de roles de usuario, que son necesarios para la implementación de autenticación y autorización.

Esta base de datos se denomina base de datos identidad y en el sistema se llamará `IdentityDatabase`. La **Figura 2.5** muestra el modelo relacional de la base de datos identidad.

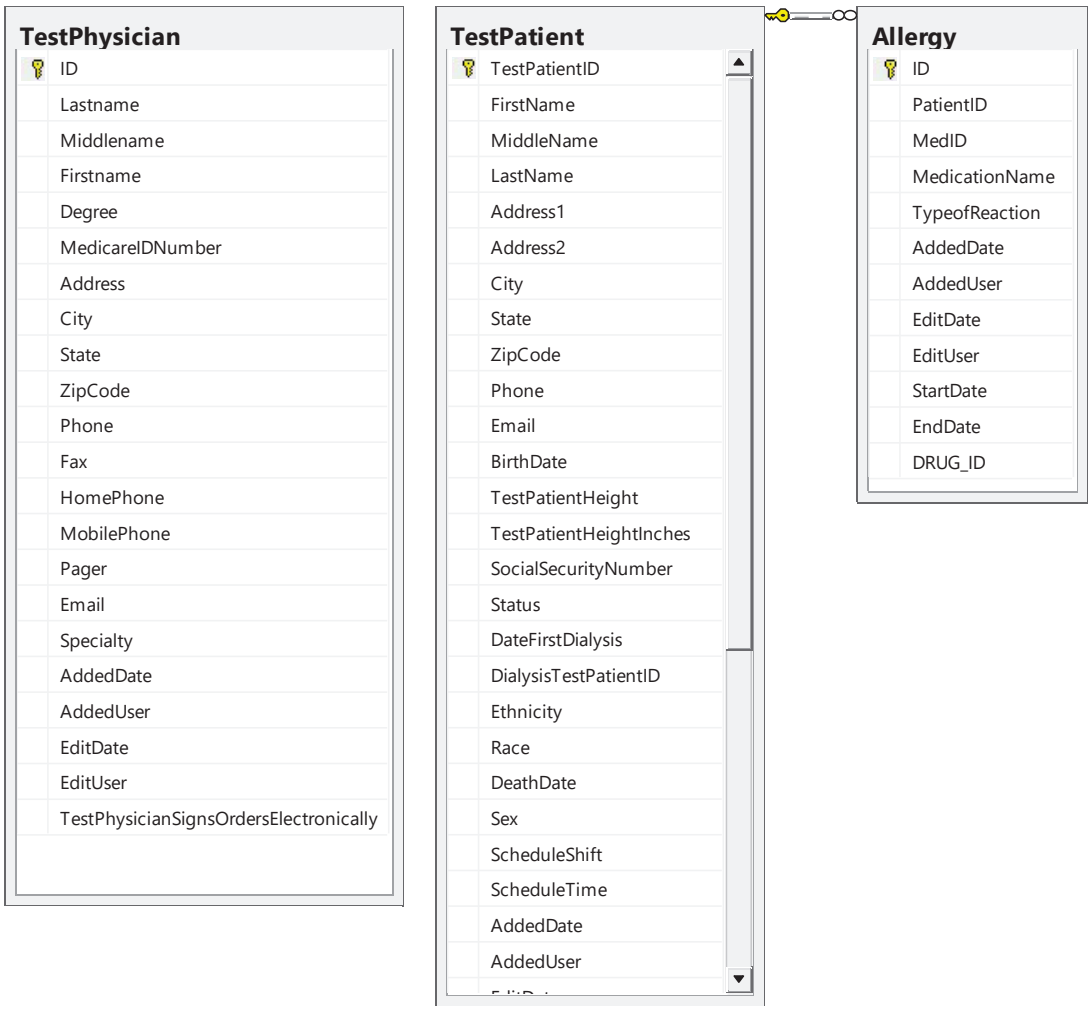


Figura 2.4 Diagrama relacional de base de datos médica parte III

En la tabla `Role` se insertarán todos los registros de roles de usuario permitidos en el sistema. En la tabla `User` se almacenarán los registros de usuarios del sistema, y la tabla `UserRole` permitirá asociar un usuario con uno o varios roles de usuario.

La tabla `User` fue diseñada tomando como base la tabla `AspNetUsers` del *framework* ASP.NET Identity. La **Tabla 2.10** muestra una breve descripción de los campos de la tabla `User` de la base de datos identidad que serán usados.

Se definen 4 tipos de roles de usuario para la implementación de autorización en el sistema, estos son: rol de usuario por defecto (`Default`), rol de médico no autorizado (`PhysicianUnauthorized`), rol de médico autorizado (`Physician`) y rol de administrador (`Admin`). Los registros de los 4 roles de usuario definidos se

insertarán en la tabla `dbo.Role`. En la **Tabla 2.11** se muestra una descripción breve de cada uno de los roles de usuario previamente mencionados.

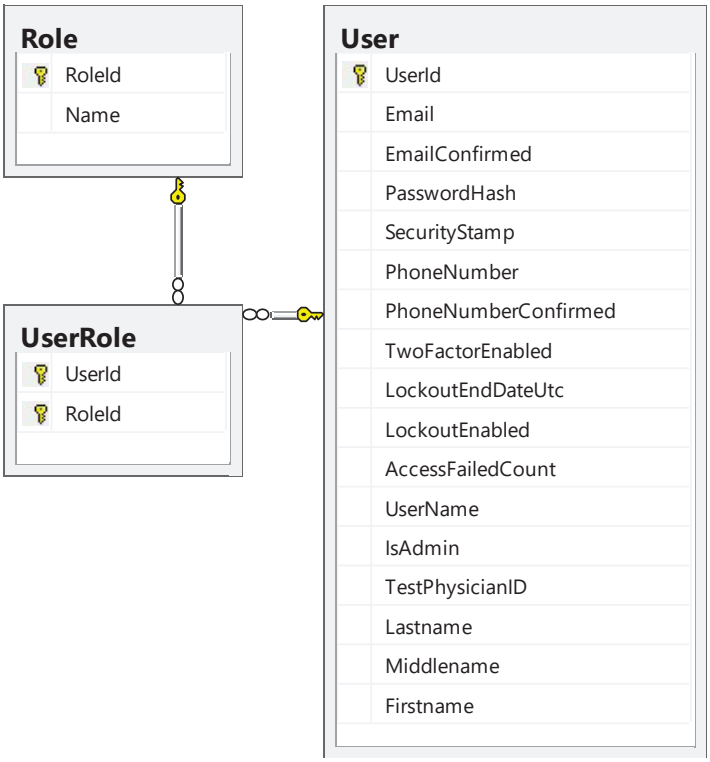


Figura 2.5 Diagrama relacional de base de datos identidad

El sistema permitirá que cualquier usuario pueda registrarse. Un usuario que complete el registro desde la interfaz de usuario del sistema tendrá el rol de usuario `Default`, sin embargo, este usuario tendrá acceso restringido al sistema, ya que solo podrá acceder a la página de inicio. Solo los usuarios del sistema que sean de tipo administrador podrán asignar otros roles de usuario los cuales permitirán el acceso a otras interfaces de usuario.

2.3.1.3 Diseño de base de datos de logs

La base de datos de *logs* será denominada `LogsDatabase` en el sistema. Esta permitirá almacenar *logs* generados desde los diferentes componentes del sistema.

Un *log* es un registro, ya sea en un archivo o en base de datos, de acontecimientos o eventos que suceden en tiempo de ejecución del sistema, algunos de ellos pueden indicar que alguna funcionalidad del sistema falló, en otros casos simplemente se quiere registrar eventos como que un método fue llamado.

Tabla 2.10 Descripción de las columnas de la tabla `User`

Columna	Tipo	¿Permite Null?	Descripción
`UserId`	`uniqueidentifier`	No	Clave primaria e identificador único de los registros
`Email`	`nvarchar(256)`	Si	Correo electrónico del usuario que permite identificar de forma única a cada registro de usuario
`PasswordHash`	`nvarchar(max)`	Si	Creado a partir de la clave de usuario
`SecurityStamp`	`nvarchar(max)`	Si	Campo relacionado a la seguridad y es modificado por el *framework* de ASP.NET Identity. Cada vez que el usuario cambia la contraseña de acceso al sistema, el campo se actualiza.
`PhoneNumber`	`nvarchar(max)`	Si	Número de teléfono del usuario
`UserName`	`nvarchar(512)`	No	Nombre de usuario
`IsAdmin`	`bool`	No	Identifica si el usuario es de tipo administrador
`TestPhysicianID`	`int`	Si	Permite asociar un registro de la tabla `TestPhysician` de la base de datos `MedicalDatabase` con un usuario de la tabla `User`
`Lastname`	`varchar(50)`	Si	Apellido del usuario
`Middlename`	`varchar(50)`	Si	Segundo nombre del usuario
`Firstname`	`varchar(50)`	Si	Primer nombre del usuario

Tabla 2.11 Descripción de los roles de usuario

#	Rol de usuario	Descripción
1	`Default`	Rol inicial de todo usuario que se registra en el sistema.
2	`PhysicianUnauthorized`	Rol inicial de los médicos una vez que se registran en el sistema. No tienen acceso a la información médica a menos que un administrador le conceda el rol `Physician`.
3	`Physician`	Rol de médico que tiene acceso a toda la información médica del sistema, el cual solo puede ser asignado por un usuario de tipo administrador.
4	`Admin`	Rol de usuario de tipo administrador que además de tener acceso a la información médica, tendrá acceso a la administración de usuarios y de roles.

El diagrama de la base de datos de *logs* se muestra en la **Figura 2.6**. Aunque el diagrama se compone de una sola tabla, es importante desacoplar la funcionalidad de *logs* del resto de funcionalidad del sistema, ya que, si llega a haber fallos en otras áreas, la componente desacoplada de *logs* debe seguir registrando los eventos.

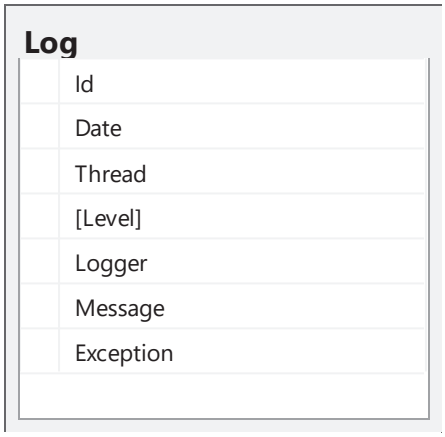


Figura 2.6 Diagrama relacional de la base de datos `LogsDatabase`

La tabla `Log` fue diseñada con base en los requerimientos de la librería Log4net[29]. La **Tabla 2.12** muestra una breve descripción de las columnas de la tabla `Log`.

[29] Log4net es una librería que permite generar declaraciones de *logs* en .NET.

Tabla 2.12 Descripción de las columnas de la tabla `Log`

Columna	Tipo de dato	¿Permite Null?	Descripción
`Id`	`int (identity)`	No	Clave primaria e identificador único de los registros
`Date`	`datetime`	No	Fecha y hora de cuando se originó el *log*
`Thread`	`varchar(255)`	No	Hilo desde donde se origina el *log*
`Level`	`varchar(50)`	No	Nivel de la declaración de *log*
`Logger`	`varchar(255)`	No	La entidad que genera los *logs*
`Message`	`varchar(4000)`	No	El mensaje de la declaración de *log*
`Exception`	`varchar(4000)`	Si	Excepción de la declaración de *log*

Los posibles valores que puede tener el campo `Level` de la tabla `Log` se describen en la **Tabla 2.13**. Log4net establece un orden estandarizado de los niveles de declaración de *logs*, el cual es mostrado también en la **Tabla 2.13**, esto permite definir que *logs* serán registrados en base a su nivel de declaración. Cuando se configura un nivel de declaración de *logs*, se registrará solamente los *logs* del nivel configurado más los *logs* de niveles superiores.

Tabla 2.13 Niveles de declaración de *logs*

Orden	Nivel	Descripción
1	`ALL`	Todos los niveles.
2	`DEBUG`	Designado para eventos que generen información útil durante la depuración de una aplicación.
3	`INFO`	Designado para generar mensajes informativos que resalten el progreso de la aplicación.
4	`WARN`	Designado para situaciones potencialmente dañinas (alertas).
5	`ERROR`	Designado para eventos de error que permitirían que la aplicación se siga ejecutando.
6	`FATAL`	Designado para eventos muy severos que presumiblemente harían detener el funcionamiento de la aplicación.
7	`OFF`	Se destina para desactivar la generación de *logs*.

2.3.1.4 Capa de acceso a datos

La capa de acceso a datos tendrá todos los elementos necesarios para proveer acceso a la información de la base de datos médica. Para facilitar la construcción de la capa de acceso a datos, se usará la tecnología *Entity Framework* que provee la funcionalidad de un ORM, de tal manera que las consultas a la base puedan ser fácilmente implementadas.

Entity Framework permite la creación de entidades de negocio (también llamadas entidades de dominio) tomando como base la estructura de las tablas de la base de datos. La **Tabla 2.14** muestra las entidades que se generarán a partir de las tablas de la base de datos médica.

Tabla 2.14 Tablas de la base de datos médica y su correspondiente entidad de negocio

#	Tabla de la base de datos	Entidad de negocio correspondiente
1	`dbo.Immunization`	`Immunization`
2	`dbo.Medication`	`Medication`
3	`dbo.Treatment`	`Treatment`
4	`dbo.VascularAccessDevice`	`VascularAccessDevice`
5	`dbo.Infection`	`Infection`
6	`dbo.Allergy`	`Allergy`
7	`dbo.TestPatient`	`TestPatient`
8	`dbo.TestPhysician`	`TestPhysician`

En esta capa se implementará el patrón de diseño repositorio. El patrón de diseño repositorio es una representación en memoria de una fuente de datos que trabaja con entidades de negocio [3]. Cada repositorio se construye creando una clase de C#. Un total de 9 repositorios serán implementados en la capa de acceso a datos, es decir uno por cada tabla de la base de datos médica, los cuales suman 8 (a estos se los denominarán repositorios no genéricos), más un repositorio genérico.

Los repositorios son los siguientes: repositorio genérico (`GenericDataRepository`), repositorio de pacientes (`PatientRepository`), repositorio de accesos vasculares (`VascularAccessRepository`), repositorio de

infecciones (`InfectionRepository`), repositorio de medicamentos (`MedicationRepository`), repositorio de tratamientos (`TreatmentRepository`), repositorio de médicos (`PhysicianRepository`), repositorio de alergias (`AllergyRepository`) y repositorio de inmunizaciones (`ImmunizationRepository`). Una breve descripción de cada repositorio se muestra en la **Tabla 2.15**.

Tabla 2.15 Repositorios a implementarse en la capa de acceso a datos

#	Repositorio	Descripción
1	`GenericDataRepository`	Repositorio con operaciones genéricas aplicables para todas las entidades de negocio.
2	`PatientRepository`	Repositorio con operaciones específicas para las entidades de tipo `TestPatient`.
3	`VascularAccessRepository`	Repositorio con operaciones específicas para las entidades de tipo `VascularAccessDevice`.
4	`InfectionRepository`	Repositorio con operaciones específicas para las entidades de tipo `Infection`.
5	`MedicationRepository`	Repositorio con operaciones específicas para las entidades de tipo `Medication`.
6	`TreatmentRepository`	Repositorio con operaciones específicas para las entidades de tipo `Treatment`.
7	`PhysicianRepository`	Repositorio con operaciones específicas para las entidades de tipo `TestPhysician`.
8	`AllergyRepository`	Repositorio con operaciones específicas para las entidades de tipo `Allergy`.
9	`ImmunizationRepository`	Repositorio con operaciones específicas para las entidades de tipo `Immunization`.

Todos los repositorios exceptuando el repositorio genérico `GenericDataRepository`, están relacionados a una tabla de la base de datos médica, y, por ende, a su respectiva entidad de negocio. Los repositorios fueron nombrados con base en los nombres de las entidades a las cuales están asociados.

El repositorio genérico implementará métodos que son aplicables a cualquiera de las entidades de negocio. Por ejemplo, el método `Get`, que devolverá una entidad de negocio. Las clases repositorio no genéricos heredarán de la clase repositorio

genérico `GenericDataRepository`, de esta manera, se centraliza funcionalidad en esa clase, siguiendo el principio DRY[30]. El método `Get` más los otros métodos que se implementarán en el repositorio genérico `GenericDataRepository` se muestran en la **Tabla 2.16**. Los métodos que no son aplicables a todas las entidades de negocio, por ejemplo, el método para obtener todos los tratamientos de un paciente no podrá ser implementado en el repositorio genérico, sino en el repositorio `TreatmentRepository`.

Tabla 2.16 Métodos que se implementarán en la clase `GenericDataRepository`

#	Operación	Descripción
1	`GetAll`	Devolverá todos los registros de la tabla.
2	`GetList`	Devolverá una lista filtrada de registros de la tabla.
3	`Get`	Devolverá un solo registro de la tabla con base en un filtro.
4	`GetById`	Retornará un solo registro de la tabla con base en su clave primaria.
5	`Add`	Inserta una fila en la tabla.
6	`Update`	Actualiza una fila en la tabla.
7	`Remove`	Elimina un registro de la tabla.

Los métodos que se implementarán en los repositorios no genéricos se muestran en la **Tabla 2.17**. El diagrama de clases de los repositorios se mostrará en la sección **2.3.3**.

2.3.1.5 Capa de negocio

La capa de negocio tendrá elementos responsables de realizar procesamiento de negocio e implementación de validaciones de negocio. Las fuentes de datos de la capa de negocio serán los repositorios definidos en la sección anterior.

Como el principal objetivo del sistema será el despliegue de información médica, la mayoría de las operaciones de la capa de negocio devolverán información médica.

---

[30] DRY (*Don't repeat yourself*) es un principio de arquitectura de *software* que promueve la reutilización del código desarrollado, de tal manera que se evita código repetido en diferentes secciones del sistema.

Tabla 2.17 Métodos que se implementarán en los repositorios no genéricos

Repositorio	Método	Descripción
`VascularAccess Repository`	`GetVascularAccessDevicesByPatient`	Devuelve todos los accesos vasculares de un paciente dado.
`Infection Repository`	`GetInfectionsByPatient`	Devuelve todas las infecciones de un paciente dado.
`Medication Repository`	`GetMedicationsByPatient`	Devuelve todos los medicamentos de un paciente dado.
`Treatment Repository`	`GetTreatmentsByPatient`	Devuelve todos los tratamientos de un paciente dado.
`Allergy Repository`	`GetAllergiesByPatient`	Devuelve todas las alergias de un paciente dado.
`Immunization Repository`	`GetImmunizationsByPatient`	Devuelve todas las inmunizaciones de un paciente dado.

En esta capa se implementará la clase `BusinessLayer` y en ella se implementarán los métodos de negocio, los cuales se muestran en la **Tabla 2.18**. El método `WritePhysician` realizará escritura sobre la base de datos médica.

2.3.1.6 Capa de servicios

La capa de servicios alojará 3 servicios web de los 4 a construirse, estos son: `PatientService` que servirá para exponer operaciones[31] relacionadas al despliegue de información médica, `PhysicianService` que expondrá operaciones relacionadas al manejo de médicos y `ReportService` que expondrá operaciones para devolver información para generar reportes. Los 3 servicios WCF tendrán acceso a los métodos definidos en la capa de negocio.

[31] En los servicios WCF, una operación de servicio se la implementa a través de un método en una clase de C#.

Tabla 2.18 Métodos de la capa de negocio

Método	Descripción
`GetAllPatients`	Devuelve todos los pacientes que se encuentran en la tabla `dbo.Patient`.
`GetPatientByID`	Devuelve el paciente con el ID dado.
`GetAllergiesByPatientID`	Devuelve las alergias del paciente dado.
`GetAllergyByID`	Devuelve la alergia con el ID dado.
`GetVascularAccessesByPatientID`	Devuelve los accesos vasculares del paciente dado.
`GetVascularAccessByID`	Devuelve el acceso vascular con el ID dado.
`GetInfectionsByPatientID`	Devuelve las infecciones del paciente dado.
`GetInfectionByID`	Devuelve la infección con el ID dado.
`GetImmunizationsByPatientID`	Devuelve las inmunizaciones del paciente dado.
`GetImmunizationByID`	Devuelve la inmunización con el ID dado.
`GetMedicationsByPatientID`	Devuelve los medicamentos del paciente dado.
`GetMedicationByID`	Devuelve el medicamento con el ID dado.
`GetTreatmentsByPatientID`	Devuelve los tratamientos del paciente dado.
`GetTreatmentByID`	Devuelve el tratamiento con el ID dado.
`GetPhysicianByID`	Devuelve el médico con el ID dado.
`WritePhysician`	Inserta o actualiza un registro de la tabla `dbo.TestPhysician`

La descripción de los tres servicios WCF de esta capa se muestran en la **Tabla 2.19**. Se tomó la decisión de construir tres servicios WCF en vez de 1 solo, tomando en consideración los siguientes aspectos:

- Un servicio WCF tiene autonomía, es decir, que no depende de otros servicios para su funcionamiento. El mal funcionamiento del servicio web `PatientService` no afectará el funcionamiento del servicio web `PhysicianService` y viceversa.
- Un servicio WCF debería contener generalmente más de una operación. Un enfoque podría ser exponer todas las operaciones relacionadas a la base de

datos médica en un solo servicio web, pero se estaría mezclando operaciones relacionadas al negocio de desplegar información médica, con la exposición de operaciones para el manejo de usuarios. Dicho enfoque tiene como desventaja la desorganización en el manejo de las operaciones, teniendo a un cliente web que llamará a un servicio web que expone operaciones de todo tipo. Tiene más sentido tener dos servicios, y agrupar operaciones con base en el negocio, es decir, en el servicio web `PatientService` se expondrán operaciones destinadas a devolver información médica para desplegarla en las interfaces de usuario, mientras que en el servicio web `PhysicianService` se expondrán las operaciones necesarias para el manejo de médicos.

- Cuando existe un cambio en el versionamiento de un servicio web, este no afectará a los otros. Es más probable que cambien las operaciones de despliegue de información médica a que cambie las operaciones de manejo de médicos, esto conlleva a tener que actualizar al servicio web `PatientService` más seguido que al servicio web `PhysicianService`. El proceso de actualizar la versión del servicio web `PatientService` implica costos de publicación y riesgos de fallos que solo afectarán a este servicio.

Tabla 2.19 Servicios web WCF a implementarse en la capa de servicio

#	Servicio Web	Descripción
1	`PatientService`	Expondrá operaciones relacionadas a la información médica asociada al paciente. Por ejemplo, si se requiere devolver los registros de medicinas o las alergias de un paciente.
2	`PhysicianService`	Expondrá operaciones relacionadas al manejo de médicos. Por ejemplo, guardar información de un médico, o devolver una lista de médicos.
3	`ReportService`	Expondrá operaciones relacionadas a la generación de reportes.

2.3.1.7 Capa de presentación

La capa de presentación será implementada usando el *framework* MVC.NET. Se creará una aplicación web de MVC[32] la cual consumirá los servicios web `PatientService`, `PhysicianService` y `ReportService`.

Esta capa usará el patrón de diseño MVC, el cual separa los datos y la lógica de presentación, de la interfaz de usuario[33]. El *framework* MVC.NET permite la implementación de una aplicación web siguiendo el patrón de diseño MVC. La aplicación web por implementarse en esta capa estará compuesta de vistas de MVC, controladores de MVC, modelos de MVC, archivos JavaScript y CSS, y varias clases de C# utilitarias que permitirán implementar varias de las funcionalidades que MVC.NET ofrece.

Una vista de MVC contiene una mezcla de código de cliente[34] con código de servidor. El código de cliente es un contenido estático que podrá formar parte de una interfaz de usuario. El código de servidor embebido en una vista de MVC permite generar más contenido en tiempo de ejecución mientras presenta una interfaz de usuario en el explorador web. Los controladores de MVC trabajan de la mano con las vistas de MVC, ya que dichos controladores ejecutan el contenido de las vistas MVC y retorna el contenido resultante para ser dibujado en la interfaz de usuario.

Una interfaz de usuario podrá estar formada por una o más vistas de MVC. En la siguiente sección se hará un diseño de las interfaces de usuario que se requieren para cumplir con los requerimientos del sistema, luego se determinará que vistas de MVC se requieren implementar por cada interfaz de usuario.

2.3.1.7.1 Diseño de interfaces de usuario y lógica de presentación

En esta sección se presentarán las interfaces de usuario que se requieren en el sistema con base en los requerimientos, se analizará la interacción que tendrán los

---

[32] MVC (*Model View Controller*) es un patrón de diseño aplicable a la capa de presentación.
[33] Una interfaz de usuario es aquella interfaz que permite pasar información entre un usuario humano y los componentes de *hardware* o *software* de un sistema computacional [50].
[34] En la capa de presentación, código de cliente se refiere al código que forma parte de una interfaz de usuario el cual es dibujado en un explorador web. El código de cliente puede ser de tipo HTML, JavaScript o CSS.

usuarios con el sistema a través de las interfaces y además se establecerá los roles de usuario requeridos para acceder a cada una de ellas. Finalmente se construirán *mockups* como parte del diseño de las interfaces, y se determinará que campos de las tablas serán presentados en cada una de ellas. Un *mockup* es una representación visual que permite a los diseñadores web mostrar al cliente como lucirán las interfaces de usuario.

2.3.1.7.2 Interfaces de usuario

Una interfaz de usuario es aquella que permite a un usuario interactuar con el sistema. En esta sección se realizará un análisis de cuantas interfaces se requieren dentro del sistema.

En la sección **2.3.1.1** se presentan las tablas de la base de datos médica y a excepción de la tabla `TestPhysician`, todas ellas contienen información médica relacionada a un paciente, por lo que tendrá más sentido construir interfaces de usuario que desplieguen información por cada una de las tablas, pero que solo corresponda a un paciente en específico, paciente que el usuario seleccionará.

Por tanto, se construirá una interfaz por cada una de las tablas de la base de datos con información médica, que permita desplegar la información contenida en ella de un paciente, exceptuando la tabla `TestPhysician` ya que un registro de esa tabla representa un médico, que en el negocio también representará a un usuario.

La **Tabla 2.20** presenta las interfaces que desplegarán información médica asociada a pacientes junto con la tabla de la cual se obtendrá la información a desplegar.

Es necesario crear varias interfaces adicionales que permitan realizar la administración de usuarios y de sus roles. En la **Tabla 2.21** se resume la información de las interfaces de usuario que se implementará para el manejo de usuarios y roles.

Cabe mencionar que los médicos son un tipo de usuario del sistema, y como los registros de médicos se encuentran en la base de datos médica, y los registros de usuario en la base identidad, la interfaz para el manejo de usuarios médicos deberá tener interacción con ambas bases de datos.

Además, se diseñarán otras interfaces con los siguientes propósitos: una interfaz de usuario para seleccionar un reporte y desplegarlo, otra interfaz para seleccionar un paciente, y otra interfaz de usuario de inicio, que se presentará una vez que un usuario inicia sesión. La **Tabla 2.22** muestra la información de otras interfaces de usuario.

2.3.1.7.3 Interacción usuario – interfaz de usuario

En esta sección se presentará la interacción de los usuarios con las interfaces mediante diagramas de flujo. Los diagramas de flujo se construyen en base a los requerimientos del sistema, y algunos de ellos son presentados en esta sección.

Tabla 2.20 Interfaces de usuario que desplegarán información médica

Interfaz de usuario	Tabla de la base de datos médica de la cual se obtiene la información	Descripción
Patient General Info	dbo.TestPatient	Permitirá desplegar información general del paciente seleccionado.
Allergy	dbo.Allergy	Permitirá desplegar información de alergias del paciente seleccionado.
Vascular Access	dbo.VascularAccessDevice	Permitirá desplegar información de dispositivos de acceso vascular del paciente seleccionado.
Infection	dbo.Infection	Permitirá desplegar información de infecciones del paciente seleccionado.
Immunization	dbo.Immunization	Permitirá desplegar información de inmunizaciones del paciente seleccionado.
Treatments	dbo.Treatments	Permitirá desplegar información de tratamientos del paciente seleccionado.
Medications	dbo.Medications	Permitirá desplegar información de medicamentos del paciente seleccionado.

Uno de los requerimientos del sistema es permitir el registro de usuarios, para ello se diseñará la interfaz `Register`, la cual puede ser accedida por cualquier usuario sin necesidad de haber iniciado sesión.

El usuario que sigue el proceso de registro deberá ingresar cierta información personal, así como un correo electrónico y la clave (el correo electrónico y la clave serán requeridos para el proceso de autenticación).

Si el proceso de registro es exitoso, se insertará un nuevo registro en la tabla `dbo.User` de la base `IdentityDatabase`, caso contrario se desplegarán un mensaje de error. El proceso completo de registrar un usuario se muestra en la **Figura 2.7**.

Tabla 2.21 Interfaces de usuario requeridas para manejo de usuarios y roles

Interfaz de usuario	Tablas de las cuales se obtiene la información	Descripción
`Register`	`dbo.User`	Permitirá registrar nuevos usuarios.
`Manage Users`	`dbo.User`	Permitirá administrar usuarios. En el caso de los usuarios médicos se requiere tener acceso tanto a la base `MedicalDatabase` como la base `IdentityDatabase`.
`Manage Physicians`	`dbo.User` `dbo.TestPhysician`	Permitirá administrar usuarios médicos. Se requiere tener acceso tanto a la base `MedicalDatabase` como la base `IdentityDatabase`.

`Manage Roles`	`dbo.Role`
	`dbo.UserRole`
	`dbo.User`
		Permitirá administrar los roles de un usuario.

Tabla 2.22 Otras interfaces de usuario a implementarse

Interfaz de usuario	Descripción
`Login`	Permitirá autenticar usuarios
`Home`	Interfaz de usuario de inicio del sistema, y es accedida una vez que el usuario inicia sesión. Esta desplegará alertas o mensajes dirigidos al usuario que inició sesión.
`Select patient`	A través de esta interfaz el usuario podrá seleccionar un paciente cuando lo requiera
`Reports`	Permitirá seleccionar y desplegar reportes

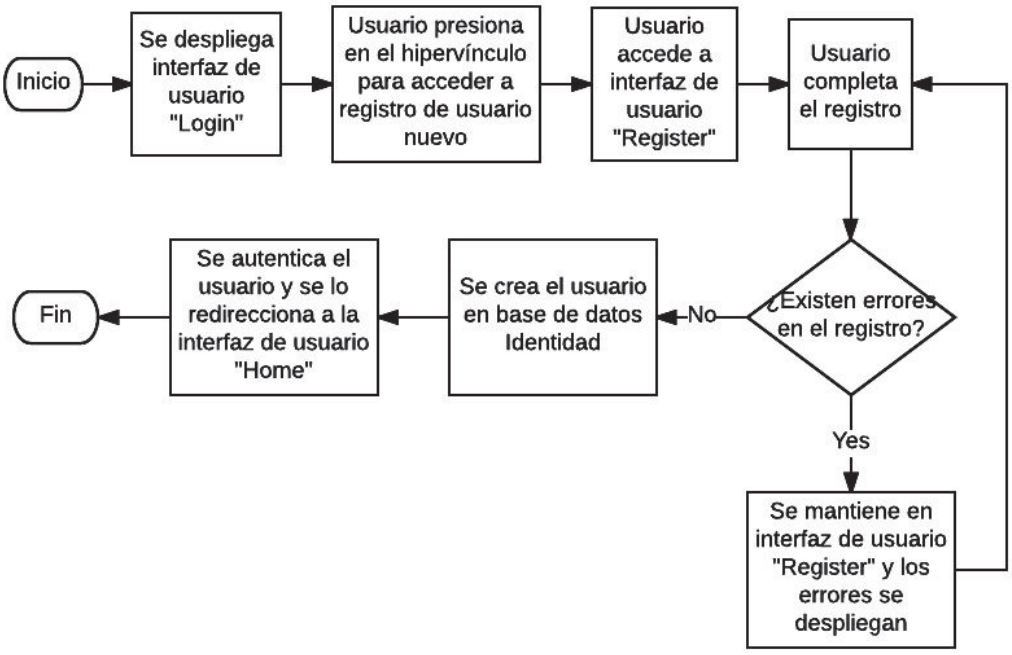


Figura 2.7 Diagrama de flujo del registro de un usuario

Una vez que un usuario se registra, tiene la posibilidad de iniciar sesión en el sistema. El proceso completo de iniciar sesión se muestra en la **Figura 2.8**, cabe añadir que la interfaz de usuario `Home` solo está disponible si el usuario inicia sesión.

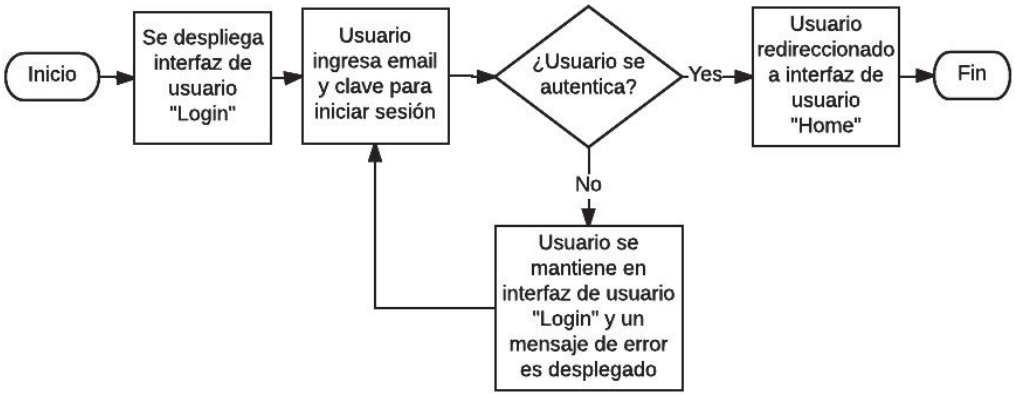


Figura 2.8 Diagrama de flujo del inicio de sesión de un usuario existente

Finalmente se presentará el diagrama de flujo del proceso de selección de un paciente a través de la interfaz `Select patient`. En el diseño de interfaces se determinó que varias de ellas desplegarán información médica de un paciente en específico, es por eso que se requerirá seleccionar un paciente antes de acceder a cada una de ellas. Además, para facilitar la interacción del usuario con las interfaces, se requiere mantener almacenado el ID del paciente seleccionado para evitar que el usuario tenga que seleccionar el mismo cada vez que acceda a otra de las interfaces asociadas a un paciente.

La **Figura 2.9** muestra el proceso completo de seleccionar el paciente la primera vez que se requiera acceder a una interfaz que despliegue información asociada a un paciente en específico durante la sesión. El ID del paciente será almacenado en una variable de sesión para evitar que el usuario lo ingrese constantemente.

2.3.1.7.4 Autorización

El sistema requiere la implementación de autorización mediante roles. Cada usuario que requiera acceder a una interfaz de usuario deberá tener asignado el rol apropiado para poder tener acceso. La **Tabla 2.23** muestra las interfaces de usuario que pueden ser accedidas considerando el rol de usuario.

En primera instancia, el usuario no podrá ver en el menú las opciones que permiten ir a las interfaces de usuario a las cuales no tiene acceso, pero si este intenta acceder a una de ellas especificando la URL en el explorador web, internamente la aplicación web denegará el acceso y re direccionará hacia la interfaz de usuario `Login` tal como lo muestra el diagrama de flujo de la **Figura 2.10**.

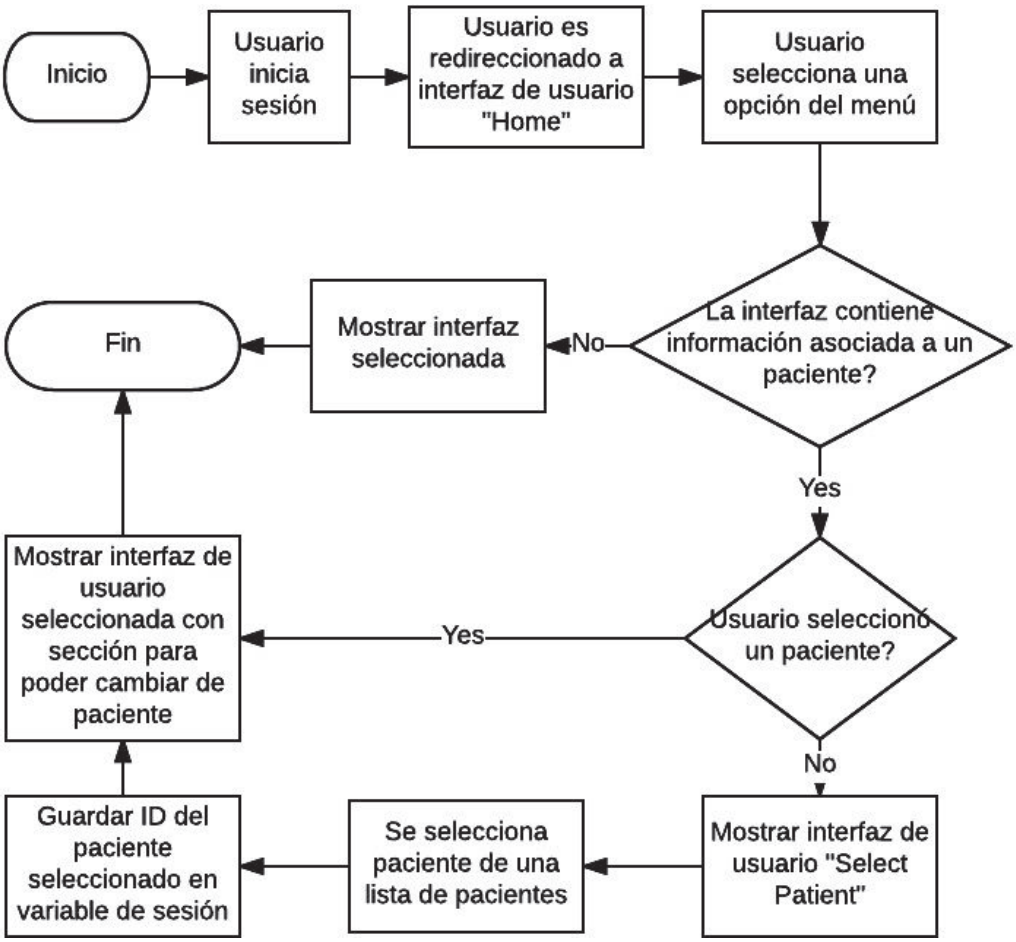


Figura 2.9 Proceso de selección de un paciente

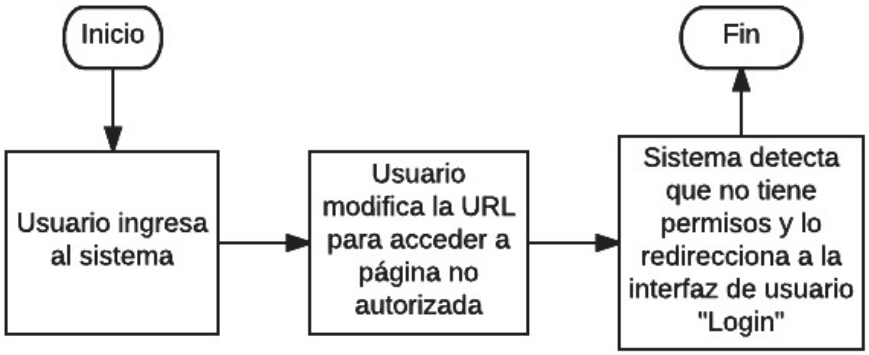


Figura 2.10 Usuario accediendo a una interfaz de usuario no autorizada

La **Figura 2.11** muestra un diagrama de interacción del usuario con las interfaces de usuario del sistema en base a su rol. Independientemente del rol del usuario, una vez que inicia sesión un usuario, este será re direccionado a la interfaz de usuario `Home`.

Tabla 2.23 Interfaces de usuario que pueden ser accedidas considerando el rol de usuario

#	Interfaz de usuario	Rol			
		Default	Physician Unauthorized	Physician	Admin
1	Log in	SI	SI	SI	SI
2	Register	SI	SI	SI	SI
3	Home	SI	SI	SI	SI
4	Select patient	NO	NO	SI	SI
5	Patient General Info	NO	NO	SI	SI
6	Allergy	NO	NO	SI	SI
7	Vascular Access	NO	NO	SI	SI
8	Infection	NO	NO	SI	SI
9	Immunization	NO	NO	SI	SI
10	Treatments	NO	NO	SI	SI
11	Medications	NO	NO	SI	SI
12	Manage Users	NO	NO	NO	SI
13	Manage Physicians	NO	NO	NO	SI
14	Manage Roles	NO	NO	NO	SI
15	Reports	NO	NO	SI	SI

2.3.1.7.5 Mockups

Es importante el desarrollo de *mockups* antes de iniciar con la construcción de las interfaces web, porque estas sirven de base para implementar la interfaz de usuario, sin embargo, es necesario aclarar que las interfaces finales podrían variar.

La **Figura 2.12** muestra el *mockup* de la interfaz de usuario `Login`.

Usualmente en otros sistemas se usa el nombre de usuario como el identificador único del usuario, sin embargo, en este sistema se determinó que para facilitar el registro de usuarios, se haga uso del correo electrónico como identificador único.

La interfaz de usuario `Login` permitirá tener acceso a la interfaz de usuario `Register` a través de un hipervínculo.

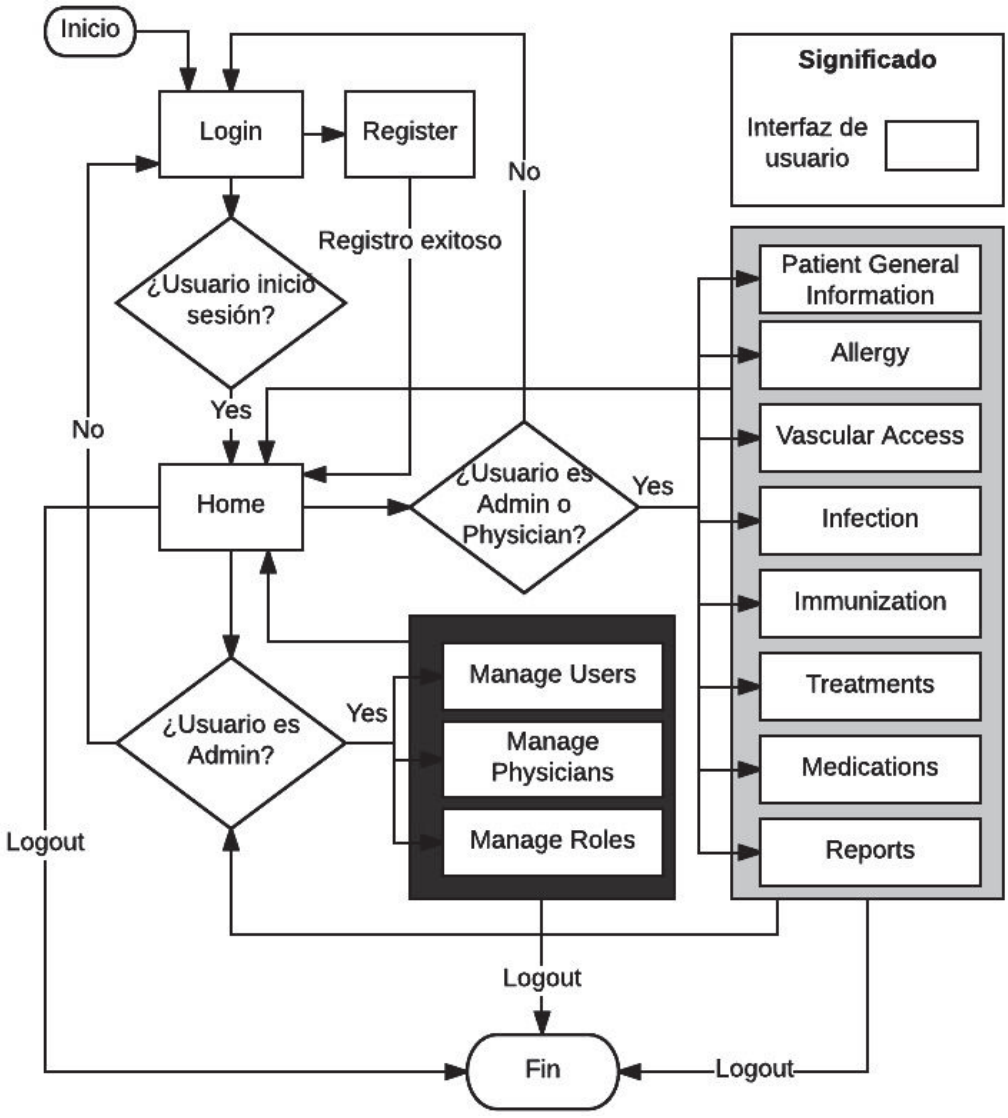


Figura 2.11 Interacción de usuario con interfaces de usuario

La **Figura 2.13** muestra el *mockup* de la interfaz de usuario `Register`. Esta interfaz permitirá el registro de nuevos usuarios. Para el registro, es necesario que el nuevo usuario ingrese información personal como nombres, apellido, correo electrónico, número de teléfono y nombre de usuario. No se podrá ingresar un correo electrónico que esté siendo usado por otro usuario.

El *framework* de estilos Twitter Bootstrap clasifica a los dispositivos en base a su tamaño de pantalla tal como se muestra en la **Tabla 2.24**. A diferencia de las interfaces `Login` y `Register`, es necesario definir *mockups* para las interfaces según el tamaño del dispositivo desde donde serán accedidas.

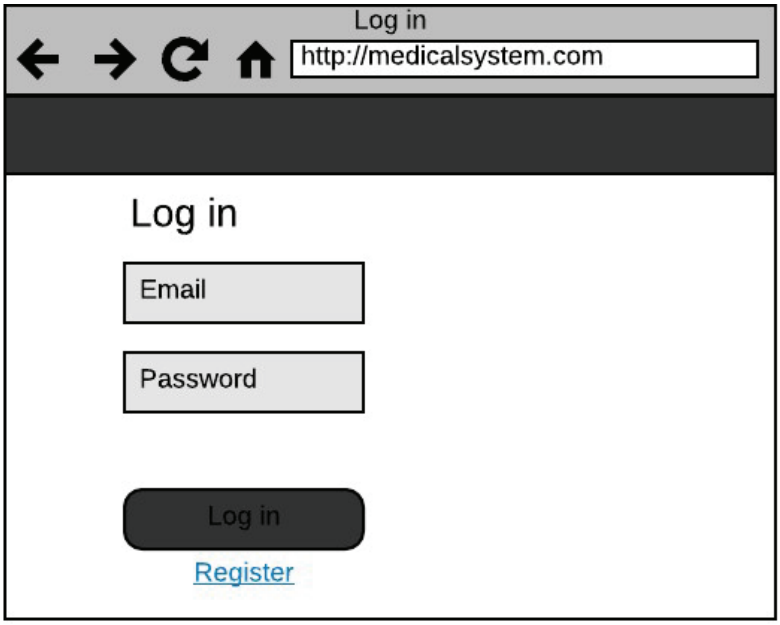


Figura 2.12 *Mockup* de la interfaz de usuario `Login`

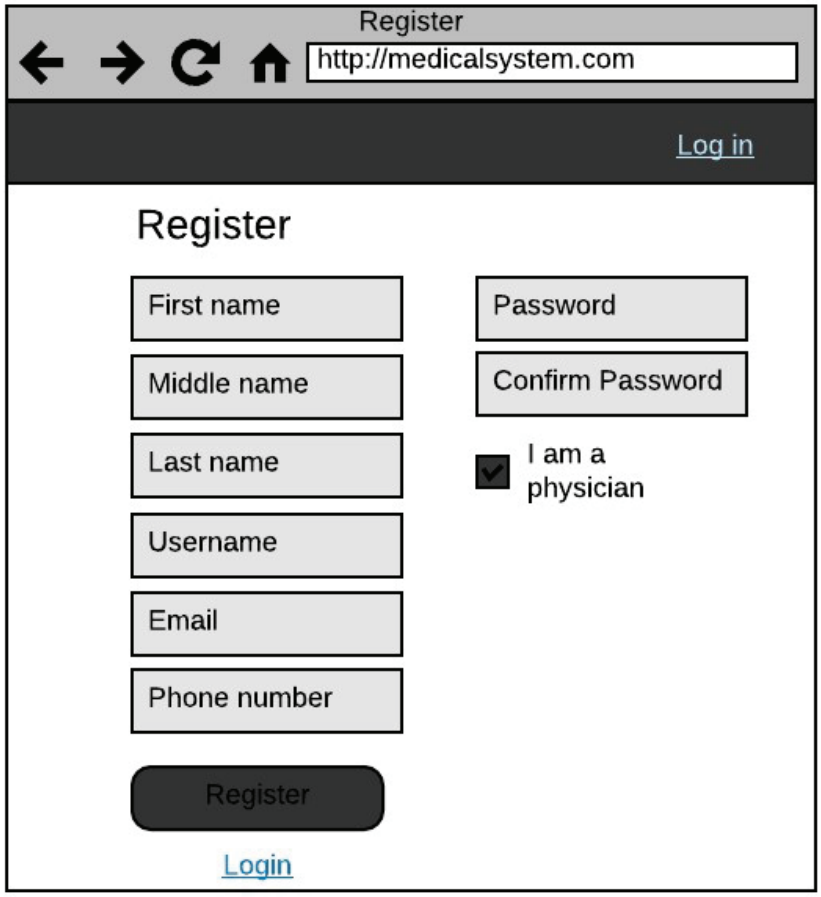


Figura 2.13 *Mockup* de la interfaz de usuario `Register`

La **Figura 2.14** muestra el *mockup* de interfaz de usuario `Home` para dispositivos cuya pantalla sea menor a 768 pixeles.

Mientras que la **Figura 2.15** muestra el *mockup* de la misma interfaz para dispositivos cuya pantalla sea mayor o igual a 768 pixeles. Se puede ver que en el primer caso el menú no aparece en el lado izquierdo de la pantalla, sino que se colapsa hacia la barra superior, dejando todo el espacio para desplegar la información de usuario, que tanto se requiere para un dispositivo con una pantalla reducida.

Figura 2.14 *Mockup* de la interfaz de usuario `Home` para dispositivos extra pequeños

Tabla 2.24 Clasificación de tipos de dispositivos según Twitter Bootstrap

Dispositivo	Tamaño de pantalla	Tipo de dispositivo
Extra pequeño	<768 px	Teléfono móvil
Pequeño	≥768 px	*Tablet*
Mediano	≥992 px	PC
Grande	≥1200 px	PC

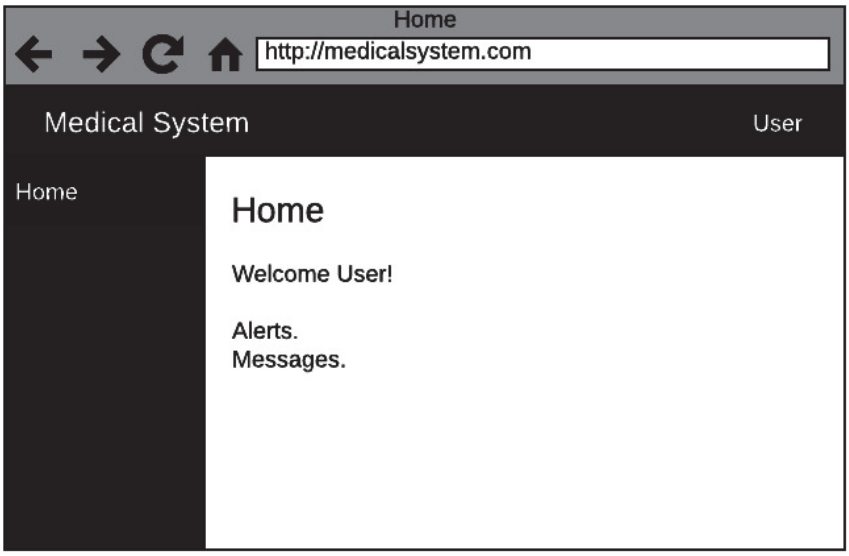


Figura 2.15 *Mockup* de la interfaz `Home` para dispositivos pequeños, medianos y grandes

La **Figura 2.16** muestra el *mockup* de la interfaz de usuario `Select patient` para dispositivos cuya pantalla sea menor a 768 pixeles, mientras que la **Figura 2.17** muestra el *mockup* de la misma interfaz para dispositivos cuya pantalla sea mayor o igual a 768 pixeles.

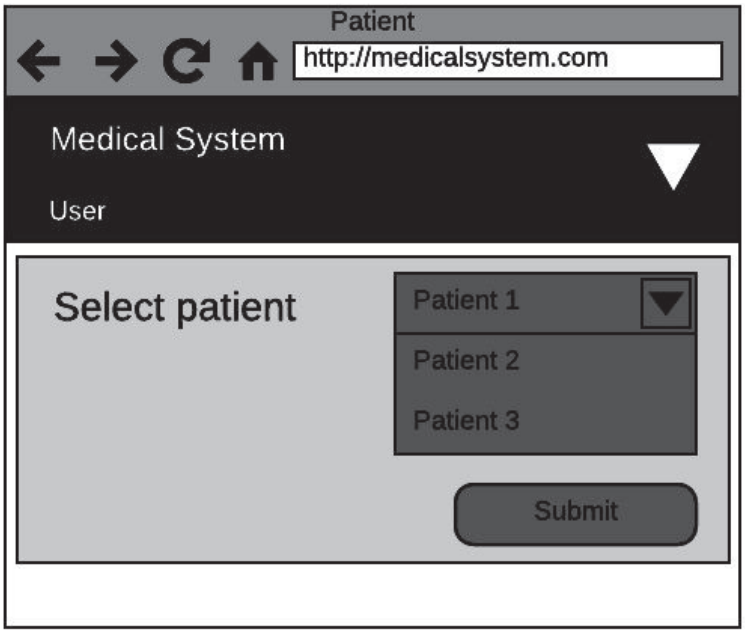


Figura 2.16 *Mockup* de la interfaz `Select patient` para dispositivos extra pequeños

La **Figura 2.18** muestra el *mockup* de la interfaz de usuario `Patient General Information` para dispositivos cuya pantalla sea menor a 768 pixeles.

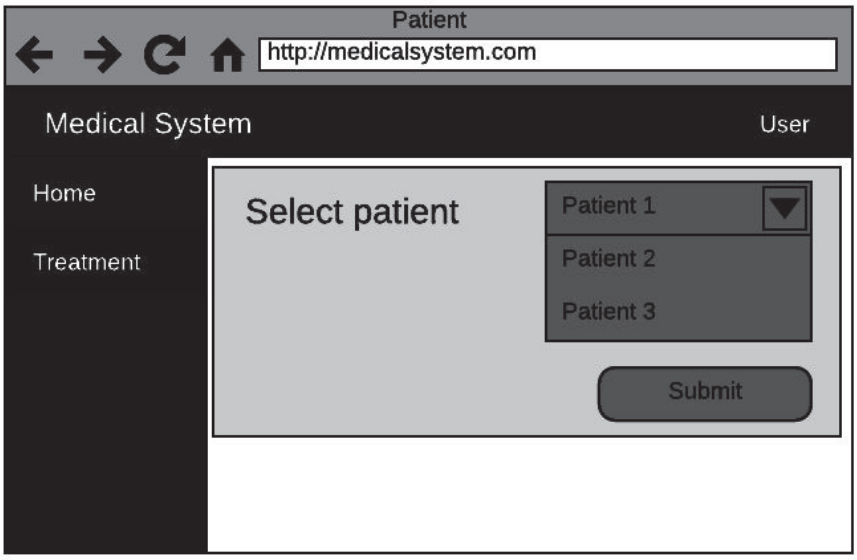


Figura 2.17 *Mockup* de la interfaz de usuario `Select patient` para dispositivos pequeños, medianos y grandes

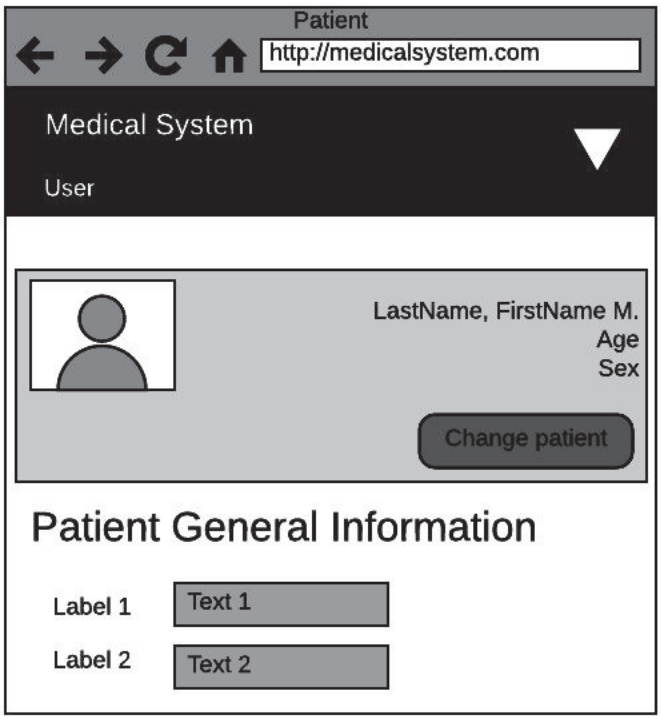


Figura 2.18 *Mockup* de la interfaz de usuario `Patient general information` para dispositivos extra pequeños

La **Figura 2.19** muestra el *mockup* de la interfaz de usuario `Patient General Information` para dispositivos cuya pantalla sea mayor o igual a 768 pixeles

En ambas interfaces `Select patient` y `Patient General Information` se aplica la estrategia de colapsar el menú para liberar el espacio cuando la interfaz sea accedida desde un dispositivo extra pequeño.

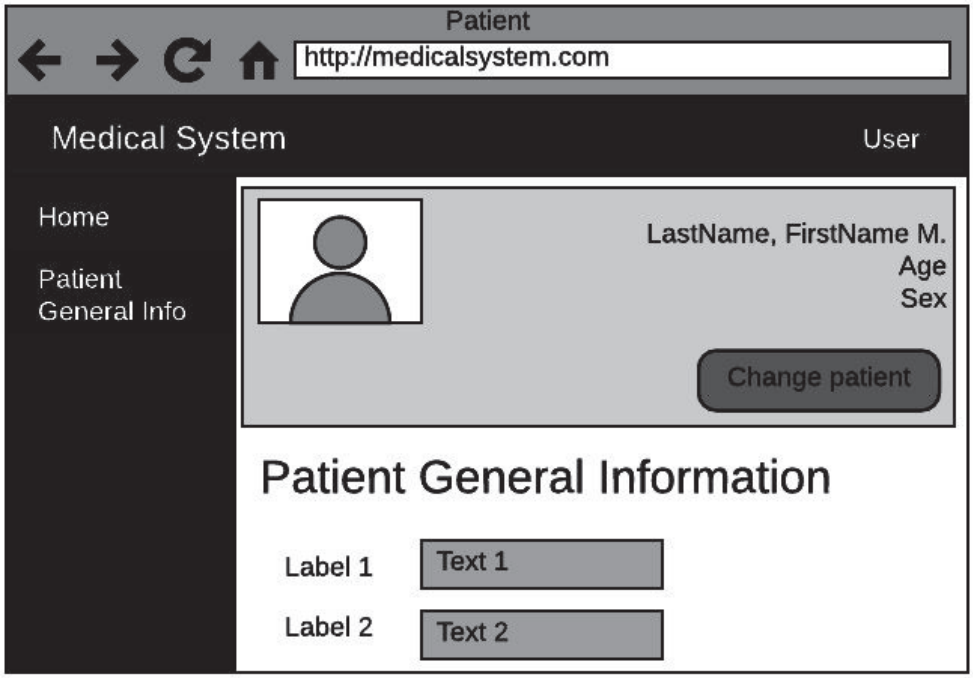


Figura 2.19 *Mockup* de la interfaz de usuario `Patient general information` para dispositivos pequeños, medianos y grandes

Las interfaces como `Allergy` manejarán un esquema diferente de despliegue. La relación de cero a muchos entre la entidad `TestPatient` y `Allergy` implica que en la interfaz de usuario `Allergy` se mostrarán varias alergias, es por ello que se implementará un *grid* que despliegue un resumen de las alergias del paciente.

Para poder visualizar el detalle de la alergia, el usuario seleccionará uno de los registros de alergias del *grid* y se desplegará una sección de detalle bajo este con todos los campos asociadas a este registro. Este esquema de interfaz se denomina *grid-detail*. Las interfaces que siguen este esquema se muestran en la **Tabla 2.25**. Se diseñó un *mockup* genérico para este tipo de interfaces de usuario. En la **Figura 2.20** se muestra el *mockup* de interfaces de usuario que siguen el esquema *grid-*

detail para dispositivos con pantalla menor a 768 pixeles, mientras que el *mockup* para los dispositivos cuya pantalla sea mayor o igual a 768 pixeles se muestra en la **Figura 2.22**.

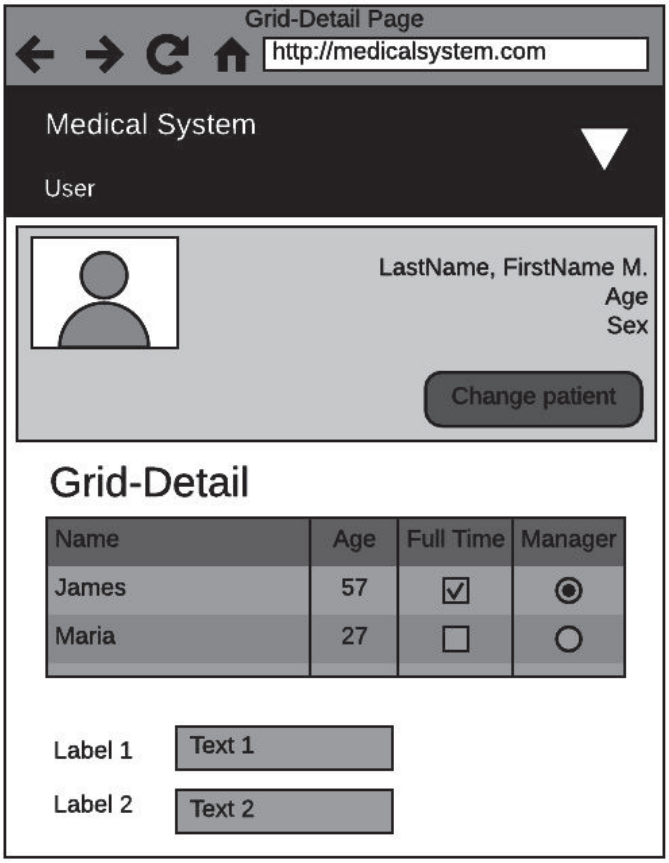


Figura 2.20 *Mockup* de las interfaces de usuario que siguen el esquema *grid-detail* para dispositivos extra pequeños

Los campos que serán desplegados en las interfaces establecidas se definirán en la siguiente sección.

La interfaz de usuario `Reports` permitirá seleccionar un reporte de una lista de reportes, seleccionar filtros[35], generar el reporte y desplegarlo al usuario. Tanto los reportes que estarán disponibles en el sistema, como los filtros requeridos, serán definidos en la sección 2.3.1.7.7. La **Figura 2.21** muestra el *mockup* de la interfaz `Reports` para dispositivos cuya pantalla sea mayor o igual a 768 pixeles.

[35] Los filtros permitirán filtrar la información que se desplegará en cada uno de los reportes

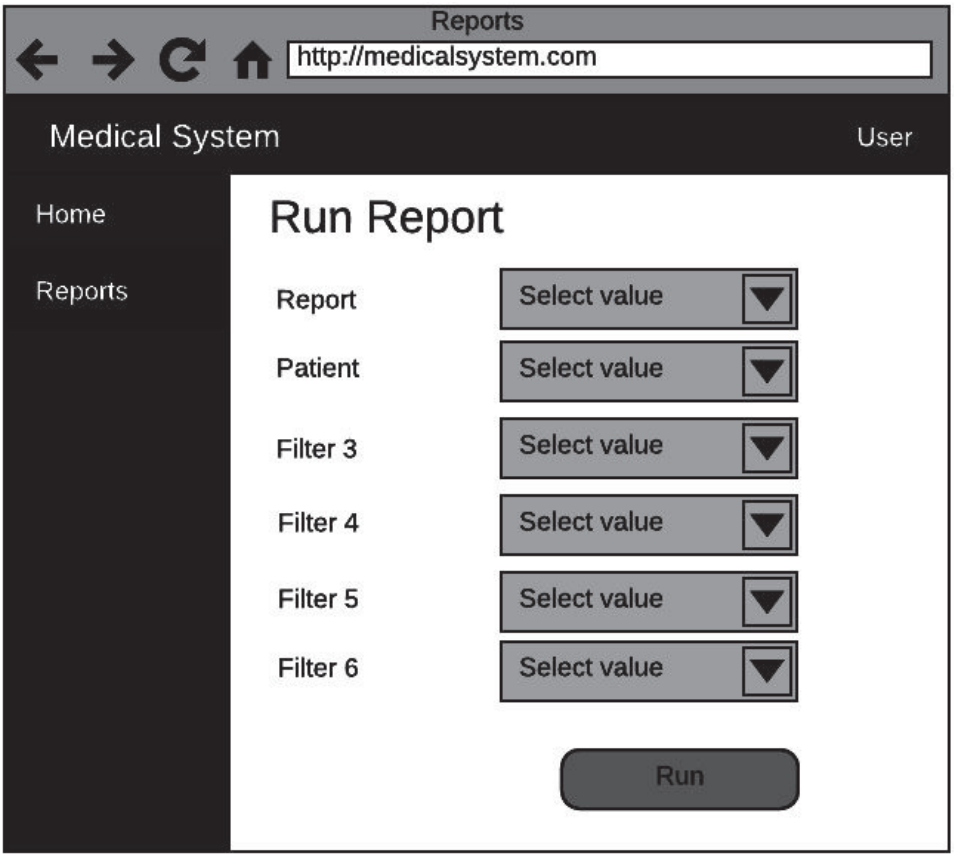


Figura 2.21 *Mockup* de la interfaz de usuario `Reports` para dispositivos pequeños, medianos y grandes

Tabla 2.25 Interfaces de usuario a implementarse

Interfaz de usuario	Orden menú	¿Específica para un paciente?	¿Sigue esquema *grid-detail*?
`Home`	1	NO	NO
`Patient General Information`	2	SI	NO
`Allergy`	3	SI	SI
`Vascular Access Device`	4	SI	SI
`Infection`	5	SI	SI
`Immunization`	6	SI	SI
`Treatment`	7	SI	SI
`Medication`	8	SI	SI
`Manage Users`	9	NO	SI
`Manage Physicians`	10	NO	SI
`Manage Roles`	11	NO	SI
`Reports`	12	NO	NO

Finalmente se mostrarán los *mockup* diseñados para seleccionar un paciente. Dicha funcionalidad será detallada usando el ejemplo de la interfaz de usuario `Treatment`. Cuando un usuario inicia sesión, sea este `Admin` o `Physician`, y accede a la interfaz de usuario `Treatment`, notará que la URL corresponde a la interfaz `Treatment`, sin embargo, visualiza la interfaz `Select Patient`, tal como se muestra en la **Figura 2.23**. Esto sucede porque el usuario no ha seleccionado un paciente aún, por lo que la aplicación web presenta la interfaz `Treatment` de tal manera que luzca como la interfaz `Select Patient`.

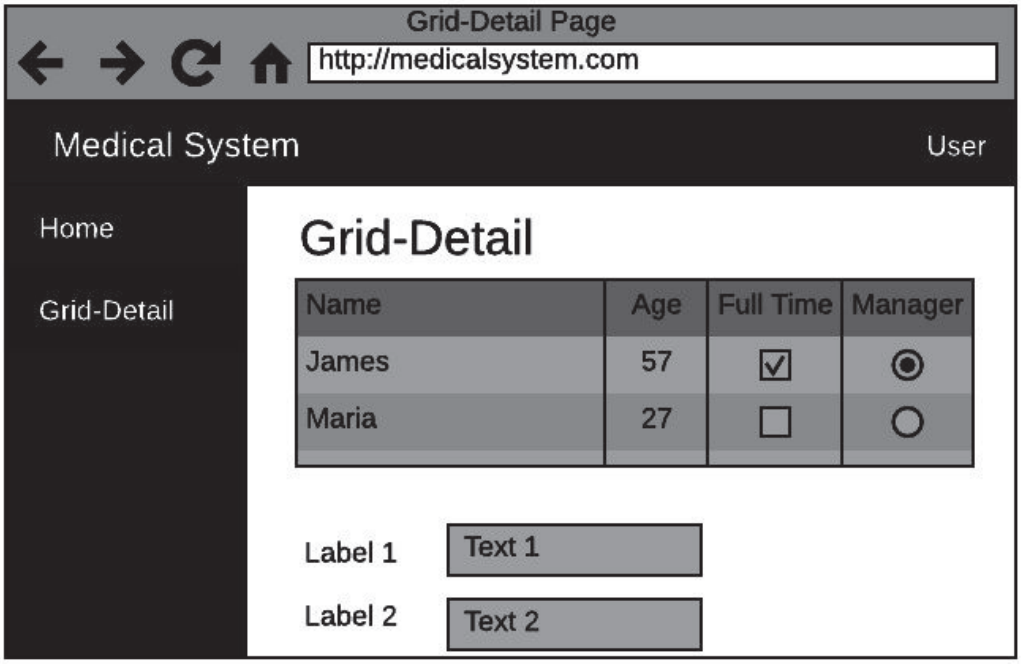


Figura 2.22 *Mockup* de las interfaces de usuario que siguen el esquema *grid-detail* para dispositivos pequeños, medianos y grandes

Luego, a lo que el usuario selecciona un paciente, la interfaz de usuario se volverá a cargar en el explorador web, con la diferencia de que esta vez sí se despliega la información médica esperada, así como también la sección para que el usuario pueda cambiar de paciente, tal como se muestra en la **Figura 2.24**.

2.3.1.7.6 Campos a ser desplegados en interfaces de usuario con información médica

En esta sección se definen que campos no serán presentados en las interfaces que desplegarán información médica, es decir, información que proviene de la base de datos médica.

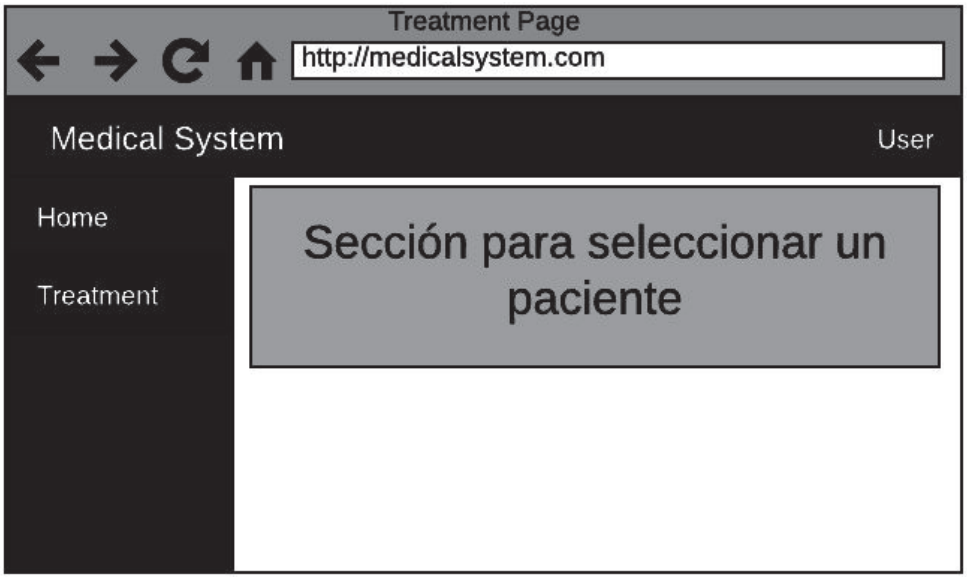


Figura 2.23 *Mockup* de la interfaz `Treatment` cuando usuario no ha seleccionado un paciente

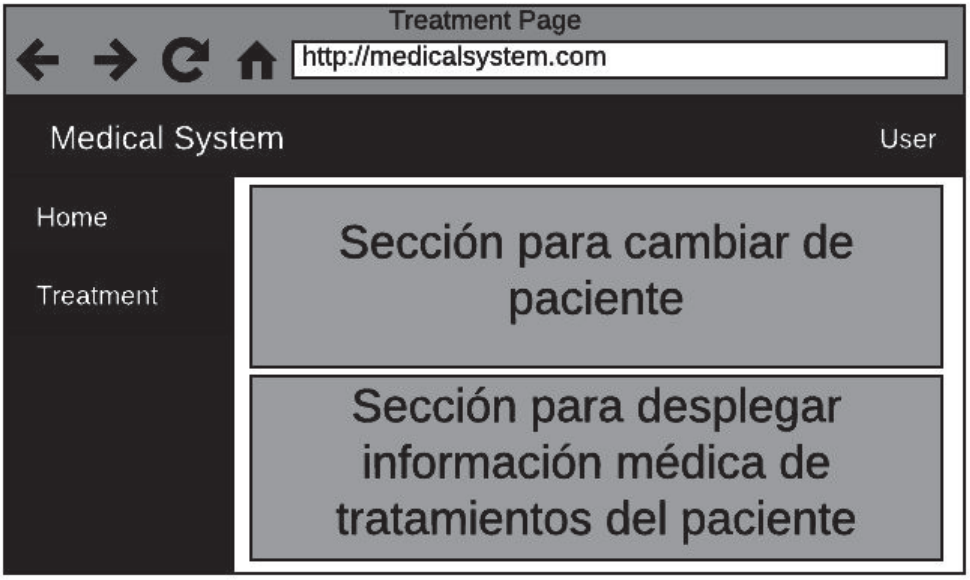


Figura 2.24 *Mockup* de la interfaz `Treatment` cuando ya se seleccionó un paciente

Para decidir cuales campos se muestran y cuales no dentro de cada interfaz, se toma en consideración el tipo de información que cada campo posee.

Aquellos campos que cumplan con al menos uno de los siguientes requisitos no serán desplegados dentro de las interfaces de usuario:

- Campos que tengan claves primarias
- Campos que tengan claves foráneas
- Campos que sean un código o un identificador
- Campos que hacen referencia a tablas que no fueron incluidas en la base de datos médica.
- Campos de auditoría

Se encontró que todas las tablas provenientes de la base de datos médica tienen 4 campos de auditoría, estos son:

- `AddedDate`: Fecha de creación del registro
- `EditDate`: Fecha de la última modificación del registro
- `AddedUser`: ID del usuario quien añadió el registro
- `EditUser`: ID del usuario que realizó la última modificación del registro

Ninguno de los campos de auditoría será desplegado en las interfaces de usuario del sistema, ya que la intención de la información de auditoría es mantener un registro para determinar quiénes y cuando hicieron alteraciones en los datos, más no para que sea desplegado a los usuarios del sistema.

La **Tabla 2.26** muestra los campos de la tabla `TestPatient` que no serán desplegados en la interfaz de usuario `Patient General Information`.

La **Tabla 2.27** muestra los campos de la tabla `TestPhysician` que no serán desplegados en las interfaces de usuario `Manage Users` y `Manage Physicians`.

Tabla 2.26 Campos de la tabla `TestPatient` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`TestPatientID`	`int`	Clave primaria
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría
`labidnumber`	`varchar`	Clave foránea

Tabla 2.27 Campos de la tabla `TestPhysician` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

La **Tabla 2.28** muestra los campos de la tabla `Allergy` que no serán desplegados en la interfaz de usuario `Allergy`.

La **Tabla 2.29** muestra los campos de la tabla `VascularAccessDevice` que no serán desplegados en la interfaz de usuario `Vascular Access Device`.

La **Tabla 2.30** muestra los campos de la tabla `Infection` que no serán desplegados en la interfaz de usuario `Infection`.

La **Tabla 2.31** muestra los campos de la tabla `Immunization` que no serán desplegados en la interfaz de usuario `Immunization`.

La **Tabla 2.32** muestra los campos de la tabla `Treatment` que no serán desplegados en la interfaz de usuario `Treatment`.

Tabla 2.28 Campos de la tabla `Allergy` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`MedID`	`int`	Clave foránea
`DRUG_ID`	`varchar`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

Tabla 2.29 Campos de la tabla `VascularAccessDevice` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

Tabla 2.30 Campos de la tabla `Infection` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`VascularAccessDeviceIDNumber`	`int`	Clave foránea
`ObservedByUserIDNumber`	`int`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

Tabla 2.31 Campos de la tabla `Immunization` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`RunIDNumber`	`int`	Clave foránea
`MedicationCode`	`varchar`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

Tabla 2.32 Campos de la tabla `Treatment` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`OutpatientTreatment`	`tinyint`	Clave foránea
`UnitID`	`smallint`	Identificador
`InputFile`	`varchar`	Información no tiene valor para el usuario del sistema
`txSubtype`	`int`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

La **Tabla 2.33** muestra los campos de la tabla `Medication` que no serán desplegados en la interfaz de usuario `Medication`.

Todos los campos restantes de las tablas de la base de datos médica serán desplegados en sus respectivas interfaces de usuario.

2.3.1.7.7 Reportes

En esta sección se definirá que reportes serán implementados en el sistema, y además que filtros podrá ser aplicado a cada uno de los reportes.

Tomando en cuenta la información que contiene las tablas de la base de datos médica, se definió una lista de reportes que podrían ser útiles en el trabajo de un médico y que se implementaron en el sistema.

La **Tabla 2.34** muestra la lista de reportes a implementarse acompañados de una breve descripción de cada uno de ellos.

La interfaz de usuario de reportes permite establecer filtros antes de ejecutar un reporte. Los filtros permiten limitar la cantidad de datos que se despliegan en cada uno de los reportes. La **Tabla 2.35** muestra los filtros a implementarse en la ejecución de reportes desde la interfaz de usuario de reportes.

Tabla 2.33 Campos de la tabla `Medication` que no serán visibles en la interfaz de usuario.

Columna	Tipo de dato	Descripción
`ID`	`int`	Clave primaria
`PatientID`	`int`	Clave foránea
`MedID`	`int`	Clave foránea
`GENPRODUCT_ID`	`int`	Clave foránea
`DRUG_ID`	`varchar`	Clave foránea
`AddedDate`	`datetime`	Campo de auditoría
`EditDate`	`datetime`	Campo de auditoría
`AddedUser`	`int`	Campo de auditoría
`EditUser`	`int`	Campo de auditoría

Tabla 2.34 Reportes

Reporte	Descripción
`Allergies`	Despliega las alergias del paciente seleccionado.
`Immunizations`	Despliega las inmunizaciones del paciente seleccionado en el rango de fechas seleccionado.
`Current Infections`	Despliega las infecciones que posee el paciente seleccionado.
`Current Medications`	Despliega los medicamentos que el paciente seleccionado tiene al momento de ejecutar el reporte.
`On Hold Medications`	Despliega los medicamentos en espera del paciente seleccionado.
`Treatments`	Despliega los tratamientos dados al paciente seleccionado en el rango de fechas seleccionado.
`Active Vascular Accesses`	Despliega los accesos vasculares activos del paciente seleccionado.

El filtro `Range Date` permite filtrar en base a fechas de inicio y de fin. Este filtro solo es aplicable a los reportes `Allergies`, `Treatments` e `Immunizations`, ya que solo en estos reportes se da la posibilidad de que existan muchos registros, al punto de que necesiten ser filtrados de esa manera. Los reportes restantes devuelven información médica más específica, por ejemplo, infecciones actuales, o accesos vasculares activos.

Tabla 2.35 Filtros de reportes

Filtro	Descripción
`Patient`	Permite seleccionar un paciente. El reporte desplegará solo la información relacionado al paciente seleccionado. Si no se selecciona un paciente, el reporte desplegará la información de todos los pacientes de la base de datos médica.
`Patient Status`	Permite seleccionar un estado de pacientes. El reporte solo desplegará datos de pacientes que cumplan con el estado de paciente seleccionado. Esta opción de filtro sólo es habilitada cuando se selecciona la opción de mostrar todos los pacientes en el filtro `Patient`.
`Patient Sex`	Permite seleccionar el sexo de pacientes. El reporte solo desplegará datos de pacientes que cumplan con el sexo del paciente seleccionado. Esta opción de filtro sólo es habilitada cuando se selecciona la opción de mostrar todos los pacientes en el filtro `Patient`.
`Range Date`	Permite seleccionar un rango de fechas sobre el cual se filtrará los datos del reporte. Este filtro no es aplicable a todos los reportes del sistema. El filtro `Range Date` se compone de dos campos: `Start Date` (fecha de inicio) y `End Date` (fecha de fin).

2.3.2 PRODUCT BACKLOG

Una vez definidos los requerimientos del sistema, y concluido el diseño de las capas, se procedió a listar cada uno de los ítems del *product backlog* que se requieren para la implementación, siguiendo los lineamientos que determina la metodología SCRUM. Los ítems del *product backlog* o PBI fueron creados a través de las herramientas de Visual Studio Team Services de Microsoft, la cual permite el manejo de código en la nube.

La **Tabla 2.36** muestra el *product backlog* definido después de concluir con el diseño del sistema. Cabe mencionar que el *product backlog* sufrió modificaciones durante todo el proyecto, en ocasiones se añadieron nuevos ítems, en otros se removieron, también se crearon ítems de tipo *bug* cuando se encontró funcionamiento erróneo del sistema, y se redefinió la prioridad de los ítems.

Tabla 2.36. *Product backlog* inicial

Prioridad	Título	Tipo de ítem de trabajo
1	*Basic web application with Home UI*	*Product Backlog Item*
2	*Implement Bootstrap Template with Menu*	*Product Backlog Item*
3	*Create Registration Page*	*Product Backlog Item*
4	*Create Login page*	*Product Backlog Item*
5	*Prototype DAL, BLL and WCF service*	*Product Backlog Item*
6	*Build PatientService WCF service*	*Product Backlog Item*
7	*Build Patient general information UI*	*Product Backlog Item*
8	*Build Treatment UI*	*Product Backlog Item*
9	*Build Problem List UI*	*Product Backlog Item*
10	*Build Vascular Access UI*	*Product Backlog Item*
11	*Build Medications UI*	*Product Backlog Item*
12	*Build Infection UI*	*Product Backlog Item*
13	*Build Manage Users UI*	*Product Backlog Item*
14	*Build Lab UI*	*Product Backlog Item*
15	*Move ASP.NET Identity EF out from Web Application*	*Product Backlog Item*
16	*Research about WCF security*	*Product Backlog Item*
17	*Implement security in PatientService WCF service*	*Product Backlog Item*
18	*Implement Treatments report*	*Product Backlog Item*
19	*Implement Summary report*	*Product Backlog Item*
20	*Implement Medications report*	*Product Backlog Item*
21	*Implement Vascular Access report*	*Product Backlog Item*
22	*Research Log4Net third party logging*	*Product Backlog Item*
23	*Implement logging in Web application*	*Product Backlog Item*
24	*Implement logging in Services layer*	*Product Backlog Item*
25	*Implement logging in BLL*	*Product Backlog Item*
26	*Implement Exception management for whole system*	*Product Backlog Item*
27	*Implement User roles in MVC Web Application*	*Product Backlog Item*

Por motivos de espacio en la redacción no se incluyó las distintas versiones del *product backlog* durante el proceso.

Tomando como referencia el *product backlog* inicial, se determinó que se requieren 4 *sprints* para desarrollar el sistema. En las siguientes secciones se detallará el trabajo realizado en cada *sprint*.

2.3.3 SPRINT 1

2.3.3.1 Sprint planning

En la **Tabla 2.37** se presentan los PBI escogidos para cumplir con el objetivo del *sprint* 1, los cuales son: construir una aplicación web básica con la interfaz de usuario "Home" y construir un prototipo de servicio WCF con las capas BLL y DAL con acceso a la base de datos médica.

Tabla 2.37. *Sprint backlog* 1

Prioridad	Título	Estado
1	*Basic web application with Home UI*	*Committed*
2	*Implement Bootstrap Template with Menu*	*Committed*
3	*Create Registration UI*	*Committed*
4	*Create Login UI*	*Committed*
5	*Prototype DAL, BLL and WCF service*	*Committed*

2.3.3.2 Desarrollo del sprint 1

2.3.3.2.1 Creación de aplicación web MVC

En esta sección se explica de manera resumida los pasos seguidos para crear una aplicación web MVC usando Visual Studio Community 2013.

Para crear un nuevo proyecto de tipo aplicación web MVC en Visual Studio se siguieron los siguientes pasos:

1. En la barra de herramientas seleccionar: FILE → New → Project
2. Seleccionar Templates → Web → ASP.NET Web Application. Seleccionar la opción *Add to source control* para habilitar el manejo de código en la nube. Finalmente darle nombre al proyecto y presionar OK.
3. Seleccionar la opción *template* = MVC. Añadir la opción *Individual User Accounts* para habilitar autenticación de usuarios. Seleccionar la opción *Host*

in the cloud para poder publicar la aplicación web en Azure. Por el momento no es necesario añadir el proyecto de *unit testing* ya que se lo podrá hacer posteriormente. Las opciones indicadas se muestran en la **Figura 2.25**. Finalmente, presionar OK.

4. La aplicación web MVC se genera en base a un *template*. La aplicación cuenta con varias interfaces de usuario y con autenticación de usuarios.

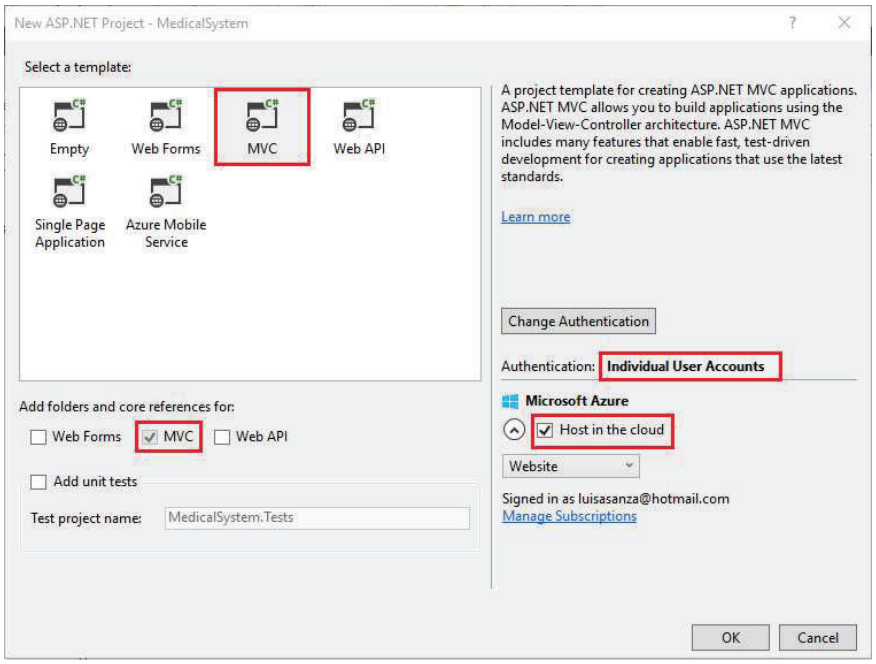


Figura 2.25 Creación de nuevo proyecto de aplicación web MVC en Visual Studio

El proyecto de aplicación web MVC generado por Visual Studio consta de los siguientes elementos:

- Carpeta `Content`: Contiene los archivos que implementan código CSS, los cuales permiten dar estilo a las interfaces de usuario.
- Carpeta `Controllers`: Contiene los archivos donde se implementan las clases controladores (`Controller`) de MVC.
- Carpeta `Models`: Contiene los archivos donde se implementan las clases modelos de MVC.
- Carpeta `Scripts`: Contiene los archivos que implementan código JavaScript, los cuales permiten añadir comportamiento a las interfaces de usuario.

- Carpeta `Views`: Contiene los archivos donde se implementan las vistas (`View`) de MVC. Típicamente, dentro de la carpeta *Views* cada controlador tiene su carpeta en la cual existen vistas para acciones específicas del controlador.
- Archivo `packages.config`: Contiene una lista de los paquetes *nuget*[36] instalados en el proyecto de aplicación web.
- Archivo `Web.config`: Es un archivo escrito en lenguaje XML[37] que permite añadir la configuración relacionada a la ejecución de la aplicación web.
- Otros: La aplicación web contiene otros archivos, los cuales no serán descritos, debido a que no se usan en este proyecto. Se recomienda revisar [25] para conocer sobre estos.

En ASP.NET MVC, un controlador es usado para definir un conjunto de acciones. Una acción es un método en un controlador que maneja solicitudes entrantes. Las solicitudes entrantes son asociadas a acciones a través de rutas.

El **Segmento de código 2.1** muestra la ruta de MVC configurada por defecto. En la línea 17 se le asigna el nombre de la ruta `Default`, en la 18 se configura la ruta y en la 19 se asigna los valores por defecto asociados a esa ruta. MVC.NET intentará extraer los valores de ruta de la URL a la cual el usuario accedió en la aplicación web.

Por ejemplo, cuando un usuario accede a la URL `Home/Index`, el framework de MVC.NET trata de buscar el controlador `Home` y la acción `Index`, y si no la encuentra, lanzará una excepción. Si la URL no especifica los valores de ruta, MVC usa los valores por defecto, en este caso se ejecutará la acción `Login` del controlador `Account`.

El controlador estará encargado de procesar las solicitudes entrantes a través de sus acciones y de devolver la apropiada vista u otro tipo de contenido. Todo método

---

[36] *Nuget* es un sistema que permite administrar paquetes de código abierto en proyectos de Visual Studio. Un paquete *nuget* es aquel que puede contener librerías DLL, archivos CSS o JavaScript. Cuando un paquete nuget es instalado en una aplicación de .NET, los archivos contenidos en él son descargados y añadidos al proyecto.

[37] XML (eXtensible Markup Language) es un lenguaje estructurado desarrollado por W3C que permite almacenar datos.

público en un controlador es una acción [26]. Los parámetros de los métodos tipo acción son enlazados a las solicitudes de entrada y validados usando *model binding*[38].

```
16.     routes.MapRoute(
17.         name: "Default",
18.         url: "{controller}/{action}/{id}",
19.         defaults: new { controller = "Account", action = "Login", id =
        UrlParameter.Optional }
20.     );
```

Segmento de código 2.1 Ruta de MVC por defecto definida en la clase `RouteConfig`

El **Segmento de código 2.2** muestra la implementación de la acción `Login` del controlador `AccountController`. La línea 35 muestra como enviar información desde el controlador a una vista. Un `ViewBag` funciona como un contenedor de información, el cual es poblado en el controlador para poder ser consumido en la vista. La línea 36 muestra un ejemplo del uso del *helper* `View`, el cual sirve para retornar una vista. Una vista de MVC genera código que será dibujado en el explorador web. El *helper* puede ser configurado para retornar cualquiera de las vistas, pero en caso de que no se especifique que vista retornar, MVC.NET reconoce que debe retornar aquella cuyo nombre sea el mismo que el de la acción. Es decir, en el ejemplo anterior, la acción retornará la vista `Login`.

```
33.     public ActionResult Login(string returnUrl)
34.     {
35.         ViewBag.ReturnUrl = returnUrl;
36.         return View();
37.     }
```

Segmento de código 2.2 Acción `Login` del controlador `AccountController`

---

[38] *Model binding* permite asociar datos desde una petición HTTP con los parámetros de una acción. *Model binding* intentará cargar los parámetros de entrada de la acción usando los datos de la petición HTTP lo que más pueda. Si no logra cargar todos los valores, intentará dejarlos en nulo, pero si el parámetro no acepta nulos, lanzará una excepción.

En el patrón de diseño MVC, la vista encapsula los detalles de presentación de la interacción del usuario con la aplicación web. En MVC.NET, las vistas funcionan como plantillas con código HTML, JavaScript y CSS. Las vistas también permiten añadir código C# embebido, el cual permite a su vez generar más contenido dentro de ella. Las vistas usan la extensión de archivo CSHTML y típicamente están asociadas a un controlador.

Existen dos tipos especiales de vistas:

- Vista parcial: Es aquella vista que se dibuja dentro de otra. Una vista de MVC debe implementar el método `@Html.Partial("NombreDeVistaParcial")` en la sección donde se requiere dibujar la vista parcial.
- *Layout*: Es un tipo de vista usado para implementar estructuras de código HTML, JavaScript o CSS, comunes para varias interfaces de usuario. El *layout* típicamente incluye elementos comunes de las interfaces de usuario como encabezado, menú de navegación y pie de página. También es muy común que en el *layout* se incluyan las referencias a archivos JavaScript y CSS, para evitar añadirlas en cada vista. Para construir una interfaz de usuario cuya estructura incluya una vista dentro de un *layout*, es necesario que la vista tenga referencia al *layout* elegido. El **Segmento de código 2.3** muestra un ejemplo de cómo referenciar un *layout* desde una vista MVC.

Una interfaz de usuario se compone de vista, *layout* y opcionalmente de vistas parciales. La **Figura 2.26** muestra la estructura típica de una interfaz de usuario.

```
1.    @{
2.        Layout = "~/Views/Shared/_Layout.cshtml";
3.    }
```

Segmento de código 2.3 Referencia a *layout* `_Layout.cshtml` de la vista `Login`

En la aplicación se tendrá interfaces que no implementan un menú de navegación, e interfaces que si lo hacen, por lo que se vio la necesidad de construir 2 *layouts*.

Los 2 *layouts* que se implementaron son los siguientes:

- `_Layout.cshtml`: Este *layout* no contiene menú de navegación y será usado para construir las interfaces `Register` y `Login`.
- `_LayoutMenu.cshtml`: Este *layout* si incluye una implementación de un menú de navegación, y será usado para construir todas interfaces de usuario excepto `Register` y `Login`.

El **Segmento de código 2.4** muestra la implementación del *layout* `_Layout.cshtml`. El *layout* está compuesto mayormente por código HTML. La declaración de la línea 1 es una instrucción usada para que el explorador web interprete que se trata de un documento HTML, desde la línea 3 hasta la línea 14 se construye la cabecera del documento HTML y desde la 15 hasta la 41 se construye el cuerpo HTML (*body*).

En las líneas 6, 7 y 8 de la cabecera HTML se muestra la manera de inyectar código de servidor[39] en una vista MVC. El *framework* MVC.NET reconoce que el código de servidor inyectado en la vista ya que este está precedido del símbolo `@`. La línea 6 imprime el valor de la variable `ViewBag.Title` en la etiqueta `title`, como se mencionó anteriormente, un `ViewBag` permite enviar información desde el controlador hacia la vista.

Las líneas 7 y 8 añaden referencias a archivos CSS y JavaScript respectivamente, cuyo funcionamiento será explicado más adelante. Desde la línea 16 hasta la línea 30 se construye una barra de navegación horizontal, que se despliega en la parte superior de la interfaz de usuario. La línea 32 muestra la llamada al método `Render`, el cual permite insertar el contenido que arroja la vista que tiene referencia al *layout* en cuestión.

Desde la línea 34 hasta la línea 36 se construye un píe de página en la interfaz de usuario, cabe mencionar que el pie de página siempre va a estar por debajo de la vista y de otras vistas parciales que se dibujen en la interfaz, esto es porque el método `Render` está colocado sobre el pie de página. Las líneas 38 y 39 añaden referencias a archivos de JavaScript.

---

[39] Código de servidor se refiere al código que se ejecuta del lado del servidor.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <meta charset="utf-8" />
5.      <meta name="viewport" content="width=device-width, initial-scale=1.0">

6.      <title>@ViewBag.Title</title>
7.      @Styles.Render("~/Content/css")
8.      @Scripts.Render("~/bundles/modernizr")
9.      <style>
10.         body{
11.         background-color: white;
12.     }
13.     </style>
14. </head>
15. <body>
16.     <div class="navbar navbar-inverse navbar-fixed-top">
17.         <div class="container">
18.             <div class="navbar-header">
25.             </div>
26.             <div class="navbar-collapse collapse">
27.                 <ul class="nav navbar-nav"></ul>
28.             </div>
29.         </div>
30.     </div>
31.     <div class="container body-content">
32.         @RenderBody()
33.         <hr />
34.         <footer>
35.             <p>&copy; @DateTime.Now.Year - Escuela Politécnica Nacional</p>
36.         </footer>
37.     </div>
38.     @Scripts.Render("~/bundles/jquery")
39.     @Scripts.Render("~/bundles/bootstrap")
40.     @RenderSection("scripts", required: false)
41. </body>
42. </html>
43.
```

Segmento de código 2.4 Implementación de *layout* `_Layout.cshtml`

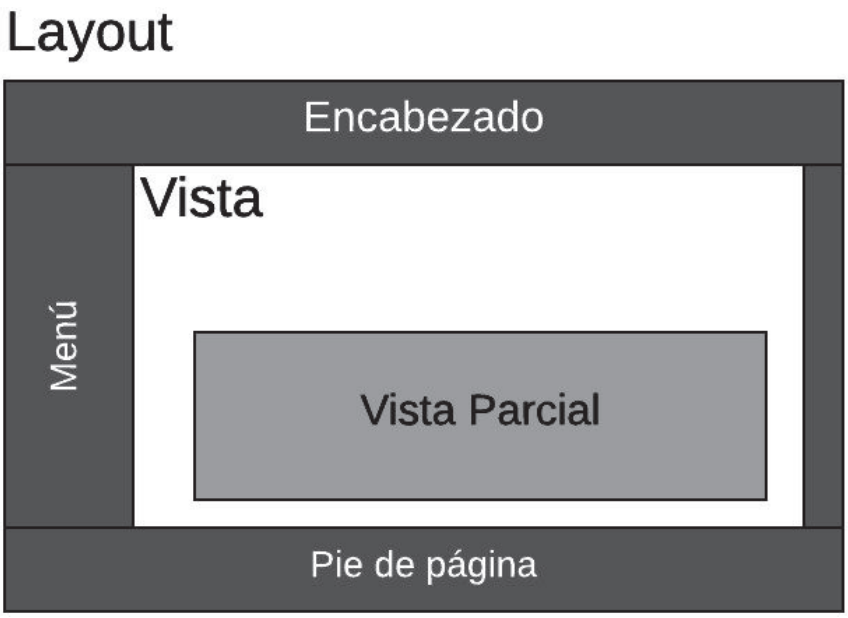


Figura 2.26 Estructura típica de una interfaz de usuario en MVC.NET

Finalmente, la línea 40 permite inyectar código de cliente JavaScript proveniente de la vista dentro del *layout*. Los métodos `Scripts.Render` y `Styles.Render` permiten añadir referencias a archivos JavaScript y CSS respectivamente en el *layout* usando la técnica de agrupación de archivos.

En la línea 7 del **Segmento de código 2.4** se referencia al grupo de archivos CSS ubicadas en `~/Content/css`. Este grupo de archivos se lo definió en la clase `BundleConfig` de la aplicación MVC.NET.

El **Segmento de código 2.5** muestra la definición del agrupamiento de archivos CSS `~/Content/css`, donde se han incluido los siguientes archivos:

- `bootstrap.css`: Archivo CSS del *framework* twitter Bootstrap.
- `site.css`: Archivo CSS que contiene estilos propios de la aplicación web.
- `sb-admin.css`: Archivo CSS que contiene estilos globales de la plantilla `sb-admin` implementada en la aplicación web.
- `morris.css`: Archivo CSS que contiene estilos para la implementación de gráficos en la interfaz web.
- `font-awesome.css`: Archivo CSS de la herramienta Font Awesome, que permite desplegar íconos en la interfaz de usuario.

```
27.     bundles.Add(new StyleBundle("~/Content/css").Include(
28.                 "~/Content/bootstrap.css",
29.                 "~/Content/site.css",
30.                 "~/Content/sb-admin.css",
31.                 "~/Content/plugins/morris.css",
32.                 "~/font-awesome/css/font-awesome.css"));
```

Segmento de código 2.5 Agrupamiento de archivos CSS `~/Content/css`

Cuando se publica la aplicación web en el ambiente de producción, MVC.NET genera un nuevo archivo CSS en el servidor web, en el cual une todo el código CSS de cada uno de los archivos que pertenecen al grupo.

Luego, cuando un usuario accede a la interfaz de usuario de la aplicación web, el servidor web no requiere enviar los 5 archivos CSS del grupo al explorador web, sino solamente un único archivo que contiene el código de los 5. Esto genera una mejora en el rendimiento de la aplicación web.

En MVC, un modelo es una clase, con propiedades, que hace las veces de contenedor para el envío de información desde el controlador a la vista, y para recoger la información que proviene de la vista y llega al controlar como parámetro. La vista MVC dibuja controles HTML asociados a las propiedades del modelo de MVC al que hace referencia.

MVC.NET permite añadir metadatos asociada a las propiedades del modelo a través de atributos. Por definición, los metadatos son datos sobre otros datos. Los metadatos ayudan a MVC.NET a entender que tipo de control se quiere dibujar en la vista con base en la propiedad definida en el modelo.

Los metadatos también permiten implementar validaciones de la capa de presentación, como validar requeridos, o inclusive validar que un texto tenga un formato adecuado.

Un ejemplo de validación de la capa de presentación es el que se muestra tanto en la línea 51 como la línea 56 del **Segmento de código 2.6**. El atributo `Required` sobre las propiedades `Email` (línea 54) y `Password` (línea 59) del modelo `LoginViewModel`, obliga a que los controles HTML construidos en una vista que están asociados a esas propiedades sean requeridos al momento de enviar el

pedido hacia el servidor web. La línea 53 del **Segmento de código 2.6** valida que la propiedad *Email* tenga el formato de una dirección electrónica válida. El atributo `Display` implementado en las líneas 52, 58 y 61 permite definir un texto asociado a cada propiedad, con el objetivo de que sea mostrado en la interfaz de usuario, en vez de mostrar el nombre de la propiedad del modelo.

```
49.     public class LoginViewModel
50.     {
51.         [Required]
52.         [Display(Name = "Email")]
53.         [EmailAddress]
54.         public string Email { get; set; }
55.
56.         [Required]
57.         [DataType(DataType.Password)]
58.         [Display(Name = "Password")]
59.         public string Password { get; set; }
60.
61.         [Display(Name = "Remember me?")]
62.         public bool RememberMe { get; set; }
63.     }
```

Segmento de código 2.6 Modelo de MVC `LoginViewModel`

El

Segmento de código 2.7 muestra una sección de código de la vista `Login.cshtml`. En la línea 1, se indica que la vista usa el modelo `LoginViewModel`, cuya implementación se muestra en el **Segmento de código 2.6**, y en la línea 4 se añade la referencia del *layout* en la vista `Login.cshtml`.

```
1.    @model LoginViewModel
2.    @{
3.        ViewBag.Title = "Log in";
4.        Layout = "~/Views/Shared/_Layout.cshtml";
5.    }
```

Segmento de código 2.7 Sección de código de la vista `Login.cshtml` donde se especifica el modelo a usar por la vista y la referencia al *layout*

La vista `Login.cshtml` no dibuja vistas parciales por lo que no tiene llamadas al método `Render`. El

Segmento de código 2.8 muestra otra sección de la vista `Login.cshtml`, en la línea 2 se llama al el método `LabelFor`, el cual imprime en la vista una etiqueta HTML `label` con el texto definido en el atributo `[Display(Name = "Password")]` de la propiedad `Password`. Si la propiedad no posee el atributo, el método `LabelFor` imprimirá el control `label` con el nombre de la propiedad.

La línea 4 muestra el uso del método `PasswordFor`, el cual devuelve un control HTML correspondiente a una caja de texto de tipo `password` y lo dibuja en la vista, el control HTML es dibujado usando los metadatos definidos en el modelo. En la línea 5 se llama al método `ValidationMessageFor` el cual imprime en la vista los errores generados cuando alguna de las validaciones relacionadas a la propiedad `Password` falló.

```
1.    <div class="form-group">
2.        @Html.LabelFor(m => m.Password, new { @class = "col-md-2
      control-label" })
3.      <div class="col-md-10">
4.        @Html.PasswordFor(m => m.Password, new { @class = "form-control"
      })
5.        @Html.ValidationMessageFor(m => m.Password, "", new { @class =
      "text-danger" })
6.      </div>
7.    </div>
```

Segmento de código 2.8 Sección de código de la vista `Login.cshtml` que permite dibujar la fila que contiene el campo `Password`

El

Segmento de código 2.9 muestra la sección de la vista después de que MVC.NET la procesó y la envió al explorador web. La línea 2 muestra la etiqueta HTML `label` generada por el método `LabelFor`, la línea 4 muestra la caja de texto de tipo `password` generada por el método `PasswordFor`, y la línea 5 muestra el resultado

que imprime el método `ValidationMessageFor` el cual es una etiqueta HTML `span`, donde se crearán los mensajes de error en caso de que haya alguno.

```
1.    <div class="form-group">
2.       <label class="col-md-2 control-label"
      for="Password">Password</label>
3.       <div class="col-md-10">
4.          <input name="Password" class="form-control" id="Password"
      type="password" data-val-required="The Password field is required."
      data-val="true">
5.          <span class="field-validation-valid text-danger" data-valmsg-
      replace="true" data-valmsg-for="Password"></span>
6.       </div>
7.    </div>
```

Segmento de código 2.9 Sección de código de la vista Login.cshtml una vez que fue procesada y dibujada en el explorador web

Un controlador de MVC posee las acciones necesarias para el manejo de las peticiones entrantes al servidor web. Cuando un usuario accede a una interfaz de usuario, el servidor interpreta la petición como una petición `GET`, por el contrario, cuando un usuario envía contenido HTML desde la interfaz de usuario hacia el servidor web, el servidor interpreta la petición como `POST`.

MVC.NET posee un mecanismo para manejar la petición `GET` y `POST` de la misma vista, a través del atributo `HttpPost`.

El **Segmento de código 2.10** muestra la acción `Login` del controlador `Account` para el manejo de peticiones `GET`, mientras que el **Segmento de código 2.10** muestra la acción `Login` del mismo controlador pero que maneja peticiones `POST`. La diferencia radica en el atributo `HttpPost` (línea 39) el cual le indica al *framework* MVC.NET cual acción debe ejecutar según el tipo de petición.

La línea 40 del **Segmento de código 2.11** muestra la firma de la acción `Login`. Cabe mencionar que la acción retorna un objeto de tipo `Task<ActionResult>` en vez de retornar un objeto de tipo `ActionResult`.

Un objeto `Task` o tarea representa una operación asincrónica, esto quiere decir que es una operación que puede procesarse simultáneamente mientras se procesan otras operaciones, en otras palabras, el servidor no espera a concluir una operación para iniciar otra. A este funcionamiento se le denomina paralelismo de tareas, y proporciona la ventaja de un uso más eficaz de los recursos del sistema.

También cabe resaltar que el *model binding* de MVC es el encargado de leer la información embebida en el documento HTML, e introducirlo como parámetro de tipo `LoginViewModel`.

```
32. public ActionResult Login(string returnUrl)
33. {
34.     ViewBag.ReturnUrl = returnUrl;
35.     return View();
36. }
```

Segmento de código 2.10 Acción `Login` del controlador `Account` que maneja peticiones `GET`

```
39.  [HttpPost]
40. public async Task<ActionResult> Login(LoginViewModel model, string
    returnUrl)
41. {
42.     …
43.     return View(model);
44. }
```

Segmento de código 2.11 Acción `Login` del controlador `Account` que maneja peticiones `POST`

Durante este *sprint* se construyeron las interfaces de usuario `Login`, `Register` y `Home`, para para lo cual se implementaron los elementos de MVC.NET que se muestran en la **Tabla 2.38**.

Para mejorar la presentación de las interfaces de usuario, y para implementar un menú de navegación en el *layout* `_LayoutMenu.cshtml`, se tomó como base la plantilla web `SB admin` del portal Start Bootstrap [27].

Para integrar la plantilla con las interfaces web a ser desplegadas en la aplicación web se realizaron los siguientes pasos:

- Se copiaron los archivos JavaScript y CSS de la plantilla dentro de la solución de la aplicación MVC.NET.
- Se añadieron las referencias a los nuevos archivos JavaScript y CSS dentro de los 2 *layout* implementados usando la técnica de agrupamiento de archivos.
- Se copió el código HTML que conforma la estructura del menú de navegación vertical de la plantilla dentro del *layout* `_LayoutMenu.cshtml`.
- Se copió el código HTML que conforma la estructura del panel de navegación horizontal superior de la plantilla dentro del *layout* `_LayoutMenu.cshtml`.
- Se modificó el código HTML copiado en los *layout* según los requerimientos del sistema, y siguiendo los *mockup* de las interfaces.

Tabla 2.38 Elementos de MVC.NET usados para la construcción de las interfaces de usuario `Login`, `Register` y `Home`

Interfaz de usuario	Controlador	Vista	Layout
`Login`	`Account`	`Login`	`_Layout.cshtml`
`Register`	`Account`	`Register`	`_Layout.cshtml`
`Home`	`Home`	`Index`	`_LayoutMenu.cshtml`

2.3.3.2.2 Proyectos de Visual Studio

En el *sprint* 1 se creó un prototipo[40] de servicio WCF. El servicio se llama `PatientService`, será parte de la capa de servicios y expondrá algunas operaciones que devuelven información médica, para ello también se crearán las siguientes capas de prueba: capa de negocio (BLL) y capa de acceso a datos (DAL). La capa de acceso a datos se conectará con la base de datos médica.

[40] Prototipo es una representación limitada de un software que sirve de modelo para la construcción del software final.

Para probar el funcionamiento del servicio web WCF se implementará una aplicación de consola que pueda consumir el servicio.

Se crearon 5 proyectos de Visual Studio para la creación de las capas DAL, BLL, servicios y presentación de prueba. Los 5 proyectos son: `DAL`, `BLL`, `BusinessEntities`, `PatientService` y `ConsoleApplication`.

La **Figura 2.27** indica los proyectos construidos y la interacción entre ellos. La aplicación de consola pertenece a la capa de presentación, y se usará para realizar pruebas, la cual será reemplazada por la aplicación web MVC.NET en los siguientes *sprints*.

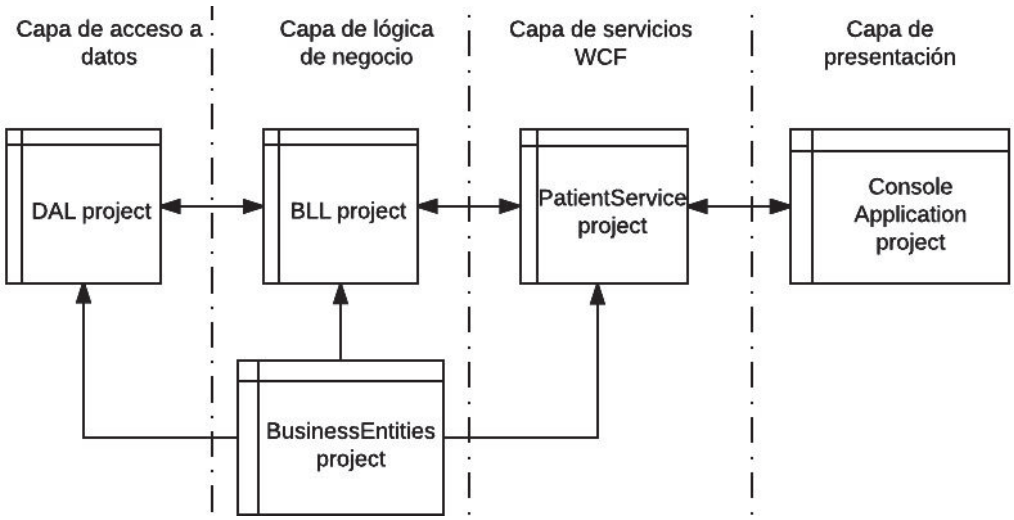


Figura 2.27. Proyectos de Visual Studio construidos para implementación del prototipo

2.3.3.2.3 Capa de acceso a datos

A la capa de datos pertenece el modelo de datos de entidades. El modelo de datos de entidades se lo crea usando Entity Framework. Cabe mencionar que solo las tablas de la base de datos `MedicalDatabase` son asociadas al modelo de entidades. Para ello se siguieron los siguientes pasos:

1. Abrir Visual Studio.
2. Ir a la barra de herramientas y seleccionar: FILE → New → Project.
3. Seleccionar *template* = librería de clases, y presionar OK.
4. La solución se crea con un proyecto de librería de clases.

5. En el explorador de la solución, *click* derecho sobre el proyecto, seleccionar opción Add → New item… → ADO.NET Entity Data Model.
6. Seleccionar la opción EF Designer from database.
7. Configurar la conexión de datos (aquí se configura el servidor de bases de datos y el nombre de la base de datos `MedicalDatabase`).
8. Finalmente seleccionar todas las tablas de la base de datos `MedicalDatabase` para ser asociadas al modelo de datos de entidades. Se crea el modelo de entidades `MedicalDatabase.edmx`.

El modelo de datos de entidades de ADO.NET `MedicalDatabase.edmx` añadido a la solución de Visual Studio dispone de dos subcapas [28], estas son: subcapa de modelado, y subcapa de objetos. La subcapa de modelado consta de 3 componentes, los cuales son:

- Un esquema de base de datos, que define las tablas y las relaciones de la base de datos `MedicalDatabase`.
- Un modelo conceptual el cual consiste de entidades de dominio y relaciones, basados en el modelo de datos de entidad de Entity Framework.
- Una asociación entre el esquema de base de datos y el modelo conceptual.

La subcapa de modelado está implementada en un archivo escrito en código XML, y cada componente corresponde a una sección del mismo. Entity Framework usa el componente de asociación para transformar operaciones sobre los objetos entidades (como crear, leer, actualizar y borrar) en su equivalente operación de la base de datos.

La subcapa de objetos contiene los objetos correspondientes a sus entidades de dominio y sus relaciones, las cuales fueron generadas en el modelo conceptual de la subcapa de modelado. Es decir, en la base de datos médica existe la tabla `dbo.Patient`, la subcapa de modelado asocia la tabla con una entidad de dominio, luego a través de la entidad de dominio se crea la clase `Patient` de C#, y a través de esta se instancias objetos de tipo `Patient`. La **Figura 2.28** muestra la interacción entre las capas y sus componentes del modelo de datos de entidades de ADO.NET. Los objetos de la subcapa de objetos se denominan objetos de negocio o de dominio.

Un contexto de base de datos es la puerta de acceso a una base de datos usando objetos C#. La **Figura 2.28** muestra gráficamente como se guarda la información contenida en el objeto `Patient` en un registro en la tabla `dbo.Patient` de la base de datos médica a través del contexto.

Entity Framework permite cargar objetos de negocio desde la base de datos mediante diferentes técnicas [29], facilitando así el proceso de desarrollo. Las técnicas son las siguientes:

1. *Eagerly loading* (cargado ansioso) que carga el objeto relacionado como parte de una consulta a la base de datos a través de la implementación del método `Include`.
2. *Lazy loading* (cargado perezoso) que carga automáticamente las entidades relacionadas desde la base de datos la primera vez que una propiedad que refiere a la entidad es accedida. Esta opción no es válida cuando se requiere serializar los objetos.

Dada la naturaleza de los servicios web, en la cual se requiere serializar los objetos antes de ser enviados al cliente web, es contraproducente usar la opción *lazy loading*.

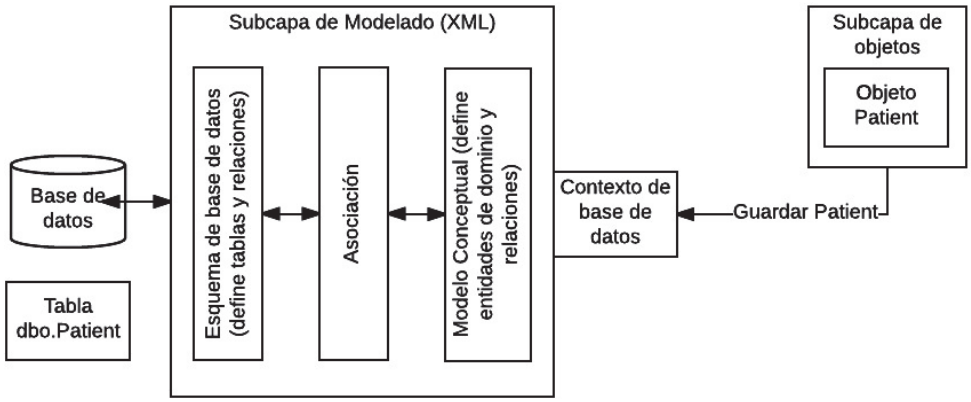


Figura 2.28 Subcapas del modelo de datos de entidades de ADO.NET

Finalmente, es necesario configurar a que base de datos se va a conectar la capa de modelado. La sección `connectionStrings` del archivo de configuración del proyecto de Visual Studio, permite añadir configuraciones de conexión a la base de datos. En una configuración de conexión a base de datos se especifica el nombre de la conexión, el servidor de base de datos, el nombre de la base de datos, el tipo

de base de datos (por ejemplo, SQL *Server*), el nombre de usuario a conectarse y la contraseña, entre otros.

En el sistema se implementó el contexto de base de datos `MedicalInfoDatabaseEntities` el cual será la puerta de entrada a la base de datos médica. El **Segmento de código 2.12** muestra la implementación de dicho contexto. Un contexto de base de datos se implementa a través de una clase de C#, la cual debe derivarse de la clase `DbContext`, tal como se muestra en la línea 1. Entre la línea 3 y 6 se implementa el constructor, el cual llama al constructor de la clase base, y le envía como parámetro el nombre de la conexión de la base de datos. Entre las líneas 8 y 11, se redefine el método `OnModelCreating` de la clase `DbContext`.

```
1.      public partial class MedicalInfoDatabaseEntities : DbContext
2.      {
3.          public MedicalInfoDatabaseEntities()
4.              : base("name=MedicalInfoDatabaseEntities")
5.          {
6.          }
7.
8.          protected override void OnModelCreating(DbModelBuilder
    modelBuilder)
9.          {
10.             Configuration.LazyLoadingEnabled = false;
11.         }
12.
13.         public virtual DbSet<Allergy> Allergies { get; set; }
14.         public virtual DbSet<TestPatient> TestPatients { get; set; }
15.         public virtual DbSet<VascularAccessDevice> VascularAccessDevices
    { get; set; }
16.         public virtual DbSet<Immunization> Immunizations { get; set; }
17.         public virtual DbSet<Infection> Infections { get; set; }
18.         public virtual DbSet<Medication> Medications { get; set; }
19.         public virtual DbSet<Treatment> Treatments { get; set; }
20.         public virtual DbSet<TestPhysician> TestPhysicians { get; set; }
21.     }
22. }
```

Segmento de código 2.12 Implementación del contexto de base de datos `MedicalInfoDatabaseEntities`

El método `OnModelCreating` permite añadir configuraciones relacionadas a Entity Framework, en la línea 10, se desactiva la técnica de carga de objetos *Lazy loading*. Finalmente, entre las líneas 13 y 20 se muestran las colecciones de todas las entidades del contexto de base de datos. Cada colección permite acceder a cada una de las tablas de la base de datos médica.

El otro componente de la capa de acceso a datos es el de repositorios. La lista de los repositorios a implementarse ya fue definida en la sección de diseño. Siguiendo el principio DRY, se implementó los repositorios de tal manera que los no genéricos hereden del repositorio genérico `GenericDataRepository`.

En el **Segmento de código 2.13** se muestra la implementación del método `GetById` del repositorio `GenericDataRepository`. En la línea 67, la declaración virtual permite implementar un método virtual, es decir, un método que se puede sustituir las clases derivadas.

El tipo `T` de la línea 67 indica que es un tipo de clase que puede ser reemplazado por cualquiera de los tipos de clases de negocio resultantes del modelado de EntityFramework, como por ejemplo el tipo `Allergy`.

Entre la línea 71 y 74, la declaración `using` permite crear una instancia del contexto de base de datos, usarla, y desecharla una vez que termina esta declaración, para que no siga ocupando recursos en memoria, para que un objeto sea desechable, debe implementar la interfaz `IDisposable`, como lo hace `MedicalInfoDatabaseEntities`.

En la línea 73 del **Segmento de código 2.13**, el método `Set` devuelve la colección de entidades del tipo `T` genérico, y el método `Find` permite encontrar la entidad en base a la clave primaria dada. La línea 76 permite retornar el objeto encontrado.

El **Segmento de código 2.14** muestra la implementación del método `GetAllergiesByPatient` del repositorio `AllergyRepository`. En la línea 3, se llama al método `GetList` de la clase base `GenericDataRepository`. El método `GetList` recibe como parámetro una sentencia lambda, donde se le

especifica que retorne todas las entidades de la colección en las cuales la propiedad `PatientID` sea igual al valor del parámetro de entrada `patientID`.

```
67.    public virtual T GetById(int id)
68.    {
69.        T item = null;
70.
71.        using (var context = new MedicalInfoDatabaseEntities())
72.        {
73.            item = context.Set<T>().Find(id);
74.        }
75.
76.        return item;
77.    }
```

Segmento de código 2.13 Operación `GetById` del repositorio genérico `GenericDataRepository`

```
1.     public IList<Allergy> GetAllergiesByPatient(int patientID)
2.     {
3.         var result = GetList(t => t.PatientID == patientID);
4.         return result;
5.     }
```

Segmento de código 2.14 Implementación del método `GetAllergiesByPatient` del repositorio `AllergyRepository`

2.3.3.2.4 Capa de negocio

Para implementar la capa de negocio se requiere de 2 proyectos de Visual Studio: `BusinessEntities` que contienen las entidades de negocio, y el proyecto `BLL` donde se maneja el procesamiento y las validaciones de negocio.

La creación de un proyecto exclusivo para el manejo de entidades de negocio permite desacoplar dichas entidades de la capa de acceso a datos y moverla a la capa de negocio. Este proyecto se construyó trasladando la capa de objetos de EntityFramework hacia un nuevo proyecto de librería de clases de Visual Studio. La **Figura 2.29** muestra los archivos que forman parte del proyecto de Visual Studio

`BusinessEntities`. Los proyectos `DAL`, `BLL` y `PatientService` deben tener referencia al proyecto `BusinessEntities` para que compile la solución de Visual Studio.

En el proyecto `BLL` se implementan las operaciones de negocio que consumen los repositorios implementados en la capa de acceso a datos, para la cual se definió la clase `Business`. En el **Segmento de código 2.15** se muestra la implementación del método `GetAllergiesByPatientID` de la clase `Business`, el cual retorna las alergias de un paciente. En la línea 3 se crea una nueva instancia del repositorio `AllergyRepository`, luego en la línea 4 se llama al método `GetAllergiesByPatient` del repositorio, el cual retorna todas las alergias de un paciente. Finalmente, la capa de negocio retorna ese resultado obtenido del repositorio.

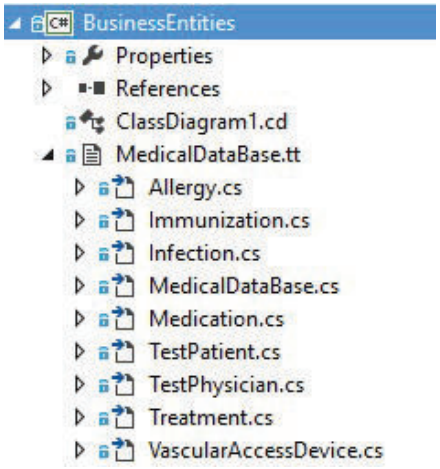


Figura 2.29 Archivos creados en el proyecto `BusinessEntities` de la capa de negocio

```
1. public IList<Allergy> GetAllergiesByPatientID(int patientID)
2. {
3.     var _allergyRepository = new AllergyRepository();
4.     var result = _allergyRepository.GetAllergiesByPatient(patientID);
5.     return result;
6. }
```

Segmento de código 2.15 Método `GetAllergiesByPatientID` de la clase `Business` de la capa de negocio

2.3.3.2.5 Capa de servicios

En la capa de servicios se implementarán tres servicios WCF: `PatientService`, `PhysicianService` y `ReportService`. Para este *sprint* solo se implementó el servicio `PatientService`, para lo cual se creó un proyecto de aplicación de servicio WCF de Visual Studio siguiente los siguientes pasos:

1. En el explorador de la solución de Visual Studio, botón derecho sobre la carpeta, y seleccionar Add → New Project…
2. En la nueva ventana seleccionar la opción WCF → Service Application.
3. Escribir el nombre del proyecto y seleccionar la opción Add.

El proyecto de Visual Studio se lo nombró `PatientService`. Este proyecto debe tener referencias a los proyectos `BusinessEntities` y `BLL`, ya que consumirá las operaciones de negocio que están definidas en la capa de negocio.

Para que una operación de servicio sea visible por un cliente de servicio, se debe crear una interfaz con el atributo `[ServiceContract]`, dentro de ella definir la operación con el atributo `[OperationContract]`, caso contrario la operación no podrá ser vista por los clientes WCF. En el **Segmento de código 2.16** se muestra un ejemplo de la interfaz de operación del servicio con sus respectivos atributos, mientras que en el **Segmento de código 2.17** se muestra la implementación de la interfaz. `PatientService` debe implementar la interfaz `IPatientService` obligatoriamente.

```
1.  namespace PatientService
2.  {
3.      [ServiceContract]
4.      public interface IPatientService
5.      {
6.          [OperationContract]
7.          Allergy GetAllergyByID(int ID);
8.      }
9.  }
```

Segmento de código 2.16 Contrato de operación `GetAllergyByID` del servicio `PatientService`

En la línea 7 del **Segmento de código 2.17** se instancia el objeto `Business` que pertenece a la capa de negocio, en la línea 8 se llama al método `GetAllergyByID` de la capa de negocio y en la línea 10 se retorna el valor obtenido. Para revisar el resto del código, acudir al Anexo A.

2.3.3.2.6 Aplicación de consola para pruebas

Se añadió un proyecto de aplicación de consola a la solución de Visual Studio para realizar las pruebas del servicio WCF `PatientService`.

```
1.    namespace PatientService
2.    {
3.        public class PatientService : IPatientService
4.        {
5.            public Allergy GetAllergyByID(int ID)
6.            {
7.                var business = new Business();
8.                var allergy = business.GetAllergyByID(ID);
9.
10.               return allergy;
11.           }
12.       }
13.   }
```

Segmento de código 2.17 Implementación de operación `GetAllergyByID` del servicio `PatientService`

Para levantar y verificar que el servicio `PatientService` está funcionando correctamente, se accede al explorador de solución de Visual Studio, luego dentro del proyecto de aplicación WCF `PatientService` se selecciona el archivo `PatientService.svc`, luego *click* derecho sobre el mismo y seleccionar la opción *View in browser*. En ese momento, Visual Studio intentará levantar el servicio en el IIS *express*[41], y en caso de que el proceso de levantar el servicio haya

[41] IIS (*Internet Information Services*) es un servidor web del sistema operativo Windows. El IIS permite publicar aplicaciones web, para su ejecución. IIS *express* es una versión ligera de IIS, usada por Visual Studio para ejecutar aplicaciones web.

sido exitoso, se abrirá el explorador web donde se muestra una interfaz similar a la de la **Figura 2.30**.

Para consumir el servicio WCF desde una aplicación de consola, se agrega un nuevo proyecto de consola a la solución de Visual Studio, luego *click* derecho sobre el proyecto nuevo, seleccionar la opción *Add* → *Service Reference*, y la ventana *Add Service Reference* se abrirá, tal como se muestra en la **Figura 2.31**.

En el campo *Address* se debe añadir la dirección del servicio, y luego de hacer *click* sobre la opción *Go*, se mostrará todas las operaciones que el servicio expone. El siguiente paso es presionar OK y la referencia de servicio será añadida al proyecto de consola.

El **Segmento de código 2.18** muestra un ejemplo de consumo de un servicio WCF, desde la aplicación de consola. Este código fue llamado desde la clase `Program`. En la línea 33 se instancia un nuevo cliente del servicio WCF, en la línea 34 se consume una de las operaciones del servicio, y en la línea 36 se imprime en consola el resultado obtenido desde el servicio.

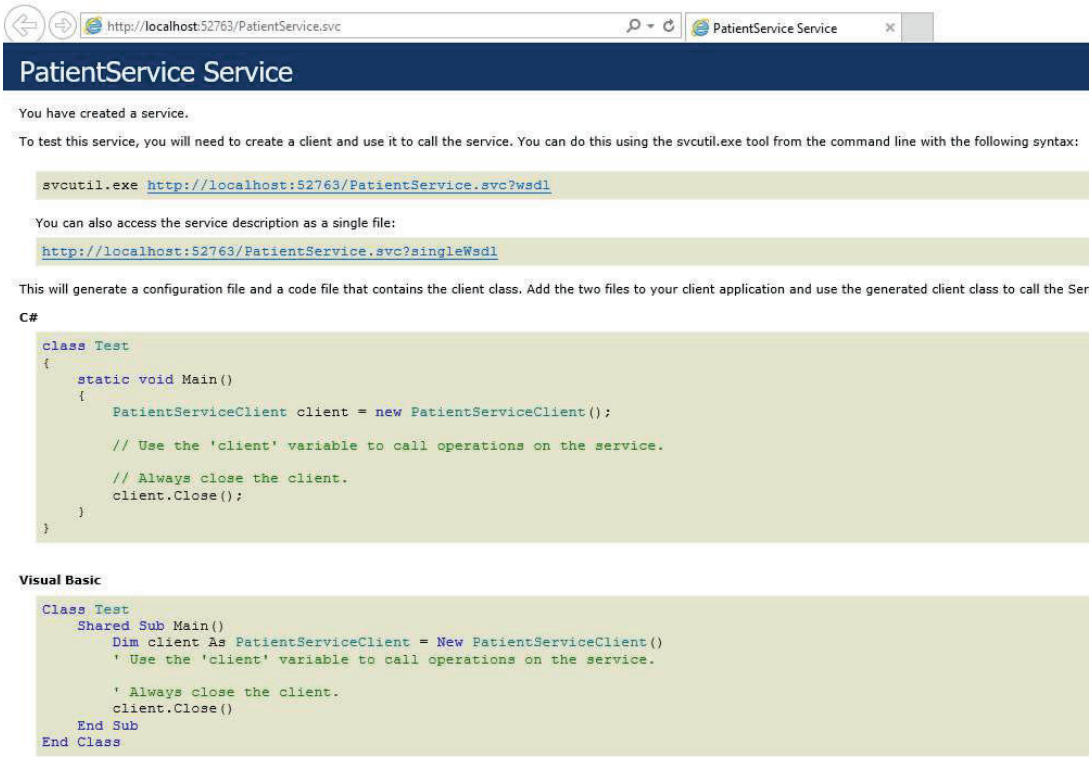


Figura 2.30 Interfaz que indica que un servicio WCF está funcionando

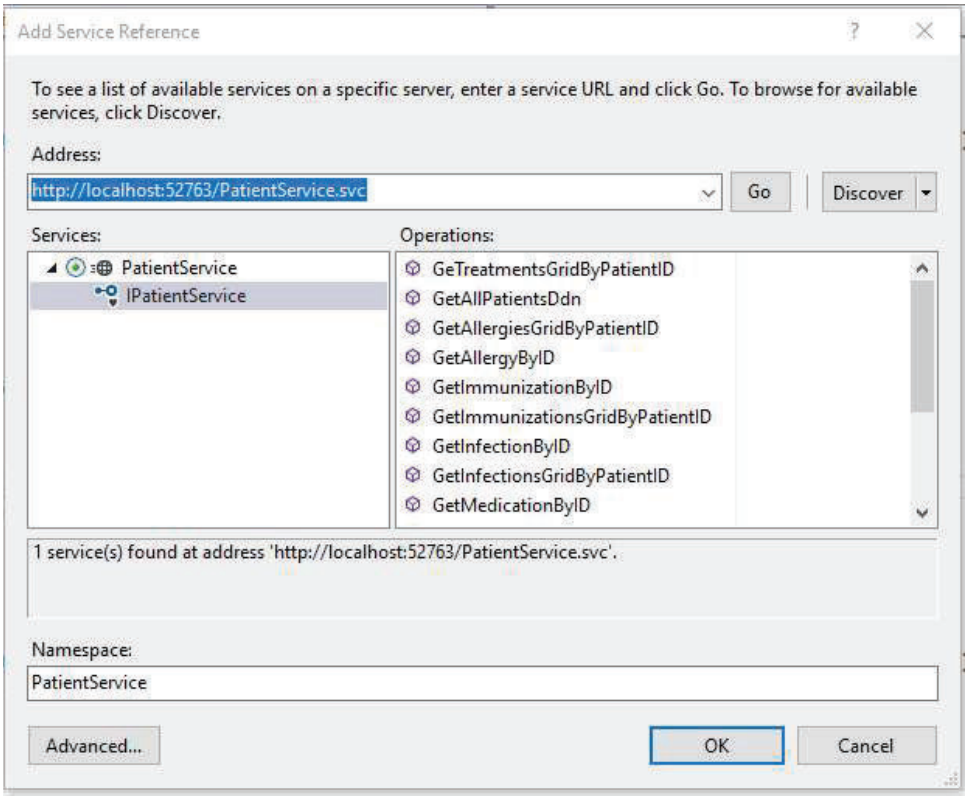


Figura 2.31 Interfaz para añadir referencia de servicio en Visual Studio

```
32.  private static void GetImmByID(){
33.      var client = new ServiceReference1.PatientServiceClient();
34.      var imm = client.GetImmunizationByID(3);
35.
36.      Console.WriteLine("Reading from Patient Service..." +
     imm.Immunization1);
37.  }
```

Segmento de código 2.18 Ejemplo de consumo de operación de servicio WCF desde aplicación de consola

2.3.4 SPRINT 2

2.3.4.1 Sprint planning

En la **Tabla 2.39** se muestra los ítems escogidos para cumplir con los objetivos del *sprint* 2. Cabe mencionar que se crearon dos ítems de tipo *bug* debido a que se presentaron problemas de mal funcionamiento del sistema, y que fueron descubiertos una vez finalizado el *sprint* 1.

Usualmente los ítems de tipo *bug* tienen alta prioridad dentro del *product backlog*, debido a que un sistema con mal funcionamiento no puede ser entregable, pero no siempre es así.

Cuando un *bug* es levantado solo por errores de tipo cosmético en la interfaz, es decir que es un problema de vista y no de funcionalidad, este podría tener una prioridad menor que cualquier otro PBI.

Tabla 2.39 *Sprint backlog* 2

Prioridad	Título	Estado	Tipo de ítem de trabajo
1	An error occurred while receiving the HTTP response in WCF service	Committed	*Bug*
3	Build "Select a Patient" page and add "Select a Patient" functionality	Committed	*Product Backlog Item*
4	Build Patient general information page	Committed	*Product Backlog Item*
5	Build Patient Allergy page	Committed	*Product Backlog Item*
6	Build Patient Vascular Access page	Committed	*Product Backlog Item*
10	Build Patient infection page	Committed	*Product Backlog Item*
11	Build Patient Immunizations page	Committed	*Product Backlog Item*

Los objetivos de este *sprint* son: solucionar los errores de funcionalidad identificados, implementar la funcionalidad de seleccionar un paciente, y finalmente implementar varias interfaces de usuario que despliegan información médica (se construirán las interfaces `Select Patient`, `Patient General Info`, `Allergy`, `Vascular Access Device`, `Infection` e `Immunization`).

2.3.4.2 Desarrollo del sprint 2

2.3.4.2.1 Error de respuesta de servicio WCF

Durante la etapa de pruebas del consumo del servicio WCF `PatientService`, apareció un tipo de excepción con el mensaje: *An error occurred while receiving the HTTP response in WCF service*. El mensaje de error no daba ninguna pista sobre lo que verdaderamente estaba ocurriendo, así que se hicieron algunos cambios en

el sistema para poder llegar a la fuente del problema y solucionarlo. Los cambios que se hicieron son los siguientes:

1. Habilitar el envío de fallas SOAP desde el servicio WCF.
2. Habilitar el manejo de fallas SOAP en la aplicación web MVC.

Según [30], cuando una petición a un servicio web es procesada, si un error ocurre, la naturaleza del error debe ser comunicada al cliente. Dado que los clientes pueden estar codificados en diferentes plataformas y en diferentes lenguajes de programación, debe existir un estándar de comunicación de errores independiente de la plataforma. La especificación SOAP define un estándar para ello. Según [31], en .NET, la comunicación de fallas SOAP se lo hace lanzando excepciones de tipo `FaultException` desde el servicio web. El primer paso fue la implementación de la clase `MyError` que servirá para encapsular la información de errores dentro de cada `FaultException`.

El **Segmento de código 2.19** muestra la clase `MyError`. Un contrato es un acuerdo formal entre un servicio y el cliente web, que abstractamente describe los datos que serán intercambiados. Los atributos `[DataContract]` y `[DataMember]` permiten definir dicho contrato, tanto a nivel de clase como a nivel de método respectivamente. En la línea 9 del **Segmento de código 2.19** se muestra el uso del atributo `[DataContract]`, en las líneas 12, 14 y 16 se muestra el uso del atributo `[DataMember]`. Entre las líneas 19 y 24 se implementa el constructor de la clase.

En la implementación de cada servicio WCF, se añadieron sentencias `try-catch` para atrapar las excepciones de .NET que se generen, y a su vez transformarlas en `FaultException` para poder enviarlas al cliente web.

El **Segmento de código 2.20** muestra cómo se realiza la transformación de excepciones de .NET a `FaultException`. En la línea 92 se implementa la sentencia `try` que intentará ejecutar todo el código contenido, y en caso de que ocurra una excepción, se ejecutará la sentencia `catch` que se muestra en la línea 98. La sentencia `catch` recibe como parámetro la excepción generada, y en la línea 99 se traslada la información del error a la clase `MyError`, luego en la línea 100 se

crea una nueva instancia de `FaultException` para finalmente lanzarla en la línea 101. Todo este procesamiento se realiza del lado del servicio WCF.

```
9.    [DataContract]
10.   public class MyError
11.   {
12.       [DataMember]
13.       public string Message { get; set; }
14.       [DataMember]
15.       public string Source { get; set; }
16.       [DataMember]
17.       public string StackTrace { get; set; }
18.
19.       public MyError(string message, string source, string stackTrace)
20.       {
21.           Message = message;
22.           Source = source;
23.           StackTrace = stackTrace;
24.       }
25.   }
```

Segmento de código 2.19 Clase generada para encapsular información de error dentro de un `FaultException`

Una vez que el servicio WCF lanza un objeto `FaultException` ante una excepción, es necesario hacer que el cliente web, es decir la aplicación MVC, pueda leerlo y procesarlo. MVC.NET permite la creación de filtros de excepciones. Esta funcionalidad permite procesar los errores que se generan cuando se ejecutan los métodos `Action` de los controladores de MVC.

El **Segmento de código 2.21** muestra la implementación del filtro de excepciones `HandleFaultExceptionAttribute`. En la línea 6 se muestra que los filtros deben heredar de `FilterAttribute` e implementar la interfaz `IExceptionFilter`, para hacerle saber a MVC.NET que se trata de un filtro de escepciones.

Se debe implementar el método `OnException` de la interfaz `IExceptionFilter` tal como se muestra en la línea 9. El método `OnException` recibe como parámetro

`filterContext`, que es un objeto de tipo `ExceptionContext`, y expone algunas propiedades como:

- `Exception (Exception)`: La excepción no manejada.
- `ExceptionHandled (bool)`: Retorna `true` solo cuando otro filtro marcó la excepción como manejada.
- `Result (ActionResult)`: El resultado del método `Action` donde se generó la excepción.

```
90.   public Allergy GetAllergyByID(int ID)
91.   {
92.       try
93.       {
94.           var business = new Business();
95.           var allergy = business.GetAllergyByID(ID);
96.
97.           return allergy;
98.       }catch (Exception ex) {
99.           MyError ErrLog = new MyError(ex.InnerException.Message,
      ex.InnerException.Source, ex.InnerException.StackTrace);
100.          var fault = new FaultException<MyError>(ErrLog, new
      FaultReason(ErrLog.Message));
101.          throw fault;
102.      }
103.  }
```

Segmento de código 2.20 Manejo de Excepciones en el método `GetAllergyByID` del servicio `PatientService`

En la línea 13 del **Segmento de código 2.21** se verifica si hubo una excepción de tipo `FaultException` y si esta no fue manejada previamente, en la línea 13 se asigna una nueva acción la cual retorna la vista `HandleError`. La vista `HandleError` permite desplegar un mensaje de error elegante al usuario. En la línea 19, se indica a MVC.NET que la excepción ya fue manejada.

Para que un controlador de MVC implemente el filtro de excepciones `HandleFaultException`, se debe añadir el atributo

`[HandleFaultException]` a la clase controlador, tal como se muestra en la línea 1 del **Segmento de código 2.22**.

```
6.  public class HandleFaultExceptionAttribute : FilterAttribute,
    IExceptionFilter
7.  {
8.
9.      public void OnException(ExceptionContext filterContext)
10.     {
11.         if (!filterContext.ExceptionHandled && filterContext.Exception
    is FaultException)
12.         {
13.             filterContext.Result = new ViewResult
14.             {
15.                 ViewName = "HandleError",
16.                 ViewData = new
    ViewDataDictionary<bool>(filterContext.IsChildAction)
17.             };
18.
19.             filterContext.ExceptionHandled = true;
20.         }
21.     }
22. }
```

Segmento de código 2.21 Creación del filtro de excepciones `HandleFaultExceptionAttribute`

```
1.      [HandleFaultException]
2.     public class PatientController : Controller
3.     {
```

Segmento de código 2.22 Controlador de MVC con filtro de excepciones `HandleFaultExceptionAttribute`

Después de implementar los cambios mencionados, la aplicación web está en capacidad de recibir y procesar las excepciones de tipo `FaultException` que el servicio WCF envía cada vez que hay una excepción. Se ejecutó la aplicación web MVC, y se encontró que existían problemas de serialización de las entidades de

negocio de lado del servicio WCF. Al momento que el servicio WCF intentaba serializar una entidad de negocio, se originaba una excepción de referencia circular.

Cuando Entity Framework genera las entidades de negocio a partir de las tablas, transforma relaciones entre tablas en propiedades autogeneradas. Por ejemplo, en la entidad `TestPatient` se autogenera la propiedad `Allergies`, que es una lista de objetos de tipo `Allergy`, mientras que en la entidad `Allergy`, se autogenera la propiedad `TestPatient` de tipo `TestPatient`, tal como se muestra en la **Figura 2.32**. Cuando se intenta serializar el objeto `Allergy`, por ejemplo, el serializador intentará serializar la propiedad `TestPatient`, pero dentro de esta existe una lista de objetos de tipo `Allergy`, por lo tanto tratará también de serializar cada objeto de tipo `Allergy`, luego nuevamente volverá a tratar de serializar el objeto `TestPatient` y así sucesivamente. Este se convierte en un proceso infinito, y la aplicación termina lanzando una excepción de referencia circular. Para solucionar este problema, se implementó la estrategia de DTO (*Data Transfer Object*), que consiste en asociar las entidades de negocio con los objetos DTO, de tal forma que los servicios web ya no responden con objetos de negocio, sino con objetos DTO.

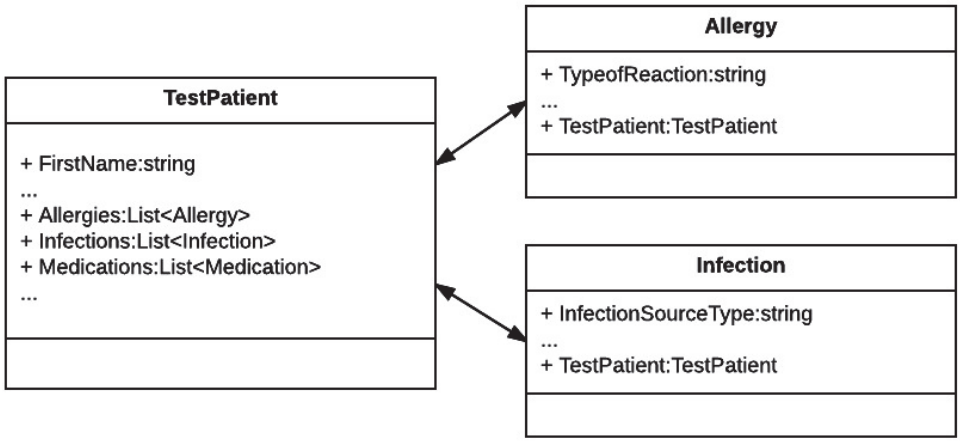


Figura 2.32 Relación entre entidades

Los objetos DTO tienen las siguientes características:

- No asocian campos de auditoría.
- No contienen las propiedades autogeneradas existentes en las entidades de negocio.

- Algunos DTO serán una versión muy reducida de los objetos de negocios originales, y serán utilizados para enviar la información a ser desplegada en los *grid* a implementarse en las interfaces que siguen el esquema *grid-detail*

El **Segmento de código 2.23** muestra el código usado para crear la clase DTO `AllergyDTO`.

```
93.    [DataContract]
94.    public class AllergyDTO
95.    {
96.        [DataMember]
97.        public int ID { get; set; }
98.        [DataMember]
99.        public Nullable<int> PatientID { get; set; }
100.       [DataMember]
101.       public Nullable<int> MedID { get; set; }
102.       [DataMember]
103.       public string MedicationName { get; set; }
104.       [DataMember]
105.       public string TypeofReaction { get; set; }
106.       [DataMember]
107.       public Nullable<System.DateTime> StartDate { get; set; }
108.       [DataMember]
109.       public Nullable<System.DateTime> EndDate { get; set; }
110.       [DataMember]
111.       public string DRUG_ID { get; set; }
112.   }
```

Segmento de código 2.23 Clase DTO `AllergyDTO`

Por cuestiones de espacio en la redacción, no se incluye los detalles de la asociación desde las entidades de negocio hacia los DTO, pero el código se encuentra en el Anexo A.

2.3.4.2.2 Funcionalidad de seleccionar un paciente

La selección del paciente se la realiza desde la vista parcial de MVC `SelectPatient`, la cual será dibujada solo en las interfaces donde se requiera implementar su funcionalidad. La vista despliega un *dropdown* con la lista de

pacientes, donde el usuario puede seleccionar uno de ellos al presionar `Submit`. La petición al servidor web para seleccionar un paciente se la realiza usando AJAX[42]. La librería de JavaScript jQuery expone métodos que permiten realizar peticiones AJAX, uno de ellos es el método `post`. El **Segmento de código 2.24** muestra la implementación de la función de JavaScript `SetCleanSessionSelectedPatient`. Este método implementa una petición al servidor web usando el método `post` de la librería jQuery. En la línea 48 se genera la URL de la petición al servidor web usando el método `Action`, que recibe como parámetros el nombre de la acción `SubmitSelectedPatient`, y el nombre del controlador que maneja la acción `UserControl`.

En la línea 49 se construye un objeto de JavaScript con los datos que se enviarán en la petición al servidor web. Entre la línea 51 y 53 se llama al método `POST` de la librería jQuery, el cual recibe como parámetros la URL de la acción hacia donde se realiza la petición, y los datos enviados en la petición. El método `done` de la línea 51 permite declarar una función anónima que se ejecutará si la petición fue satisfactoria. Una función anónima en JavaScript es aquella que no tiene un nombre definido. La línea 52 llama a la función de la librería jQuery, la cual permite enviar el contenido del atributo HTML `form` de la interfaz de usuario al servidor web. Cuando se realiza el envío, la página web es recargada en el explorador web.

El **Segmento de código 2.25** muestra la acción `SubmitSelectedPatient` que se ejecuta cuando se emula un pedido producto del método `post` de la librería jQuery. Si la opción enviada en el parámetro de entrada `option` es igual a `set`, entonces se almacenará el valor del parámetro de entrada `patient` en la variable de sesión `selectedPatient` tal como se muestra en la línea 5.

En la línea 9 del **Segmento de código 2.25** se inicializa el valor de la variable de sesión `selectedPatient` cuando el parámetro `option` tenga un valor distinto de `set`, finalmente en la línea 12 se retorna el valor `true` en formato `JSON`[43].

---

[42] AJAX (*Asynchronous JavaScript and XML*) es una técnica que permite realizar peticiones al servidor web usando código de cliente JavaScript.
[43] JSON (JavaScript *Object Notation*) es un formato usado para el intercambio de datos entre el servidor web y el explorador web.

```
41.  function SetCleanSessionSelectedPatient(selectedPatient) {
42.    var option = "clear";
43.
44.    if (selectedPatient) {
45.      option = "set";
46.    }
47.
48.    var url = "@Url.Action("SubmitSelectedPatient",
     "UserControl")";
49.    var data = { patientID: $("#ddnPatientList").val(), option:
     option };
50.
51.    $.post(url, data).done(function () {
52.      $("form").submit();
53.    });
54.  }
```

Segmento de código 2.24 Implementación de la función de JavaScript `SetCleanSessionSelectedPatient`

```
1.   public JsonResult SubmitSelectedPatient(int? patientID,
     MedicalSystem.Enum.PatientSessionOption option)
2.   {
3.       if (option == Enum.PatientSessionOption.set)
4.       {
5.           this.Session["selectedPatient"] = patientID;
6.       }
7.       else
8.       {
9.           this.Session["selectedPatient"] = null;
10.      }
11.
12.      return Json(true);
13.  }
```

Segmento de código 2.25 Implementación de la acción `SubmitSelectedPatient`

2.3.4.2.3 Construcción de las interfaces de usuario

Las interfaces de usuario fueron determinadas durante la fase de diseño. En esta sección se establecerá que controladores y vistas de MVC se requieren para la construcción de las interfaces, además de las operaciones del servicio WCF que se necesiten.

Como ya se mencionó anteriormente, una interfaz de usuario estará formada por un *layout*, una vista y opcionalmente una o más vistas parciales. El definir el número de vistas que forman una interfaz depende básicamente de la funcionalidad que se requiera.

Por ejemplo, la funcionalidad de seleccionar un paciente obligó a añadir la vista parcial `SelectPatient` a todas las interfaces que despliegan información asociada a un paciente. La **Tabla 2.41** muestra las vistas de MVC que se requieren implementar por cada interfaz de usuario.

De todas las interfaces de usuario que se construyeron durante el *sprint* 2, solo se mostrará un resumen del proceso de desarrollo de la interfaz `Allergy`, el proceso es similar para el resto de interfaces. El código de todas las interfaces se presenta en el Anexo A.

Para desplegar la interfaz de usuario `Allergy` se realizaron los siguientes pasos:

- Construcción del repositorio `AllergyRepository` con 1 método, en la capa de acceso a datos. La implementación del repositorio `AllergyRepository` se muestra en el **Segmento de código 2.26**.

```
27.  public class AllergyRepository : GenericDataRepository<Allergy>
28.  {
29.      public IList<Allergy> GetAllergiesByPatient(int patientID)
30.      {
31.          var result = GetList(t => t.PatientID == patientID);
32.          return result;
33.      }
34.  }
```

Segmento de código 2.26 Implementación del repositorio `AllergyRepository`

- Construcción de dos métodos en la capa de negocio que retornan información de alergias del paciente.

Tabla 2.40 Métodos de la capa de negocio necesarios para desplegar la interfaz `Allergy`

#	Firma del método
1	`public IList<Allergy> GetAllergiesByPatientID(int patientID)`
2	`public Allergy GetAllergyByID(int ID)`

- Construcción de las clases DTO y de los métodos necesarios para la asociación de las entidades de negocio con los objetos DTO.

El **Segmento de código 2.27** muestra la clase DTO generada para el transporte de información de alergias desde el servicio WCF hasta la aplicación web, la cual será desplegada dentro de un grid.

La estructura DTO para el *grid* `AllergyGridDTO` es mucho más pequeña que la estructura `AllergyDTO`, con el objetivo de mejorar el rendimiento del sistema, y solo presentar información resumida. Ambas clases DTO poseen el atributo `[DataContract]`, y todas sus propiedades poseen el atributo `[DataMember]` para establecer el acuerdo formal entre el servicio y el cliente. El atributo `DataMember` debe ser aplicado a cada miembro del tipo `DataContract` para indicar que es un miembro de datos y que debe ser serializado.

```
74.         [DataContract]
75.         public class AllergyGridDTO
76.         {
77.             [DataMember]
78.             public int ID { get; set; }
79.             [DataMember]
80.             public Nullable<System.DateTime> StartDate { get; set; }
81.             [DataMember]
82.             public string MedicationName { get; set; }
83.             [DataMember]
84.             public string TypeofReaction { get; set; }
85.         }
```

Segmento de código 2.27 Clase DTO `AllergyGridDTO`

Tabla 2.41 Vistas de MVC que se requieren por cada interfaz de usuario

Interfaz de usuario	Controlador	Vista	¿Es vista Parcial?	*Layout*
Login	Account	Login	NO	_Layout.cshtml
Register	Account	Register	NO	_Layout.cshtml
Home	Home	Index	NO	_LayoutMenu.cshtml
Patient General Info	Patient	General Information	NO	_LayoutMenu.cshtml
	UserControl	SelectPatient	SI	_LayoutMenu.cshtml
Allergy	Patient	Allergy	NO	_LayoutMenu.cshtml
	Patient	AllergyDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Vascular Access	Patient	VascularAccess	NO	_LayoutMenu.cshtml
	Patient	VascularAccessDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Infection	Patient	Infection	NO	_LayoutMenu.cshtml
	Patient	InfectionDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Immunization	Patient	Immunization	NO	_LayoutMenu.cshtml
	Patient	ImmunizationDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Treatments	Patient	Treatment	NO	_LayoutMenu.cshtml
	Patient	TreatmentDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Medications	Patient	Medication	NO	_LayoutMenu.cshtml
	Patient	MedicationDetail	SI	_LayoutMenu.cshtml
	UserControl	Select Patient	SI	_LayoutMenu.cshtml
Manage Users	User	UserHeader	NO	_LayoutMenu.cshtml
	User	UserDetail	SI	_LayoutMenu.cshtml
	User	PhysicianDetail	SI	_LayoutMenu.cshtml
Manage Physicians	User	UserHeader		_LayoutMenu.cshtml
	User	PhysicianDetail		_LayoutMenu.cshtml

Interfaz de usuario	Controlador	Vista	¿Es vista Parcial?	*Layout*
Manage Roles	User	UserHeader		_LayoutMenu.cshtml
	User	ManageRole		_LayoutMenu.cshtml
Reports	Patient	Report		_LayoutMenu.cshtml

El **Segmento de código 2.28** muestra la asociación de una entidad de negocio `Allergy` con una entidad DTO `AllergyGridDTO`. En la línea 48 se valida que la entidad `Allergy` sea distinta de nulo, caso contrario podría arrojar una excepción. Luego en las siguientes líneas se instancia un nuevo objeto DTO y se cargan las propiedades en base a los datos de la entidad origen.

```
44.     public AllergyGridDTO ToAllergyGridDTO(Allergy allergy)
45.     {
46.         AllergyGridDTO result = null;
47.
48.         if (allergy != null)
49.         {
50.             result = new AllergyGridDTO();
51.
52.             result.ID = allergy.ID;
53.             result.StartDate = allergy.StartDate;
54.             result.EndDate = allergy.EndDate;
55.             result.MedicationName = allergy.MedicationName;
56.             result.TypeofReaction = allergy.TypeofReaction;
57.         }
58.
59.         return result;
60.     }
```

Segmento de código 2.28 Método `ToAllergyGridDTO` que permite asociar un objeto de tipo `Allergy` con un objeto de tipo `AllergyGridDTO`

Implementar DTO para cada una de las entidades de negocio es un trabajo extenso, sobre todo si se tiene demasiadas entidades de negocio que asociar a cada DTO, es por ello por lo que las empresas suelen optar por usar librerías de terceros que hagan la asociación automática, reduciendo considerablemente el trabajo de desarrollo.

Además, para desplegar la interfaz de usuario `Allergy`, se realizaron los siguientes pasos:

- Construcción de dos operaciones del servicio `PatientService` para exponer información de alergias del paciente. Las operaciones incluyen la interfaz, el contrato de servicio y el manejo de excepciones. La **Tabla 2.42** muestra ambas operaciones.

Tabla 2.42 Operaciones que expone el servicio `PatientService` necesarias para la implementación de la interfaz `Allergy`

#	Operaciones del servicio
1	`public IList<AllergyGridDTO> GetAllergiesGridByPatientID(int patientID)`
2	`public AllergyDTO GetAllergyByID(int ID)`

- Construcción de 2 vistas MVC en la capa de presentación: `Allergy.cshtml` y `AllergyDetail.cshtml`, así como las dos acciones que retornan ambas vistas de MVC. La vista `Allergy.cshtml` implementa un control de tipo *grid* para el despliegue de un resumen de las alergias del paciente, además tiene referencia a la acción `SelectPatient`. El usuario podrá seleccionar una de las alergias del *grid*, y cuando eso sucede, se cargará la vista `AllergyDetail.cshtml` debajo del *grid*, donde se podrá visualizar el detalle de la alergia seleccionada. Esta misma funcionalidad se aplicará a diferentes páginas que despliegan información médica de pacientes.

El **Segmento de código 2.29** muestra la implementación de la acción `Allergy`, mientras que el **Segmento de código 2.30** muestra la implementación de la acción `AllergyDetail`.

En **Segmento de código 2.31** muestra el código de la vista MVC `AllergyDetail`. En la línea 125 se añade una referencia a `PatientService` para tener acceso a la clase `AllergyDTO`. En la línea 126 se define el modelo, en la línea 128 se especifica que no existe *layout* debido a que esta es una vista parcial. A partir de la línea 130, hasta la línea 158, se muestra código en lenguaje HTML para definir la estructura de la interfaz de usuario, mezclado con líneas de código de servidor,

como las líneas 134, 136, etc. En ellas se llama a *helpers* HTML, cuya función es devolver contenido HTML hacia el explorador web. Por ejemplo, la línea 137 es una línea de código que generará una etiqueta HTML que a la vista del usuario será una caja de texto. La etiqueta HTML generada tendrá el atributo `name` con el valor `MedicationName`, y el atributo `class` cuyo valor será `form-control`.

```
61.         public ActionResult Allergy()
62.         {
63.             var result = new List<AllergyGridDTO>();
64.             var selectedPatientInSession = GetSelectedPatientInSession();
65.             ViewBag.SessionSelectedPatient = selectedPatientInSession;
66.
67.             if (selectedPatientInSession > 0)
68.             {
69.                 using (var patientServiceClient = new
            PatientService.PatientServiceClient())
70.                 {
71.                     result =
            patientServiceClient.GetAllergiesGridByPatientID(selectedPatientInSession);
72.                 }
73.             }
74.
75.             return View(result);
76.         }
```

Segmento de código 2.29 Acción que devuelve la vista `Allergy`

```
1.          public ActionResult AllergyDetail(int recordID)
2.          {
3.              AllergyDTO result = new AllergyDTO();
4.              using (var patientServiceClient = new
            PatientService.PatientServiceClient())
5.              {
6.                  result = patientServiceClient.GetAllergyByID(recordID);
7.              }
8.
9.              return View(result);
10.         }
```

Segmento de código 2.30 Acción que devuelve la vista `AllergyDetail`

2.3.5 SPRINT 3

2.3.5.1 Sprint planning

La **Tabla 2.43** muestra los ítems escogidos para cumplir con los objetivos del *sprint* 3, los cuales son: Construir las interfaces de usuario `Medication` y `Treatment`, crear capa de autenticación y autorización, implementar autorización con base en los roles de usuario, construir las interfaces para el manejo de usuarios e implementar la seguridad de los servicios WCF. El código de las interfaces `Medication` y `Treatment` no se incluyeron, ya que es similar al código de la interfaz de usuario `Allergy`, pero este puede ser encontrado en el ANEXO A.

Tabla 2.43 *Sprint backlog* 3

Prioridad	Título	Estado	Tipo de ítem de trabajo
1	*Build Patient medications page*	*Committed*	*Product Backlog Item*
2	*Build treatments page*	*Committed*	*Product Backlog Item*
3	*Move ASP.NET Identity out from Web Application into a new Library*	*Committed*	*Product Backlog Item*
4	*Implement Authorization in MVC Web Application*	*Committed*	*Product Backlog Item*
5	*Build "Users" page*	*Committed*	*Product Backlog Item*
6	*Build Physicians page*	*Committed*	*Product Backlog Item*
7	*Research about securing WCF services*	*Committed*	*Product Backlog Item*
8	*Implement security token in WCF services*	*Committed*	*Product Backlog Item*
9	*Implement authentication in WCF Services side*	*Committed*	*Product Backlog Item*

2.3.5.2 Desarrollo del sprint 3

2.3.5.2.1 Creación de capa de autenticación y autorización

Cuando se creó la aplicación web de MVC desde Visual Studio durante el *sprint* 1, se seleccionó la opción de incluir autenticación de usuarios, para facilitar el proceso de desarrollo. En este punto, la aplicación web MVC cuenta con funcionalidad de autorización y autenticación que provee el *framework* de ASP.NET Identity, pero dicha funcionalidad está embebida dentro de la aplicación web.

```
@using MedicalSystem.PatientService;
@model AllergyDTO
@{
    Layout = null;
}
<br />
<div class="row">
    <div class="col-lg-12">
        <div class="form-group">
            @Html.LabelFor(t => t.MedicationName, "Medication Name", new { @class = "col-sm-3 control-label" })
            <div class="col-sm-9">
                @Html.TextBoxFor(t => t.MedicationName, new { @class = "form-control" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(t => t.TypeofReaction, "Type of Reaction", new { @class = "col-sm-3 control-label" })
            <div class="col-sm-9">
                @Html.TextBoxFor(t => t.TypeofReaction, new { @class = "form-control" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(t => t.StartDate, "Start Date", new { @class = "col-sm-3 control-label" })
            <div class="col-sm-9">
                @Html.TextBoxFor(t => t.StartDate, new { @class = "form-control" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(t => t.EndDate, "End Date", new { @class = "col-sm-3 control-label" })
            <div class="col-sm-9">
                @Html.TextBoxFor(t => t.EndDate, new { @class = "form-control" })
            </div>
        </div>
    </div>
</div>
```

Segmento de código 2.31 Vista de MVC `AllergyDetail`

Algunos de los elementos de ASP.NET Identity que se incluyeron dentro de la aplicación web MVC son:

- La implementación de una base de datos para manejo de información de autenticación y autorización usando el enfoque *code first*. La persistencia de datos hacia la base de datos de autenticación y autorización es manejada por el `UserStore` y el `RoleStore`.
- La clase `UserStore`, que es parte del ensamblado[44] `Microsoft.AspNet.Identity.EntityFramework`, y que es una implementación de Entity Framework para el almacenamiento de información relacionada al usuario.
- La clase `RoleStore`, que es parte del ensamblado `Microsoft.AspNet.Identity.EntityFramework` y que es una implementación de Entity Framework para el almacenamiento de información relacionada a los roles de usuario.
- La clase `UserManager`, que expone funcionalidad asociada al usuario, y que guarda los cambios usando la clase `UserStore`. Esta clase pertenece al ensamblado `Microsoft.AspNet.Identity.Core`.
- La clase `RoleManager`, que expone funcionalidad asociada al usuario, y que guarda los cambios usando la clase `RoleStore`. Esta clase pertenece al ensamblado `Microsoft.AspNet.Identity.Core`.
- El acceso a datos de la base de datos identidad es manejada por el `UserStore` y el `RoleStore`.

Los elementos antes mencionados son clave dentro de la funcionalidad de autenticación y autorización de la aplicación web de MVC. La **Figura 2.33** muestra algunos de esos componentes.

Es necesario separar la funcionalidad de acceso a datos de la funcionalidad de autenticación y autorización en un proyecto separado para conformar la capa de autenticación y autorización, para que de esa manera pueda ser reutilizada tanto por la aplicación web MVC como por el servicio WCF y así implementar autenticación de ese lado del sistema siguiendo el principio DRY.

---

[44] Ensamblado o *assembly* es un archivo o varios archivos que contienen el código y los recursos de una aplicación. Un ensamblado es reutilizable y versionable.

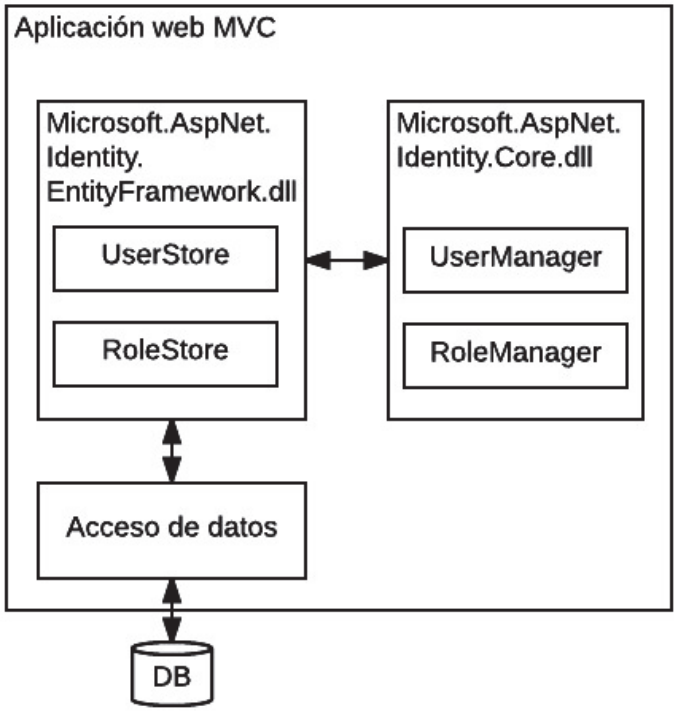


Figura 2.33 Elementos de ASP.NET Identity incluidos en la aplicación web MVC

La **Figura 2.34** muestra un diagrama de cómo queda la aplicación web después de desacoplar el acceso a datos de la funcionalidad de autenticación y autorización fuera de la aplicación web MVC.

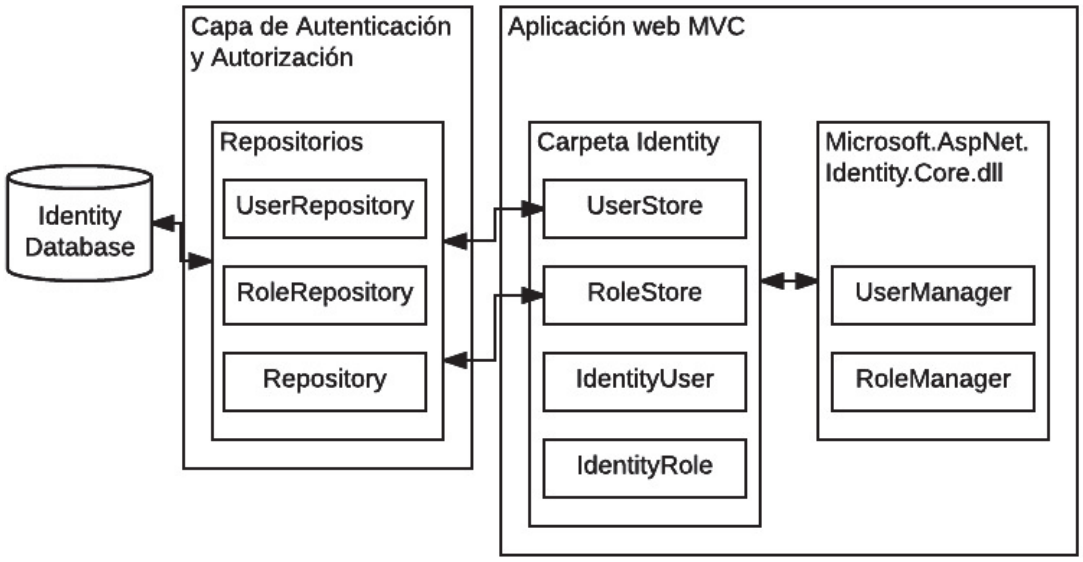


Figura 2.34 Interacción de aplicación web MVC con capa de autenticación y autorización

Las clases `UserStore` y `RoleStore` fueron construidas con una funcionalidad similar a la que tenían anteriormente, con la diferencia que se conectarán a una nueva capa de acceso a datos que se encuentra dentro del componente de autenticación y autorización. Las clases `IdenditeyUser` y `IdentityRole` son estructuras que permitirán instanciar objetos de tipo `IdenditeyUser` y `IdentityRole` respectivamente de lado de la aplicación web MVC, pero una vez que pasen al componente de autenticación y autorización, serán asociados a las entidades `User` y `Role` respectivamente.

Para crear la capa de autenticación y autorización se siguieron los siguientes pasos:

1. Crear un nuevo proyecto de librería de clases de Visual Studio llamado `MedicalSystem.EntityFramework` en una solución nueva.
2. Añadir las clases `User` y `Role` dentro de la carpeta `Entities`. La estructura de las clases deben tener relación con la estructura de las tablas `dbo.User` y `dbo.Role` de la base de datos Identidad. En el **Segmento de código 2.32** se muestra la implementación de la clase `Role`.

```
3.  public class Role
4.  {
5.      #region Fields
6.      private ICollection<User> _users;
7.      #endregion
8.
9.      #region Scalar Properties
10.     public Guid RoleId { get; set; }
11.     public string Name { get; set; }
12.     #endregion
13.
14.     #region Navigation Properties
15.     public ICollection<User> Users
16.     {
17.         get { return _users ?? (_users = new List<User>()); }
18.         set { _users = value; }
19.     }
20.     #endregion
21. }
```

Segmento de código 2.32 Clase `Role` de componente de autorización y autenticación

En la línea 15 del **Segmento de código 2.32** se muestra que, para un rol de usuario, puede haber varios usuarios relacionados. En la clase `User` en cambio, existirá la propiedad `public ICollection<User> Users`, ya que un usuario podrá tener varios roles de usuario. Esta es la manera en que Entity Framework asocia la relación de muchos a muchos entre las entidades `User` y `Role`, mientras que en la base de datos Identidad, la tabla intermedia `dbo.UserRole` permite crear la relación entre las tablas `dbo.User` y `dbo.Role`.

3. Añadir las configuraciones `EntityTypeConfiguration` que Entity Framework requiere para asociar las entidades `User` y `Role` con las tablas `dbo.User`, `dbo.Role` y `dbo.UserRole` de la base de datos Identidad. El `EntityTypeConfiguration` es análogo a la subcapa de asociación de la subcapa de modelado de datos de ADO.NET, y es requerido cuando se usa el enfoque *code first*.
4. Implementar el contexto de base de datos.
5. Implementar la cadena de conexión a la base de datos. Una cadena de conexión especifica una fuente de información, en este caso la fuente del componente de autenticación y autorización es la base de datos Identidad. El **Segmento de código 2.33** muestra la cadena de conexión empleada para conectar el componente con la base de datos Identidad.

```
7.     <connectionStrings>
8.       <add name="IdentityConnection"
    connectionString="Server=tcp:medicaldatabase.database.windows.net,1433;I
    nitial Catalog=IdentityDatabase;Persist Security Info=False;User
    ID=yyyyyyyyyyy;Password=xxxxxxxxxxxx;MultipleActiveResultSets=False;En
    crypt=True;TrustServerCertificate=False;Connection Timeout=30;"
    providerName="System.Data.SqlClient" />
9.     </connectionStrings>
```

Segmento de código 2.33 Cadena de conexión `IdentityConnection`

El **Segmento de código 2.33** muestra que el nombre del servidor de base de datos es `medicaldatabase.database.windows.net` y que el nombre de la base de datos es `IdentityDatabase`, además incluye un ID de usuario y su clave. La

base de datos debe tener permisos asignados a ese usuario para que pueda realizar operaciones sobre la misma.

6. Implementar los repositorios `Repository`, `UserRepository` y `RoleRepository`. El código de los repositorios puede ser encontrados en el ANEXO A.

7. Implementar el patrón de diseño *unit of work* (unidad de trabajo). Una unidad de trabajo permite realizar varias operaciones sobre la base de datos como si fuera una sola transacción.

Para implementar el patrón de diseño *unit of* work se requiere crear la clase `UnitOfWork` que contiene a los repositorios `UserRepository` y `RoleRepository`. Los controladores de la aplicación web MVC no consumen directamente los repositorios implementados, en vez de eso, instancian un nuevo objeto de tipo `UnitOfWork`, el cual recibe como parámetro el nombre de la cadena de conexión, y de ahí acceden a los repositorios. La **Figura 2.35** muestra como un controlador de MVC accede a la base de datos Identidad a través del objeto `UnitOfWork`.

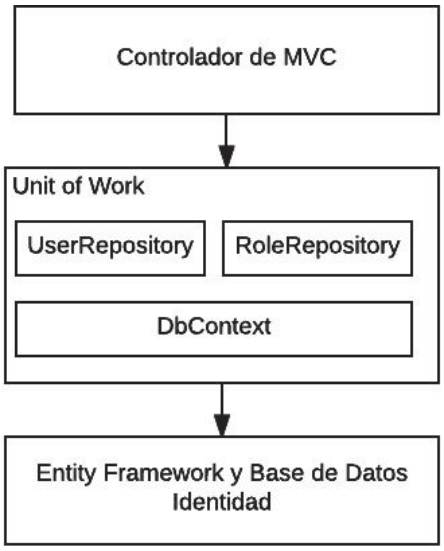


Figura 2.35 Aplicación consume objeto `UnitOfWork` para acceder a los repositorios

Una vez implementados los pasos anteriores, es necesario añadir la referencia de esta componente en el proyecto de aplicación web de MVC. El **Segmento de**

código 2.34 línea 27 muestra el constructor de la clase `UserController`. Cada vez que se instancia un controlador, se inyecta el objeto `UserManager`. Por otro lado, si se requiere instanciar un objeto de tipo `UserStore`, este requiere de una nueva instancia de un objeto de tipo `UnitOfWork`, tal como se muestra en la línea 29 del **Segmento de código 2.34**.

2.3.5.2.2 Implementación de autorización mediante roles de usuario

Para implementar autenticación mediante roles, primero se insertó en la base de datos `IdentityDatabase` los roles ya definidos en la fase de diseño. El **Segmento de código 2.35** muestra el *query* ejecutado para la inserción de registros de roles de usuario en la tabla `dbo.Role`. La línea 1 inserta el role `Default`, La línea 2 inserta el role `PhysicianUnauthorized`, la línea 3 inserta el role `Physician` y la línea 4 inserta el role `Admin`.

```
22.  #region Constructor and Members
23.
24.  private readonly UserStore _userStore;
25.  private readonly UserManager<IdentityUser, Guid> _userManager;
26.
27.  public UserController(UserManager<IdentityUser, Guid> userManager)
28.  {
29.      _userStore = new UserStore(new UnitOfWork("IdentityConnection"));
30.      _userManager = userManager;
31.  }
32.
33.  #endregion
```

Segmento de código 2.34 Constructor y miembros del controlador `UserController` de la aplicación web de MVC

```
1. INSERT INTO [dbo].[Role] ([RoleId], [Name]) VALUES (NEWID(), 'Default')
2. INSERT INTO [dbo].[Role] ([RoleId], [Name]) VALUES (NEWID(),
   'PhysicianUnauthorized')
3. INSERT INTO [dbo].[Role] ([RoleId], [Name]) VALUES (NEWID(), 'Physician')
4. INSERT INTO [dbo].[Role] ([RoleId], [Name]) VALUES (NEWID(), 'Admin')
```

Segmento de código 2.35 *Query* para inserción de roles de usuario en tabla `dbo.Role`

La aplicación web MVC usa autenticación basada en *cookie*[45]. Cuando un usuario intenta iniciar sesión en la aplicación web de MVC, ASP.NET Identity valida las credenciales ingresadas en la interfaz de usuario, y si son válidas, ASP.NET Identity obtiene la información de seguridad del usuario (que incluye la información de roles de usuario) a través del `UserManager`, y después de cierto proceso, es almacenado en una *cookie* del lado del cliente. Cada vez que el usuario realiza una petición HTTP al servidor web, la *cookie* viaja con esa petición, y luego de ser procesada, el servidor es capaz de leerla. La *cookie* contiene información de seguridad del usuario. Para más información del funcionamiento de la autenticación basada en *cookie*, revisar [32].

MVC.NET permite implementar filtros de autorización a nivel de controlador y de acción. Este filtro permite la implementación basada en roles dentro de la aplicación web a través de atributos. Cuando una clase controlador posee el atributo `[Authorize]`, se implementa autorización a nivel de controlador, por tanto, para acceder a cualquiera de las acciones dentro de ese controlador, se requiere que el usuario haya iniciado sesión, independientemente del rol que este tenga. Si se requiere permitir el acceso a una de las acciones dentro del controlador sin necesidad de que un usuario inicie sesión, entonces se puede aplicar el atributo `[AllowAnonymous]` a nivel de método.

El **Segmento de código 2.36** muestra un ejemplo de atributo a nivel de clase en la línea 16 y de método en la línea 19. Todas las acciones del controlador `AccountController` implementan autorización, excepto las acciones `Login` y `Register`, las cuales permitirán acceso anónimo.

El controlador `PatientController` devuelve las vistas de MVC que despliegan información médica, solo los usuarios que posean el rol `Admin` o `Physician` tendrán acceso a ellas. Cuando se requiere implementar autorización basada en roles `Role1` o `Role2`, se añade la etiqueta `[Authorize(Roles="Role1,Role2,…")]` ya sea a nivel de acción o de

---

[45] Cookie es una pieza pequeña de información que es generada por el servidor web y enviada al explorador web. El explorador web almacena la *cookie* en el computador desde donde se accede el sitio web.

controlador. Basta que el usuario tenga uno de los roles, para que este esté autorizado a acceder a la acción donde se aplique la etiqueta.

Un ejemplo de implementación se muestra en el **Segmento de código 2.37**, donde para acceder a cualquiera de las acciones del controlador, es necesario que un usuario haya iniciado sesión y que dicho usuario tenga el rol `Physician` o el rol `Admin`.

```
16.  [Authorize]
17.  public class AccountController : Controller
18.  {
19.      [AllowAnonymous]
20.      public ActionResult Login(string returnUrl)
21.      {
22.          ViewBag.ReturnUrl = returnUrl;
23.          return View();
24.      }
25.  }
```

Segmento de código 2.36 Atributo `Authorize` y `AllowAnonymous` en controlador `Account`

```
11.   [Authorize(Roles="Physician,Admin")]
12.  [HandleFaultException]
13.  public class PatientController : Controller
14.  {
```

Segmento de código 2.37 Atributo `Authorize` sobre clase controlador `PatientController`

Si se requiere implementar autorización `Role1` y `Role2` (usuario debe tener ambos roles), entonces se debe implementar los atributos `[Authorize(Roles="Role1")]` y `[Authorize(Roles="Role2")`, ya sea sobre la acción o sobre el controlador.

Internamente, ASP.NET Identity lee la información de seguridad del usuario que se guarda en la cookie de autenticación para obtener los roles del usuario y así implementar la autorización a través del atributo `Authorization`.

2.3.5.2.3 Desarrollo de interfaces para el manejo de usuarios

Existen 4 tipos de usuario: `Default`, `PhysicianUnauthorized`, `Physician` y `Admin`. Los usuarios de tipo `Default` y `Admin` son almacenados usando la tabla `dbo.User` de la base de datos Identidad. Los usuarios `PhysicianUnauthorized` y `Physician`, al ser usuarios médicos, su registro está dividido en dos tablas: `dbo.User` de la base de datos Identidad, y la tabla `dbo.TestPhysician` de la base de datos médica.

El almacenamiento de los datos para los usuarios médicos no es problema de implementar, ya que desde el controlador de MVC se hacen las llamadas a los métodos correspondientes para que hagan la inserción, modificación, lectura o borrado del registro de cada base de datos, es decir, si se requiere hacer la operación sobre la tabla `dbo.TestPhysician` de la base de datos médica, se llamará a una operación del servicio `PhysicianService`, mientras que si se requiere acceder a la tabla `dbo.User` de la base identidad, se lo podrá hacer llamando a un método del `UserStore`.

En la interfaz de usuario `Manage Users` se debe incluir la funcionalidad de desplegar el detalle del usuario de acuerdo al tipo de rol que tiene, para ello se construyeron 2 vistas, una para mostrar el detalle de un usuario médico, y otra para mostrar el detalle de los usuarios que no son médicos. La **Tabla 2.44** muestra las vistas implementadas para implementar la interfaz de usuario `Manage Users`.

Dentro de la clase controlador `UserController`, se implementó la lógica para desplegar la vista de detalle adecuada en la interfaz de usuario `Manage Users`. La **Figura 2.36** muestra que vistas de MVC conforman cada interfaz de usuario según el tipo de usuario.

Finalmente, existen dos interfaces de usuario que permiten administrar usuarios del sistema, estas son: `Manage Users` y `Manage Physicians`. Ambas siguen la estructura de la **Figura 2.36**, sin embargo, la lista de usuarios que se presenta en el *grid* de la interfaz de usuarios `ManagePhysicians` en la vista `UserHeader` está filtrada para mostrar solo los usuarios que sean `Physician` o `PhysicianUnauthorized`.

Cabe mencionar que ambas interfaces de usuario siguen el esquema *grid-detail* definido en la fase de diseño. El control tipo *grid* de las interfaces que siguen el esquema *grid-detail* fue implementado usando el *plugin* jsGrid [33]. En el Anexo A se incluye el código del *grid*.

El **Segmento de código 2.38** muestra la implementación de la acción `UserDetail` del controlador `UserController`, aquí es donde se implementa la lógica que decide cual vista de detalle mostrar en la interfaz de usuario `Manage Users` con base en el rol del usuario.

En la línea 54 se muestra el parámetro de entrada `recordID` que corresponde al ID del usuario seleccionado en el *grid* de usuarios de la interfaz de usuario `Manage Users`.

En la línea 59 se obtiene los roles del usuario seleccionado, y en base a esa lista de roles, en la línea 60 se verifica si el usuario tiene al menos uno de los dos roles `Physician` o `PhysicianUnauthorized`.

Finalmente, en la línea 62, si se cumple la condición de que el usuario tiene uno de los roles mencionados, entonces se devuelve la vista `PhysicianDetail`, caso contrario se devolverá la vista `UserDetail`, tal como se muestra en la línea 67.

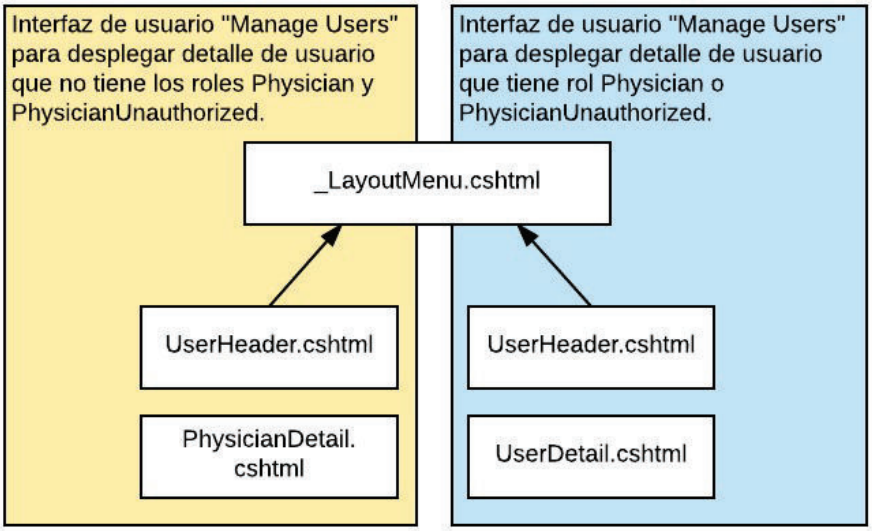


Figura 2.36 Estructura de interfaz de usuario `Manage Users` según el rol de usuario

Tabla 2.44 Vistas de MVC del controlador `UserController`

Vista MVC	Descripción
`UserHeader.cshtml`	Desplegará un *grid* con un número mínimo de columnas que desplegarán información del usuario. Desde el *grid* se podrá seleccionar la opción de desplegar el detalle de un usuario en la parte inferior.
`UserDetail.cshtml`	Desplegará el detalle del usuario seleccionado en caso de que el usuario no tenga el rol `Physician` y `PhysicianUnauthorized`.
`PhysicianDetail.cshtml`	Desplegará el detalle de un usuario que tenga el rol `Physician` o `PhysicianUnauthorized`.

```
54.  public ActionResult UserDetail(string recordID)
55.  {
56.      var identityUser = new IdentityUser();
57.
58.      identityUser = _userManager.FindByIdAsync(new Guid(recordID)).Result;
59.      var userRoles = _userStore.GetRolesAsync(identityUser).Result;
60.      var isUserPhysician = userRoles.Contains("Physician") ||
     userRoles.Contains("PhysicianUnauthorized");
61.
62.      if (isUserPhysician)
63.      {
64.          return RedirectToAction("PhysicianDetail", "User", new{ userID =
     recordID, physicianID = identityUser.TestPhysicianID });
65.      }
66.
67.      return View(identityUser);
68.  }
```

Segmento de código 2.38 Acción `GET UserDetail` del controlador `UserController`

2.3.5.2.4 Implementación de seguridad de servicios WCF

En esta sección se presentará un resumen de la implementación de seguridad de los servicios web WCF. Los servicios WCF `PatientService` y `PhysicianService` exponen información médica, la cual no debe ser accedida por un cliente web no autorizado.

Típicamente, la autenticación se refiere solo a la verificación de quien quiere acceder a un recurso, un recurso como un servicio WCF. En un servicio WCF, la autenticación es de dos vías, es decir, el servicio debe autenticar al cliente web, pero también debe verificar del lado del cliente web, para que solo pueda conectarse el servicio web que tenga acceso, y así evitar diferentes tipos de ataques informáticos. La **Tabla 2.45** muestra varios mecanismos de autenticación que WCF ofrece.

Tabla 2.45 Mecanismos de autenticación soportados en WCF

#	Mecanismo	Descripción
1	*No authentication*	Es un mecanismo en el cual el servicio no autentica a los clientes que lo llaman.
2	*Windows authentication*	Es un mecanismo en el cual el servicio autentica a los clientes en base a credenciales de Windows.
3	*Username and password*	Es un mecanismo en el cual el cliente web que llama al servicio provee de credenciales conformadas por un nombre de usuario y una contraseña. El servicio web usa esas credenciales y las compara contra otras para verificar si tiene acceso o no.
4	*X509 certificate*	Es un mecanismo en el cual el cliente web se identifica a sí mismo usando un certificado. Típicamente, el certificado es conocido por el servicio web con anterioridad.
5	*Custom mechanism*	Es un mecanismo de seguridad personalizado tomando como base los mecanismos existentes de WCF.

Un cliente web se comunica con un servicio web WCF a través de mensajes, y previo a la autenticación del servicio web al cliente, es necesario garantizar que los mensajes viajen seguros a su destino. La transferencia de mensajes entre el cliente web y el servicio web debe ser segura, caso contrario la autenticación del cliente web puede ser cuestionable.

Existen 3 aspectos esenciales de seguridad en la transferencia de mensajes entre el servicio web y el cliente web, estos son:

- Integridad del mensaje: El mensaje no debe ser alterado. Un intermediario podría interceptar el mensaje y modificar su contenido, para luego reenviarlo hacia su destino.

- Privacidad del mensaje: El mensaje solo puede ser visto por quien esté autorizado para hacerlo.
- Autenticación mutua: No solo se debe asegurar que el servicio web autentique al cliente web que lo llama, sino que también se debe verificar que el cliente se conecte al servicio correcto.

Una vez que las credenciales contenidas en un mensaje lleguen al servicio WCF, este debe autenticarlas localmente. WCF soporta cinco modos de transporte seguro para cumplir con los 3 aspectos de seguridad. La **Tabla 2.46** muestra 5 modos de transporte seguro que WCF soporta.

Tabla 2.46 Modos de transferencia segura soportados por WCF

#	Modo de transporte seguro	Descripción
1	*None*	Modo de seguridad apagado. La transferencia de mensajes está totalmente abierta a cualquier ataque malicioso.
2	*Transport security*	Permite usar un protocolo seguro de comunicación como HTTPS. Este modo de seguridad cifra toda la comunicación del canal, permitiendo asegurar la integridad y la privacidad de los mensajes. Soporta autenticación mutua. Cuando existen muchos intermediarios entre el cliente web y el servicio web, la seguridad se vuelve cuestionable, debido a que uno de esos intermediarios puede ser malicioso.
3	*Message security*	Permite aplicar cifrado a nivel del mensaje. Como se cifran los mensajes, permite asegurar la integridad y privacidad de los mensajes, y además la autenticación mutua. El cifrado a nivel de mensajes permite comunicar al servicio web con el cliente web de una manera segura sobre transportes no seguros como HTTP. Este modo de seguridad provee seguridad de extremo a extremo en la comunicación de mensajes. Un aspecto negativo es que los mensajes se vuelven pesados al tener sobrecarga.
4	*Mixed*	Permite combinar algunas de las ventajas tanto del modo de seguridad *Transport security*, como del *Message security*.
5	*Both*	Usa ambos modos de seguridad para maximizar la seguridad en la comunicación, sin embargo, mata el rendimiento de la misma.

La aplicación web y los servicios WCF del sistema fueron publicados en la nube, usando los servicios de Microsoft Azure, es decir que para la comunicación entre el cliente web y el servicio se usará el ambiente de Internet, el cual no podrá ser controlado. Al ser un ambiente inseguro, el modo de transporte seguro adecuado para proveer seguridad en la comunicación de mensajes es *Message security*.

WCF permite la configuración de la comunicación de los servicios web con los clientes usando el archivo de configuración `web.config`. La configuración relacionada al transporte, cifrado y protocolo en la comunicación entre los clientes y el servicio es realizada a través del *binding*. Existen diferentes tipos de `binding`, los cuales se muestran en la **Tabla 2.47**.

Los `binding` `NetTcpBinding`, `NetNamedPipeBinding` y `NetMsmqBinding` fueron diseñados para ser implementados en ambientes de intranet, y usan el modo de transporte seguro *Transport security* por defecto. Los *binding* `BasicHttpBinding` y `WSHttpBinding` fueron diseñados para ser implementados en un ambiente web, y soportan el modo de transporte seguro *Message security*.

Tabla 2.47 Modo de transferencia segura soportado según el tipo de *binding*

Binding	**Modo de transporte seguro**				
	None	***Transport***	***Message***	***Mixed***	***Both***
`BasicHttpBinding`	Si (defecto)	Si	Si	Si	No
`NetTcpBinding`	Si	Si (por defecto)	Si	Si	No
`NetNamedPipeBinding`	Si	Si (por defecto)	No	No	No
`WSHttpBinding`	Si	Si	Si (por defecto)	Si	No
`NetMsmqBinding`	Si	Si (por defecto)	Si	No	Si

Un cliente web que se conecta a un servicio WCF, requiere tener credenciales que lo identifiquen. Usualmente un cliente web se identifica a sí mismo usando un

nombre de usuario y una contraseña. En el modo de transporte seguro *message security*, WCF soporta los siguientes tipos de credenciales:

- *None*: No se envían credenciales, el cliente web se conecta de forma anónima al servicio web.
- *Windows*: Permite el intercambio de mensajes SOAP bajo una seguridad basada en credenciales de Windows.
- *Username*: Cliente web provee un nombre de usuario y una contraseña para que el servicio web las valide.
- *Certificate*: El servicio web realiza la autenticación del cliente usando un certificado X.509.
- *Otros*: Existen otros tipos de credenciales [34].

Los `binding` de Internet `BasicHttpBinding` y `WSHttpBinding` soportan las credenciales que se muestran en la **Tabla 2.48**.

Tabla 2.48 Tipos de credenciales aplicables a servicios y clientes web que usan modo de transporte seguro *Message Security*

Binding	**Tipo de Credencial**			
	None	***Windows***	***UserName***	***Certificate***
`BasicHttpBinding`	No	No	No	Si
`WSHttpBinding`	Si	Si (por defecto)	Si	Si

En un escenario de Internet, es posible que los clientes web no usen Windows, por lo que no es aconsejable usar credenciales de tipo Windows, sino más bien usar credenciales personalizadas que puedan ser obtenidas por ejemplo desde una base de datos, ese es el caso de la credencial de tipo *Username*.

Dado el análisis de la seguridad en la comunicación entre el servicio y el cliente web, se llega a la conclusión que el modo de transporte seguro adecuado es *Message Security* y que el tipo de credencial más conveniente es *Username*. Por tanto, los servicios WCF se configuraron para que usen el *binding* `WSHttpBinding`, ya que es el único que soporta las configuraciones previamente mencionadas.

El **Segmento de código 2.39** muestra la configuración del *binding* `MessageAndUserName` implementada en los proyectos de servicios WCF. En la línea 40 se muestra la etiqueta `wsHttpBinding`, que indica el tipo de *binding* y dentro de ella la configuración del *binding* `MessageAndUserName`.

En la línea 42 se indica que el *binding* está configurado con el modo de transporte seguro *Message Security*. Finalmente, la línea 43 muestra el tipo de credencial configurado el cual es `UserName`.

```
40.    <wsHttpBinding>
41.      <binding name="MessageAndUserName">
42.        <security mode="Message">
43.          <message clientCredentialType="UserName"/>
44.        </security>
45.      </binding>
46.    </wsHttpBinding>
```

Segmento de código 2.39 Configuración de *binding* `MessageAndUserName`

Para cifrar los datos contenidos en los mensajes de comunicación entre el servicio web y el cliente web, WCF usa un certificado X509, el cual permite autenticar el servicio al cliente web.

Un certificado funciona usando 2 llaves, una pública y una privada, así como también usa un nombre común (CN). La idea de las llaves es que la información se cifra usando la llave pública, y solo puede ser descifrada usando la respectiva llave privada.

El certificado contiene la llave pública y el nombre común, mientras que la llave privada es almacenada en un lugar seguro de la máquina donde reside. Cualquier cliente web puede obtener la llave pública.

El proceso de seguridad en el envío de mensajes desde el cliente web hacia el servicio web funciona de la siguiente manera: El cliente web se conecta al servicio web a través del `endpoint`. El cliente obtiene la llave pública que el servicio WCF le otorga. El cliente web cifra los mensajes usando la llave pública.

Los mensajes no pueden ser vistos ni modificados por terceros, porque no tienen la llave privada. Los mensajes llegan al servicio, y los descifra ya que este tiene la llave privada. El servicio WCF usa las credenciales descifradas del cliente, y autentica al cliente web.

Se creó un certificado auto firmado, usando `PowerShell`[46]. `PowerShell` es una consola que permite entre otras cosas la creación de certificados a través de código a través de la instrucción `SelfSignedCertificate`.

El **Segmento de código 2.40** muestra el comando empleado para la creación del certificado, el parámetro `Subject` permite configurar el *common name* (CN) del certificado, el cual es `WCFServiceTestCert`, el parámetro `keyUsage` permite especificar la intención de la llave pública contenida en el certificado. El parámetro `keySpec` permite especificar el tipo de operaciones de la llave pública, y `NotAfter` permite especificar la fecha de expiración del certificado.

La configuración del certificado permite asegurar los mensajes en los servicios, no se detalla el funcionamiento de cada uno de los parámetros del certificado porque está fuera del alcance de este Trabajo de Titulación.

```
SelfSignedCertificate -Subject "CN=WCFServiceTestCert" -KeyUsage
"DataEncipherment",    "KeyEncipherment",    "DigitalSignature",
"NonRepudiation"   -KeySpec   "KeyExchange"   -NotAfter   (get-
date).AddYears(20)
```

Segmento de código 2.40 Comando usado para la creación del certificado

La **Figura 2.37** muestra la interfaz de consola de `PowerShell` luego que haber generado el certificado usando el comando especificada anteriormente.

La aplicación de Windows *Microsoft Management Console* permite visualizar los certificados instalados en el equipo. En la **Figura 2.38** se muestra la interfaz de *Microsoft Management Console* desplegando el certificado `WCFServiceTestCert` que fue instalado.

---

[46] PowerShell es una interfaz de consola (CLI) con posibilidad de escritura y unión de comandos por medio de instrucciones.

En WCF, un *behavior* o comportamiento es una configuración asociada a un `endpoint` del servicio. Para configurar el comportamiento del `endpoint`, se incluyó en el archivo de configuración del servicio WCF `web.config` la sección de código XML presentada en el **Segmento de código 2.41**.

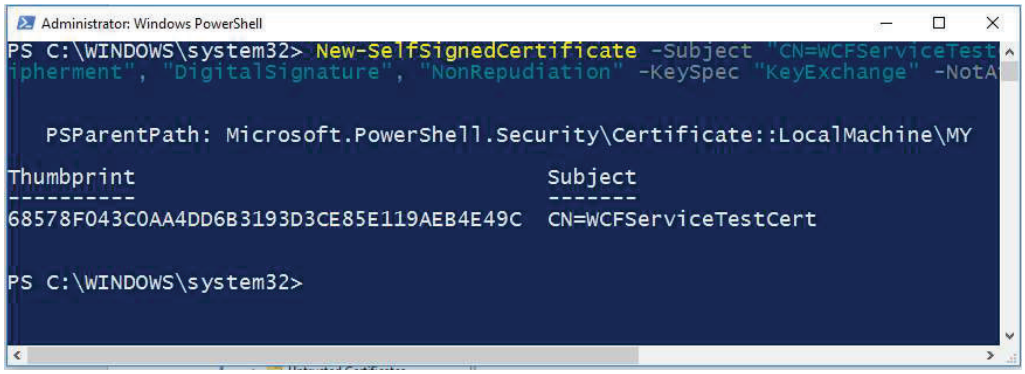


Figura 2.37 Creación de certificado usando `PowerShell`

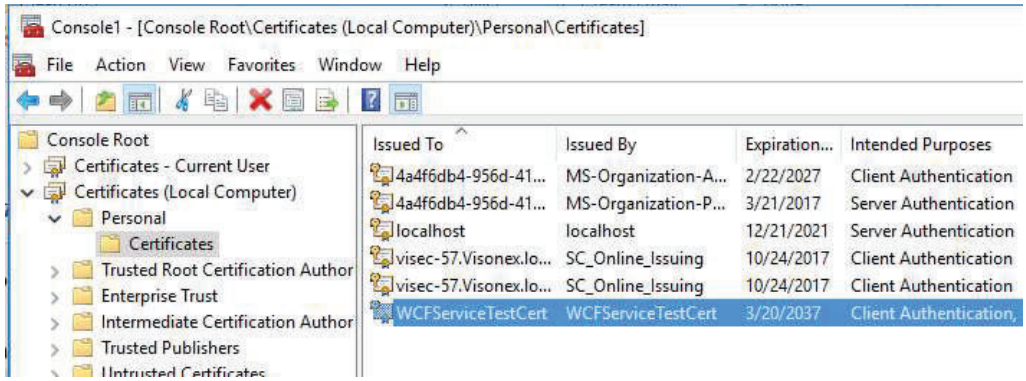


Figura 2.38 Interfaz Microsoft Management Console desplegando los certificados instalados

En la línea 22 del **Segmento de código 2.41**, se configura el nombre del *behavior*, la línea 23 indica que el servicio soporta peticiones HTTP como peticiones HTTPS[47]. De la línea 26 a la 29 se configura el certificado a usarse para la comunicación. En la línea 26 y 27 se le especifica a WCF que busque el certificado según el nombre común CN, en la línea 27 y 28 se especifica la ubicación del certificado.

[47] HTTPS (*Hypertext Transfer Protocol Secure*) es un protocolo seguro de transferencia basado en HTTP.

```
22.   <behavior name="ServiceCredentialsBehavior">
23.     <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
24.     <serviceDebug includeExceptionDetailInFaults="true"/>
25.     <serviceCredentials>
26.       <serviceCertificate findValue="WCFServiceTestCert"
27.                           storeLocation="LocalMachine"
28.                           storeName="My"
29.                           x509FindType="FindBySubjectName"/>
30.     </serviceCredentials>
31.   </behavior>
```

Segmento de código 2.41 Configuración de *behaviour* del servicio WCF

Para especificar que el `endpoint` del servicio WCF use las configuraciones de `binding` y de `behavior` desarrolladas anteriormente, es necesario especificar en la configuración del `endpoint` los nombres de las configuraciones de `binding` y `behaviour`, tal como se muestra en el **Segmento de código 2.42**. En la línea 41 se indica el nombre de la configuración de `behavior` que el `endpoint` del servicio usará. En la línea 44 se indica el nombre de la configuración de `binding` que el `endpoint` del servicio usará.

Una vez asegurada la integridad de los mensajes, es posible confiar en las credenciales del usuario que llegan dentro de cada mensaje. El siguiente y último paso es validar las credenciales del usuario y permitirle o negarle el acceso a las operaciones del servicio.

```
41.   <service behaviorConfiguration="ServiceCredentialsBehavior"
      name="ServiceTest.Service1">
42.     <endpoint address="PatientService.svc"
43.               binding="wsHttpBinding"
44.               bindingConfiguration="MessageAndUserName"
45.               name="PatientServiceEndpoint"
46.               contract="ServiceTest.IService1"/>
47.   </service>
```

Segmento de código 2.42 Configuración del `endpoint` del servicio `PatientService`

El **Segmento de código 2.43** muestra un ejemplo del validador de credenciales del lado del servicio WCF. En la línea 14 se muestra cómo validar el nombre de usuario y la contraseña, pero esta implementación es solo de demostración, ya que la implementación real llama a la base de datos para obtener las credenciales reales del usuario y compararlas con las credenciales que vienen en el mensaje. La línea 16 permite lanzar una excepción cuando las credenciales no fueron validas, por ende, el usuario no tiene acceso a ninguna operación del servicio.

```
12. public override void Validate(string userName, string password)
13. {
14.     if (userName == "test" && password == "test")
15.         return;
16.     throw new SecurityTokenException(
17.         "Unknown Username or Password");
18. }
```

Segmento de código 2.43 Método que valida las credenciales del usuario

2.3.6 SPRINT 4

2.3.6.1 Sprint planning

En la **Tabla 2.49** se muestra los ítems escogidos para cumplir con los objetivos del *sprint* 4: Construir los reportes definidos en la etapa de diseño, implementar la generación de *logs* y generar varios pruebas unitarias en el sistema.

2.3.6.2 Desarrollo del sprint 4

2.3.6.2.1 Construcción de reportes

Para la generación de reportes, se usó la herramienta `DoodleReport` [35]. Esta herramienta permite la generación de reportes a documentos Excel, HTML, PDF, entre otros. `DoodleReport` genera los reportes usando una lista como fuente de datos.

Esta solución para generar reportes tiene las ventajas de ser gratuita y de ser de fácil implementación en una aplicación web de MVC.NET, sin embargo, tiene la desventaja de disponer de funcionalidades muy básicas respecto a otras herramientas de generación de reportes.

Tabla 2.49 *Sprint backlog* 4

Prioridad	Título	Estado	Tipo de ítem de trabajo
1	*Build Allergies report*	*Committed*	*Product Backlog Item*
2	*Build Immunizations report*	*Committed*	*Product Backlog Item*
3	*Build Current Infections report*	*Committed*	*Product Backlog Item*
4	*Build Current Medications report*	*Committed*	*Product Backlog Item*
5	*Research LogN third party logging*	*Committed*	*Product Backlog Item*
6	*Implement logging in Web application*	*Committed*	*Product Backlog Item*
7	*Implement logging in Services layer*	*Committed*	*Product Backlog Item*
8	*Implement logging in BLL*	*Committed*	*Product Backlog Item*
9	*Implement unit testing in different layers of the system*	*Committed*	*Product Backlog Item*

La herramienta DoodleReport no permite el diseño explícito del contenido del reporte, es decir, no permite establecer la ubicación de rótulos y agrupamiento de texto. DoodleReport solo permite generar filas de texto con datos que obtiene de las listas que provienen desde el servicio WCF de reportes.

Para implementar la generación de reportes, se realizaron los siguientes pasos:

1. Definir los reportes a ser implementados. Los mismos fueron definidos en la sección 2.3.1.7.7.
2. Crear un nuevo servicio WCF `ReportService` con operaciones que retornen listas de información para que se conviertan en las fuentes de datos para la generación de reportes. Todas las operaciones retornan una lista de objetos, y reciben como parámetro los filtros previamente definidos. Se implementó un nuevo servicio WCF denominado `ReportService` el cual expone operaciones que devuelven listas usados para el despliegue de reportes. La **Tabla 2.50** muestra las operaciones implementadas en el

servicio `ReportService`. Cada una de ellas recibirá como parámetro los filtros especificados por el usuario a través de la interfaz de usuario de reportes, y devolverá como resultado una lista de datos que permitirá dibujar cada uno de los reportes. Los detalles de la implementación del servicio WCF `ReportService` y de sus operaciones no serán incluidos en esta sección, ya que la misma es similar a la implementación del servicio WCF `PatientService` mostrada anteriormente. El código implementado para la construcción del servicio `PatientService` se muestra en el Anexo A.

Tabla 2.50 Operaciones del servicio `ReportService` implementadas

Reporte	Operación del servicio `ReportService`	Descripción
`Allergies`	`AllergiesReport`	Devuelve una lista de objetos de tipo `AllergyDTO`.
`Immunizations`	`ImmunizationsReport`	Devuelve una lista de objetos de tipo `ImmunizationDTO`.
`Current Infections`	`CurrentInfectionsReport`	Devuelve una lista de objetos de tipo `InfectionDTO`.
`Current Medications`	`CurrentMedicationsReport`	Devuelve una lista de objetos de tipo `MedicationDTO`.
`On Hold Medications`	`OnHoldMedsReport`	Devuelve una lista de objetos de tipo `MedicationDTO`.
`Treatments`	`TreatmentsReport`	Devuelve una lista de objetos de tipo `TreatmentDTO`.
`Active Vascular Accesses`	`ActiveVascularAccessesReport`	Devuelve una lista de objetos de tipo `VascularAccessDeviceDTO`.

3. Instalar y configurar la herramienta de generación de reportes `DoodleReport` en la aplicación web MVC.NET.

Para instalar y configurar la herramienta de reportes en la aplicación web MVC.NET se realizaron los siguientes pasos:

- Instalación de los paquetes *nuget* requeridos para ejecutar la herramienta. Estos son: `DoddleReport` que contiene librerías necesarias para ejecutar la herramienta de generación de reportes y el paquete `DoddleReportWeb` que permite la integración con ASP.NET. Por defecto, la librería principal permite escribir reportes en formato HTML y TXT.
- Instalación de los paquetes *nuget* que permiten escribir reportes en otros formatos. El paquete `DoodleReport.iTextSharp` permite escribir reportes en formato PDF, mientras que `DoodleReport.OpenXml` permite generar reportes en formato XLSX (formato de hoja de cálculo de Excel).
- Añadir la configuración necesaria en el archivo de configuración `Web.config` de la aplicación web de MVC.NET. Esta configuración incluye definir estilos de letra por defecto, y fijar los escritores de reportes. Un escritor de reportes es aquel que permite determinar el formato al cual la herramienta `DoodleReport` será capaz de generar. El sistema permitirá ejecutar reportes en los formatos: HTML, PDF y XLSX.

El **Segmento de código 2.44** muestra parte del código de configuración realizada en la aplicación web para el funcionamiento de la herramienta DoodleReport. Esta configuración fue realizada en el archivo de configuración `web.config`.

En la línea 103 se define los nombres de los estilos que serán aplicados a cada sección del reporte, es decir, se define un nombre de estilo aplicable para todas las filas del reporte (`DataRowStyle`), otro para la cabecera (`HeaderRowStyle`) y otro para el pie del reporte (`FooterRowStyle`). También se define el valor por defecto del formato en que se generan los reportes, en este caso está definido que por defecto los reportes se van a ejecutar en formato XLSX, o formato Excel.

Desde la línea 105 hasta la 109, se configura cada uno de los formatos definidos anteriormente en base a un nombre. Por ejemplo, en la línea 108 se establece que el pie de reporte esté en cursivas.

Finalmente, entre las líneas 111 y 117 se definen los generadores o escritores de reportes. Cada escritor de reporte genera en un formato específico. Por ejemplo, en la línea 117 se define al escritor `iTextSharpPdf`, el cual genera reportes en

formato PDF. DoodleReport hace uso de librerías de terceros para la generación de reportes en diferentes formatos.

```
103. <doddleReport defaultWriter="ExcelOpenXml" dataRowStyle="DataRowStyle"
   headerRowStyle="HeaderRowStyle" footerRowStyle="FooterRowStyle">
104.     <styles>
105.       <style name="DataRowStyle" />
106.       <style name="HeaderRowStyle" bold="true" underline="true" />
107.       <style name="FooterRowStyle" bold="true" />
108.       <style name="Footer" italic="true" />
109.       <style name="Title" fontSize="16" />
110.     </styles>
111.     <writers>
112.       <clear />
113.       <add format="Html" type="DoddleReport.Writers.HtmlReportWriter,
   DoddleReport" contentType="text/html;charset=UTF-8" fileExtension=".html"
   />
114.       <add format="Text"
   type="DoddleReport.Writers.DelimitedTextReportWriter, DoddleReport"
   contentType="text/plain;charset=UTF-8" fileExtension=".txt"
   offerDownload="true" />
115.       <add format="Excel" type="DoddleReport.Writers.ExcelReportWriter,
   DoddleReport" contentType="application/vnd.ms-excel" offerDownload="true"
   fileExtension=".xls" />
116.       <add format="ExcelOpenXml"
   type="DoddleReport.OpenXml.ExcelReportWriter, DoddleReport.OpenXml"
   contentType="application/vnd.openxmlformats-
   officedocument.spreadsheetml.sheet" offerDownload="true"
   fileExtension=".xlsx" />
117.       <add format="iTextSharpPdf"
   type="DoddleReport.iTextSharp.PdfReportWriter, DoddleReport.iTextSharp"
   contentType="application/pdf" offerDownload="false" fileExtension=".pdf"
   /></writers>
118. </doddleReport>
```

Segmento de código 2.44 Configuración de la herramienta DoodleReport en la aplicación web

4. Implementar la interfaz de usuario `Reports` para ejecutar reportes en la aplicación web MVC.NET. El código que genera la interfaz de usuario de reportes se muestra en el Anexo A.
5. Construir el controlador de MVC `DoodleReportController` con las acciones necesarias para generar cada uno de los reportes. El **Segmento de código 2.45** muestra la acción para generar el reporte `Allergies`.

```
6. public ReportResult Allergies(int? patientID, string patientStatus,
   string patientSex, DateTime startDate, DateTime endDate)
7. {
8.     var query = GetAllergies(patientID, patientStatus, patientSex,
   startDate, endDate);
9.
10.    var report = new Report(query.ToReportSource());
11.
12.    report.TextFields.Title = "Allergies Report";
13.    report.TextFields.SubTitle = "This is a sample report showing how
   Doddle Report works";
14.    report.TextFields.Footer = "Copyright 2009 &copy; The Doddle
   Project";
15.    report.TextFields.Header = string.Format("Report Generated: {0}",
   DateTime.Now);
16.    report.RenderHints.BooleanCheckboxes = true;
17.
18.    return new ReportResult(report);
19. }
```

Segmento de código 2.45 Acción MVC para generar reporte `Allergies`

En la línea 6 la acción recibe los filtros de reportes como parámetro, en la línea 8 se llama a una de las operaciones del servicio `ReportService` para obtener la lista de alergias a desplegarse en el reporte. Entre las líneas 12 y 14 se configura el texto del título, subtítulo, pie y cabecera del reporte. La línea 16 permite que los campos que sean de tipo `boolean` sean mostrados como `checkbox`.

La **Figura 2.39** muestra como luce el reporte `Allergies` luego de ser desplegado en formato XLSX (formato Excel).

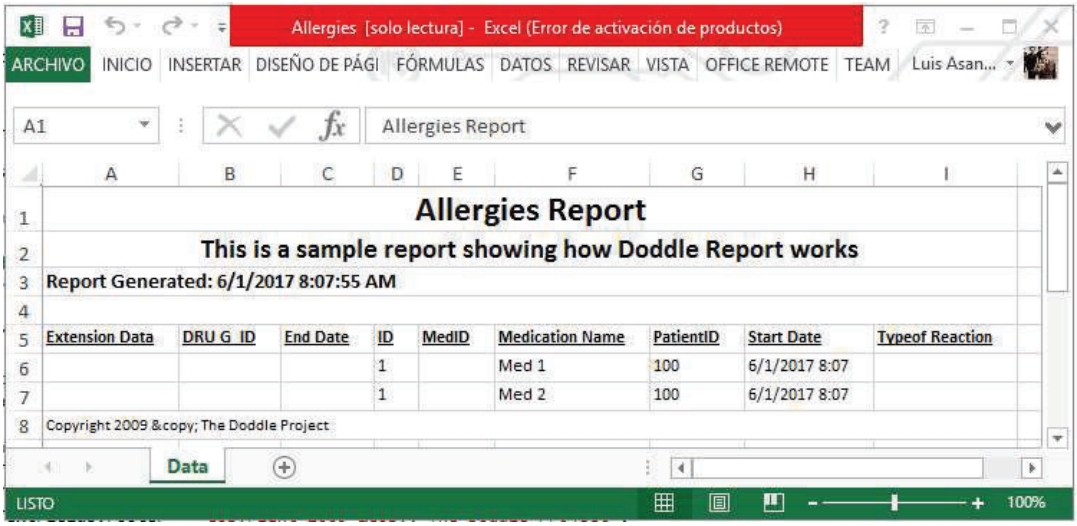


Figura 2.39 Reporte `Allergies` desplegado en formato Excel

2.3.6.2.2 Generación de logs

La generación de *log* fue implementada usando la herramienta log4Net. Se realizaron los siguientes pasos:

- Creación del servicio WCF para la generación de *logs*. El nuevo servicio WCF se denominó `LoggingService`. Para ello se creó un nuevo proyecto de aplicación WCF en Visual Studio de la misma manera que se hizo para crear el servicio `PatientService`. El código implementado para la creación de este servicio se muestra en el Anexo A.
- Instalación del paquete *nuget* de log4Net en el proyecto de servicio WCF `LoggingService`.
- Definir sección `root` en el archivo de configuración `Web.Config` para definir los generadores de *log* `appender`. Un `appender` sirve para definir quien genera los *logs*, por ejemplo, la aplicación web MVC es un generador de *logs*, pero los servicios web también son generadores, para lo cual se definen varios `appender`. De esta manera se podrá saber desde donde se genera el error.
- Configurar los `appender`. Por ejemplo, se puede configurar para que se filtre los *logs* que un `appender` en específico genera con base en un carácter o cadena de caracteres dentro de la información que genera el *log*, o sino en base a nivel o rangos de niveles de *log*.

- Añadir referencia de log4Net DLL en el archivo de configuración `Web.Config`.
- Implementar la llamada que genera el *log*.

Existen principalmente dos actores dentro de la generación de *logs*, uno es el cliente que llama al servicio `LoggingService` y le solicita que genere los *logs*, y por otro lado está el servicio quien recibe la petición.

El cliente que llama al servicio WCF puede ser la aplicación de MVC, o puede ser uno de los otros servicios WCF, ya sea PatientService, ReportService, PhysicianService, etc.

Cuando el cliente pide al servicio que genere logs, este debe indicar por parámetro quien es, y el nivel de *logging* que está generando. Por otro lado, el servicio `LoggingService` filtrará la generación de *logs* en base a la configuración establecida, es decir, la configuración indicará el nivel mínimo de *logs* que deberá registrar.

El **Segmento de código 2.46** muestra la implementación de la operación `WriteLog` del servicio.

```
23.  public void WriteLog(string message, LoggingLevel level, AppLogger
     appLogger, Exception exception)
24.  {
25.      try
26.      {
27.          log4net.ILog log =
     log4net.LogManager.GetLogger(appLogger.ToString());
28.          GenerateLog(ref log, message, level, exception);
29.      }
30.      catch (Exception)
31.      {
32.      }
33.
34.      throw new Exception();
35.  }
```

Segmento de código 2.46 Operación del método `LoggingService`

Entre las líneas 25 y 32 se muestra la sentencia `try – catch` que fue añadida debido a que la generación de *logs* no debería por ningún motivo generar excepciones que puedan interrumpir el normal funcionamiento del resto de las aplicaciones. La línea 27 por otro lado está obtiene el objeto generador de *logs* de la librería Log4Net, luego con ese objeto se llama al método `GenerateLog` en la línea 28, el cual genera el registro de *log* en la base de datos de *logs*.

El **Segmento de código 2.47** muestra la implementación del lado del cliente aplicación web MVC para generar *logs* en caso de que exista una excepción. La clase `HandleFaultExceptionAttribute` es un tipo de filtro de MVC. Este código es ejecutado cada vez que existe un error en la aplicación. En la línea 12 se instancia el cliente del servicio `LoggingService`, y se llama a la operación `WriteLog`. Cuando se añade la referencia del servicio web a la aplicación web MVC, con una configuración adicional se puede generar automáticamente operaciones asincrónicas. Una operación asincrónica es aquella que se ejecuta paralelamente mientras sigue la ejecución de la aplicación cliente, en este caso, la aplicación MVC. Es por ello por lo que en la línea 13 se llama a versión asincrónica de la operación `WriteLog`. En la línea 13 también se muestra los parámetros de entrada, estos son: El mensaje de la excepción, el nivel de *log* y el *logger* (quien genera los *logs*).

```
6.    public class HandleFaultExceptionAttribute : FilterAttribute, IExceptionFilter
7.    {
8.        public void OnException(ExceptionContext filterContext)
9.        {
10.           if (!filterContext.ExceptionHandled && filterContext.Exception is
      FaultException)
11.           {
12.               var client = new
      LogWritterLocal.LoggingServiceClient();
13.               client.WriteLogAsync(filterContext.Exception,
      LogWritterLocal.LoggingLevel.DEBUG, LogWritterLocal.AppLogger.MedicalSystem,
      null);
14.           }
15.       }
16.   }
```

Segmento de código 2.47 Generación de *log* en el filtro de excepciones `HandleFaultExceptionAttribute`

CAPÍTULO 3. PRUEBAS Y RESULTADOS OBTENIDOS

3.1 INTRODUCCIÓN

En este capítulo se presentarán los resultados de las pruebas unitarias y de orden superior que se realizaron al sistema. Finalmente, se indicará un resumen de algunos errores que se encontraron durante el desarrollo del sistema.

Para la ejecución de las pruebas, tanto la aplicación web como las bases de datos y los servicios web del sistema fueron publicados en la nube mediante la plataforma de servicios Microsoft Azure.

La aplicación web está disponible a través del siguiente enlace: http://medicalsystemtesis.azurewebsites.net/.

3.2 PRUEBAS

3.2.1 PRUEBAS UNITARIAS

Visual Studio permite crear proyectos de pruebas unitarias dentro de una solución. En la ventana *Solution Explorer* se hace clic derecho sobre la solución, y se selecciona la opción: "Add New Project". En la ventana "Add New Project" seleccionar la opción Visual C# → Test → Unit Test Project. La **Figura 3.1** muestra la ventana que permite añadir un nuevo proyecto de pruebas unitarias a la solución de Visual Studio.

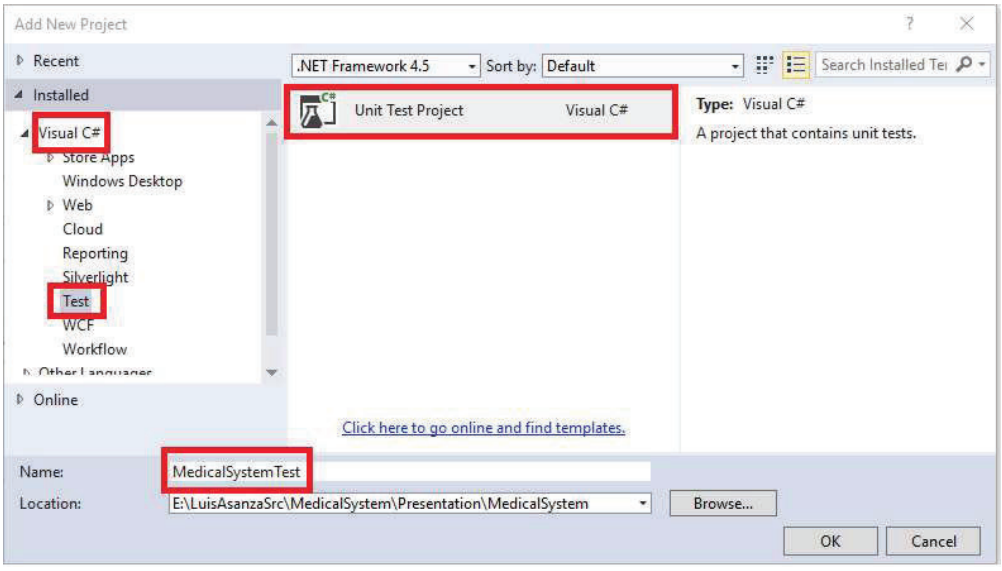


Figura 3.1 Añadir proyecto de pruebas unitarias a la solución de Visual Studio

En esta sección se detallará un ejemplo de pruebas unitarias realizado en la capa de presentación, específicamente en uno de los controladores de MVC.

Los proyectos de pruebas unitarias, por estándar, deben ser nombrados con el nombre del proyecto al cual se hará las pruebas unitarias, más el sufijo `Test`. Por ejemplo, el proyecto de aplicación web se denomina `MedicalSystem`, por tanto, el proyecto de pruebas unitarias se llamará `MedicalSystemTest`.

En una solución de Visual Studio puede existir más de un proyecto que no sea de pruebas unitarias, por tanto, es muy importante nombrar apropiadamente los proyectos de pruebas unitarias para evitar confusiones. El proyecto de pruebas unitarias debe tener referencias del proyecto del cual se harán las pruebas.

En [36] se indica como construir 3 diferentes tipos de pruebas unitarias para un controlador en una aplicación de MVC, una de ellas consiste en comprobar que una acción del controlador devuelva la vista esperada. Se realizó la prueba unitaria que comprueba que la acción `Allergy` del controlador `PatientController` devuelva la vista esperada `Allergy`, y los detalles de la implementación son presentados a continuación.

Antes de implementar la prueba unitaria, es necesario preparar al método `Allergy`. En el **Segmento de código 2.29** se presenta el código de la acción `Allergy` del controlador `PatientController`, la línea 75 permite a la acción `Allergy` retornar la vista con su modelo de MVC, el modelo es un objeto contenido en la variable `result`, mientras que el nombre de la vista que retorna es inferido en base al nombre de la acción, es decir, la acción `Allergy` retorna la vista del mismo nombre `Allergy`. Para poder realizar la prueba unitaria de comprobación de retorno de la vista esperada, es necesario que la acción indique explícitamente el nombre de la vista que retorna en vez de que la misma sea inferida, tal como se muestra en la línea 58 del **Segmento de código 3.1**, donde el método `View` retorna la vista. El primer parámetro del método `View` pemite establecer el nombre de la vista, y el segundo parámetro corresponde al modelo de MVC [36].

```
58.  return View("Allergy", result);
```

Segmento de código 3.1 Retorno de vista explícito

La línea de código 75 del **Segmento de código 2.29** de la acción `Allergy` del controlador `PatientController` fue reemplazada por la línea de código 58 del **Segmento de código 3.1**. De esta manera, el nombre de la vista retornada por la acción `Allergy` es explícitamente indicada.

Una vez preparado el método `Allergy` del controlador `PatientController`, se procedió a implementar la clase donde se implementarán las pruebas unitarias para la clase `PatientController`. Para ello, se añadió la carpeta `Controllers` dentro del proyecto de pruebas unitarias `MedicalSystemTest`, y dentro de esta carpeta se añadió la clase `PatientControllerTest`. El estándar para nombrar las clases de pruebas unitarias es similar al aplicado para nombrar los proyectos de pruebas unitarias, es decir, una clase de pruebas unitarias tiene por nombre el nombre de la clase a la cual se harán las pruebas unitarias más el sufijo `Test`.

Luego, dentro de la clase de pruebas unitarias `PatientControllerTest` se añadió el código de la prueba unitaria `AllergyViewReturn`. La prueba unitaria de comprobación de vista esperada se implementa a través de un método dentro de la clase `PatientControllerTest`. El **Segmento de código 3.2** muestra el código de la clase `PatientControllerTest`. En la línea 10 se aprecia que la misma está decorada con el atributo `[TestClass]`, así mismo en la línea 13 se muestra que el método `AllergyViewReturn` está decorado con el atributo `[TestMethod]`. Ambos atributos son requeridos en el marco de pruebas unitarias para código de .NET que se requiera ejecutar en el Explorador de pruebas[48]. En la línea 17 se crea una nueva instancia del controlador `PatientController`.

Luego, en la línea 18, se llama a la acción `Allergy`, la cual retorna un tipo `ActionResult`[49]. Las acciones de MVC típicamente retornan un resultado, el cual se conoce como resultado de acción (*action result*). Una acción puede retornar cualquier tipo de objeto, ya sea primitivo como un *string* (cadena de caracteres) o complejo como una vista de MVC, pero dicho objeto es envuelto en un resultado de

---

[48] Visual Studio incluye un *framework* de pruebas unitarias de Microsoft para código nativo. El explorador de pruebas (*Test Explorer*) es una ventana de Visual Studio desde donde se pueden ejecutar las pruebas unitarias usando el *framework* de pruebas unitarias de Microsoft.
[49] Es una clase abstracta que representa el resultado de una acción de MVC [37].

acción apropiado. En el caso de que la acción retorne una vista, el resultado de acción será de tipo `ViewResult`. `ActionResult` es la clase base para todos los resultados de acción [37].

```
10. [TestClass]
11. public class PatientControllerTest
12. {
13.     [TestMethod]
14.     public void AllergyViewReturn()
15.     {
16.
17.         var controller = new PatientController();
18.         var result = controller.Allergy() as ViewResult;
19.         Assert.AreEqual("Allergy", result.ViewName);
20.     }
21. }
```

Segmento de código 3.2 Prueba unitaria `AllergyViewReturn`

Finalmente, en la línea 19 se realiza la comprobación de la prueba unitaria, es decir, se comprueba que el nombre de la vista retornada sea `Allergy`. Las verificaciones de condiciones en pruebas unitarias se hacen a través de la clase `Assert`[50]. La clase `Assert` dispone de varios métodos para validar la condición de una prueba unitaria, tal es el caso del método `AreEqual`, el cual recibe como parámetros de entrada el valor esperado y el valor actual de la prueba unitaria. Para este caso de análisis, el valor esperado es el nombre de la vista `Allergy`, mientras que el valor actual será el nombre de la vista que retorne la acción `Allergy`. Cuando `AreEqual` es llamado, compara ambos valores y en caso de no coincidir, el Explorador de Pruebas desplegará información indicando que la prueba unitaria ha fallado; caso contrario, indica que fue existosa. La **Figura 3.2** muestra una captura del Explorador de Pruebas teniendo como resultado que la prueba unitaria `AllergyViewResult` fue existosa. Para más detalles sobre los métodos para verificar condiciones en pruebas unitarias usando proposiciones de verdadero y

[50] Clase estática usada para verificar condiciones en pruebas unitarias usando proposiciones de verdadero o falso [38].

falso, dirigirse a [38]. El código de las pruebas unitarias desarrolladas en las diferentes capas del sistema se encuentra en el ANEXO A.

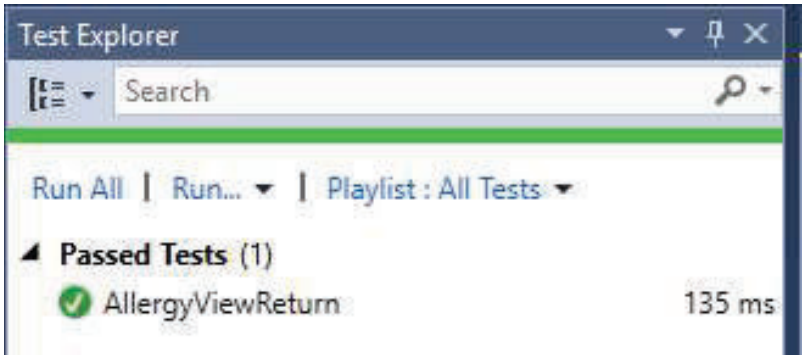


Figura 3.2 Prueba unitaria `AllergyViewReturn` exitosa

3.2.2 PRUEBAS DE ORDEN SUPERIOR

3.2.2.1 Pruebas de función

Las pruebas de función se basan en comparar el funcionamiento del sistema con la descripción del comportamiento del programa desde el punto de vista del usuario final. Para ello, se realizaron las siguientes pruebas:

3.2.2.1.1 Generación de reportes en distintos formatos

Para realizar esta prueba de función se generaron reportes en los distintos formatos que el sistema debe soportar. Estos son: XLSX, HTML y PDF.

La **Tabla 3.1** muestra la información de la prueba realizada para verificar el adecuado funcionamiento del generador de reportes en formato XLSX.

La **Figura 3.3** muestra una captura de pantalla del reporte generado en formato XLSX.

Tabla 3.1 Prueba de generación de reporte en formato XLSX

Caso de prueba	Generación de reporte `Allergies` en formato Excel (XLSX)	
Número de caso de prueba	1	
Variables	**Se espera**	**Se obtuvo**
Filtros de reportes: `PatientID` 100.	Ejecución normal del reporte. 2 registros.	Ejecución normal del reporte. 2 registros.

La **Tabla 3.2** muestra la información de la prueba realizada para verificar el adecuado funcionamiento del generador de reportes en formato HTML.

La **Tabla 3.3** muestra la información de la prueba realizada para verificar el adecuado funcionamiento del generador de reportes en formato PDF.

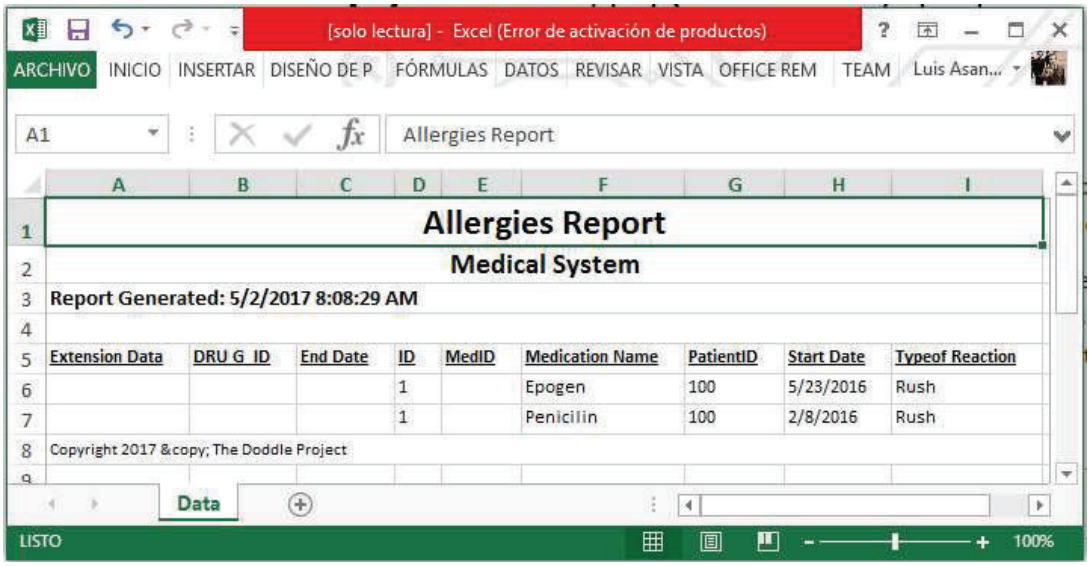


Figura 3.3 Prueba de generación de reporte en formato XLSX

Tabla 3.2 Prueba de generación de reporte en formato HTML

Caso de prueba	Generación de reporte `Allergies` en formato HTML	
Número de caso de prueba	2	
Variables	**Se espera**	**Se obtuvo**
Filtros de reportes: `PatientID` 158.	Ejecución normal del reporte. 0 registros.	Ejecución normal del reporte. 0 registros.

Tabla 3.3 Prueba de generación de reporte en formato PDF

Caso de prueba	Generación de reporte `Allergies` en formato PDF	
Número de caso de prueba	3	
Variables	**Se espera**	**Se obtuvo**
Filtros de reportes: `PatientID` 101.	Ejecución normal del reporte. 1 registro.	Ejecución normal del reporte. 1 registro.

La **Figura 3.4** muestra una captura de pantalla del reporte generado en formato HTML, mientras que la **Figura 3.5** muestra una captura de pantalla del reporte generado en formato PDF.

Figura 3.4 Prueba de generación de reporte en formato HTML

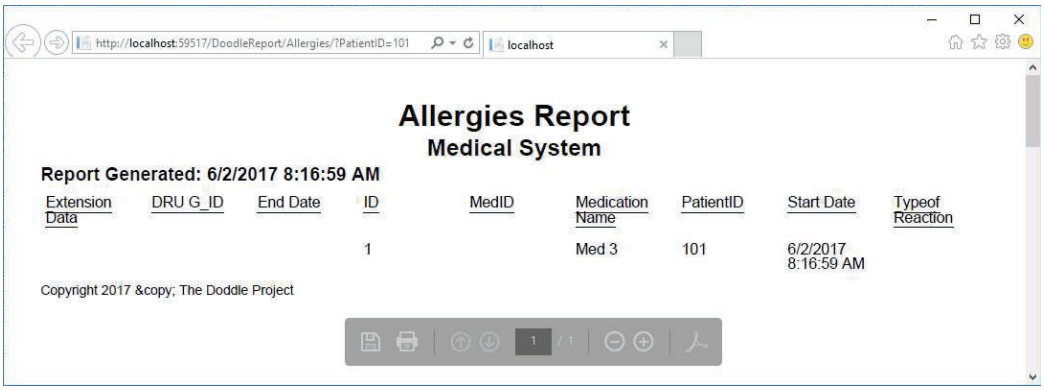


Figura 3.5 Prueba de generación de reporte en formato PDF

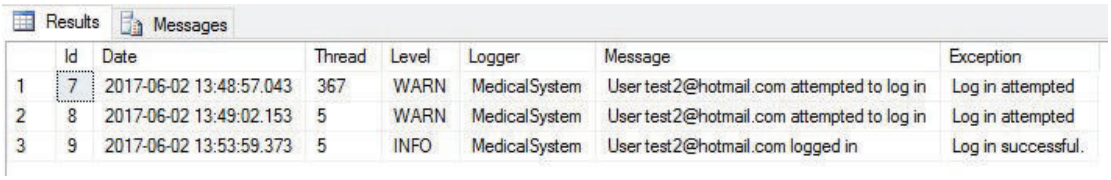
3.2.2.1.2 Generación de logs cuando un usuario inicia sesión

El evento de iniciar sesión o de intentar iniciar sesión al sistema es considerado como un evento de importancia en cuanto a vulnerabilidades de seguridad, por lo cual se debe generar un registro de *log*. Ante un ataque de seguridad al sistema, se podrá obtener información del mismo a través de la información que los *logs* proveen. Para esta prueba se usó un usuario de prueba, el cual intenta iniciar sesión 2 ocasiones, y luego finalmente logra acceder al sistema. Para este caso se deben generar 3 registros de *logs*. La información de la prueba de inicio de sesión de un

usuario y generación de *log* se muestra en la **Tabla 3.4**. La **Figura 3.6** muestra los registros generados en la tabla `Log`.

Tabla 3.4 Prueba de inicio de sesión y generación de registros de *logs*

Caso de prueba	Usuario intenta iniciar sesión 3 veces de las cuales 2 son fallidas.	
Número de caso de prueba	4	
Variables	**Se espera**	**Se obtuvo**
Usuario luisasanza@hotmail.com.	Generación de 3 registros de *logs*.	Generación de 3 registros de *logs*.



Results | Messages

	Id	Date	Thread	Level	Logger	Message	Exception
1	7	2017-06-02 13:48:57.043	367	WARN	MedicalSystem	User test2@hotmail.com attempted to log in	Log in attempted
2	8	2017-06-02 13:49:02.153	5	WARN	MedicalSystem	User test2@hotmail.com attempted to log in	Log in attempted
3	9	2017-06-02 13:53:59.373	5	INFO	MedicalSystem	User test2@hotmail.com logged in	Log in successful.

Figura 3.6 Prueba de intentos de inicio de sesión y generación de *logs*

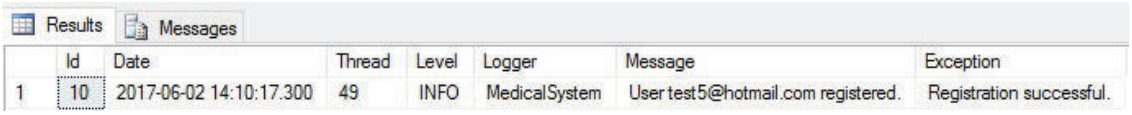
3.2.2.1.3 Generación de logs cuando un usuario se registra

El evento de registrar un usuario también debe generar registros de *logs* debido a que podría ser una fuente de vulnerabilidad del sistema. La presenta prueba consiste en realizar el registro de un nuevo usuario y verificar que efectivamente se haya creado el registro de *log* informativo indicando que tal evento sucedió.

La **Tabla 3.5** muestra el caso de prueba de generación de registro de *log* cuando sucedió el evento de registro de un usuario nuevo en el sistema. La **Figura 3.7** muestra el registro generado en la tabla `Log`.

Tabla 3.5 Prueba de registro de usuario y generación de registros de *logs*

Caso de prueba	Usuario se registra	
Número de caso de prueba	5	
Variables	**Se espera**	**Se obtuvo**
Usuario nuevo se registra. Correo de usuario test5@hotmail.com.	Se genera un registro de *log* indicando que un nuevo usuario se ha registrado.	Se genera un registro de *log* indicando que un nuevo usuario se ha registrado.



Results | Messages

	Id	Date	Thread	Level	Logger	Message	Exception
1	10	2017-06-02 14:10:17.300	49	INFO	MedicalSystem	User test5@hotmail.com registered.	Registration successful.

Figura 3.7 Prueba de registro de usuario y generación de registros de *logs*

3.2.2.2 Pruebas de sistema

Las pruebas de sistema se basan en comparar el funcionamiento del sistema con sus objetivos originales. Es por ello que se realizaron las siguientes pruebas:

3.2.2.2.1 Desplegar información médica de pacientes

En esta prueba se demuestra que se cumplió con el objetivo principal de desplegar información médica de pacientes. Solo se incluyen capturas de la interfaz de usuario `Infection`, por cuestión de espacio en la redacción.

La **Tabla 3.6** muestra el caso de prueba de desplegar información médica en el sistema.

Tabla 3.6 Prueba de despliegue de información médica

Caso de prueba	Desplegar información médica	
Número de caso de prueba	7	
Variables	**Se espera**	**Se obtuvo**
`PatientID` 2	Se despliegue información médica de pacientes en las interfaces de usuario	Se despliegue información médica de pacientes en las interfaces de usuario

La **Figura 3.8** muestra el despliegue de información médica en la interfaz de usuario `Infection`.

3.2.2.2.2 Verificar responsividad de la aplicación en un dispositivo móvil

Para realizar esta prueba, se accedió a la aplicación web desde un dispositivo móvil. Se verificó el acceso a las diferentes interfaces de usuario, y se verificó que funcione el inicio de sesión ingresando las credenciales adecuadas.

La **Tabla 3.7** muestra la información de la prueba de responsividad en un dispositivo móvil extra pequeño. También se tomó una captura de pantalla de la interfaz de inicio de sesión desde el dispositivo móvil y se la incluyó como parte de los resultados, la cual se muestra en la **Figura 3.9**.

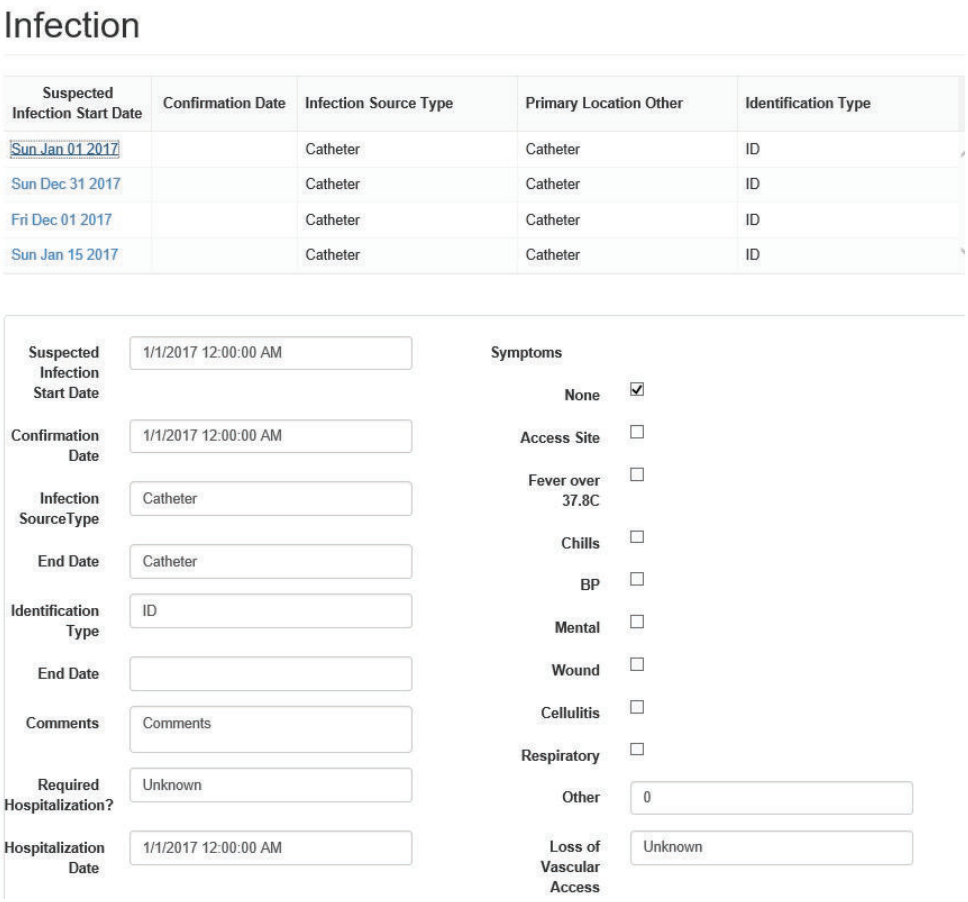


Figura 3.8 Prueba de despliegue de información médica

Tabla 3.7 Prueba de responsividad en dispositivo móvil extra pequeño

Caso de prueba	Prueba de responsividad en dispositivo móvil extra pequeño	
Número de caso de prueba	8	
Variables	**Se espera**	**Se obtuvo**
Correo electrónico del usuario: luisasanza@hotmail.com.	Interfaz de usuario se despliega según los *mockup* definidos. Funcionamiento adecuado.	Interfaz de usuario se despliega según los *mockup* definidos. Funcionamiento adecuado.

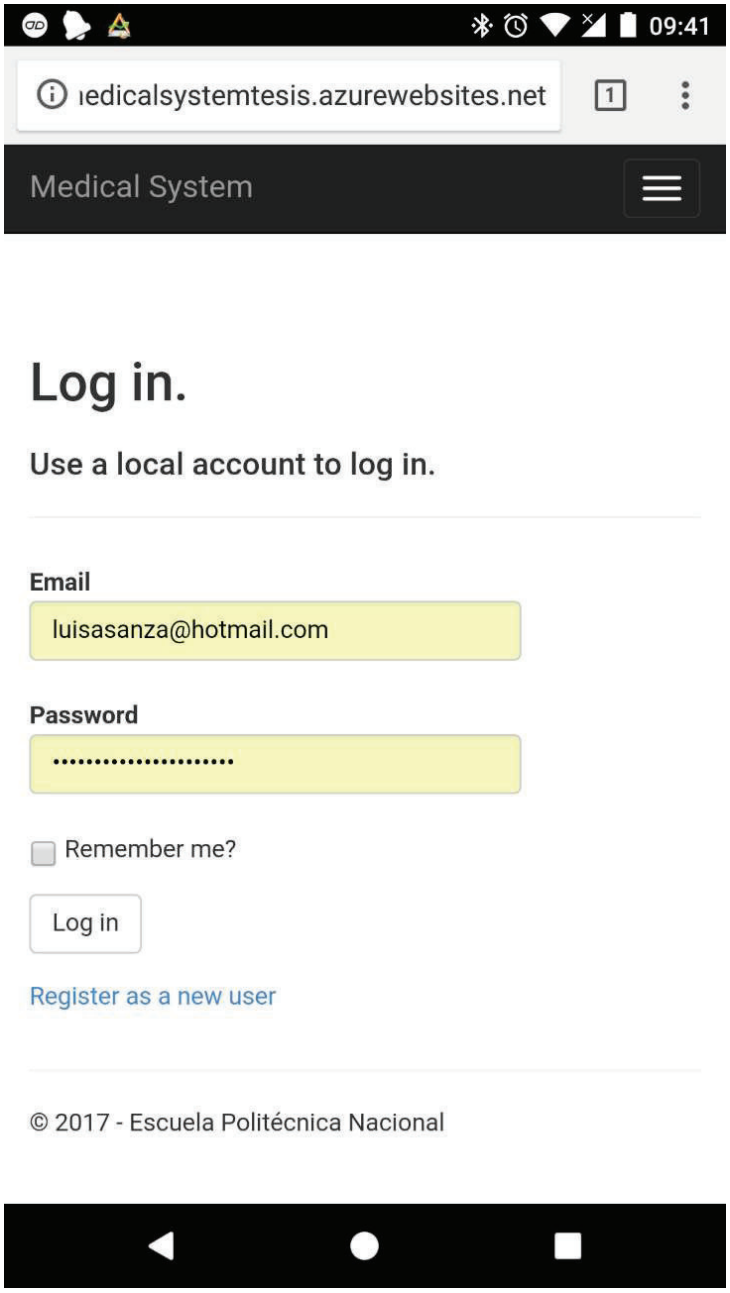


Figura 3.9 Prueba de responsividad en dispositivo móvil extra pequeño

3.2.2.2.3 Uso de credenciales falsas para ingresar el sistema

En esta prueba se realizaron varios intentos de vulnerar la seguridad al intentar acceder al sistema usando diferentes tipos de credenciales incorrectos. La **Tabla 3.8** muestra la información de la prueba de intento de vulnerabilidad de la seguridad al ingresar credenciales falsas. La **Figura 3.10** muestra una captura de pantalla de la interfaz de usuario `Log in` donde se indica el mensaje de error obtenido.

Tabla 3.8 Prueba de vulnerabilidad de seguridad ingresando credenciales incorrectas al sistema

Caso de prueba	Prueba de vulnerabilidad de seguridad con credenciales incorrectas	
Número de caso de prueba	9	
Variables	**Se espera**	
Correo electrónico del usuario: test2@hotmail.com	Sistema niegue el acceso. Mostrar un mensaje de error.	Sistema niegue el acceso. Mostrar un mensaje de error.

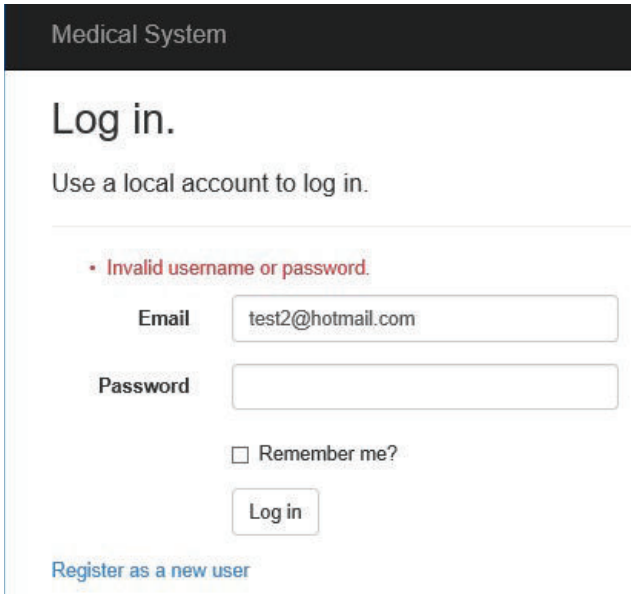


Figura 3.10 Prueba de vulnerabilidad de seguridad ingresando credenciales incorrectas al sistema

3.2.2.3 Pruebas de aceptación

Las pruebas de aceptación se basan en comparar el programa con sus requerimientos originales. Para ello se realizaron las siguientes pruebas:

- Actualización de la información de un usuario del sistema.
- Registro de un usuario del sistema.
- Compatibilidad del sistema con los exploradores web Google Chrome e Internet Explorer.

3.2.2.3.1 Actualizar la información de un usuario a través de la interfaz de manejo de usuarios

Mediante esta prueba se verificó el funcionamiento de guardar un registro de un usuario existente a través de la interfaz de manejo de usuarios. La **Tabla 3.9** muestra la información de la prueba de modificación de un usuario. La **Figura 3.11** muestra una captura de la interfaz de usuario donde se hizo la modificación del registro.

Tabla 3.9 Prueba de modificación de un registro de usuario de sistema

Caso de prueba	Actualizar un registro de usuario a través de la interfaz de usuario	
Número de caso de prueba	10	
Variables	**Se espera**	**Se obtuvo**
`UserID` 1	Sistema permite modificar los campos del registro de un usuario y guardarlos en base de datos.	Sistema permite modificar los campos del registro de un usuario y guardarlos en base de datos.

3.2.2.3.2 Registrar un usuario

Esta prueba consiste en registrar un nuevo usuario a través de la interfaz de usuario del sistema `Register`, y verificar que el usuario fue efectivamente creado. La **Tabla 2.1** muestra la información de la prueba del registro de un usuario nuevo al sistema. La **Figura 3.12** muestra una captura de la interfaz de usuario `Register`.

Tabla 3.10 Prueba de registro de usuario

Caso de prueba	Registro de un usuario	
Número de caso de prueba	11	
Variables	**Se espera**	**Se obtuvo**
Correo electrónico del usuario: TestCaseEmail@correo.com	Usuario se registra.	Usuario se registra.

Luego de que el usuario fue registrado, inició su sesión dentro de la aplicación. Se realizó la última prueba de cerrar y reiniciar sesión en el sistema.

La **Figura 3.13** muestra una captura de pantalla de la interfaz de usuario `Home` luego de que el usuario recién registrado inició sesión.

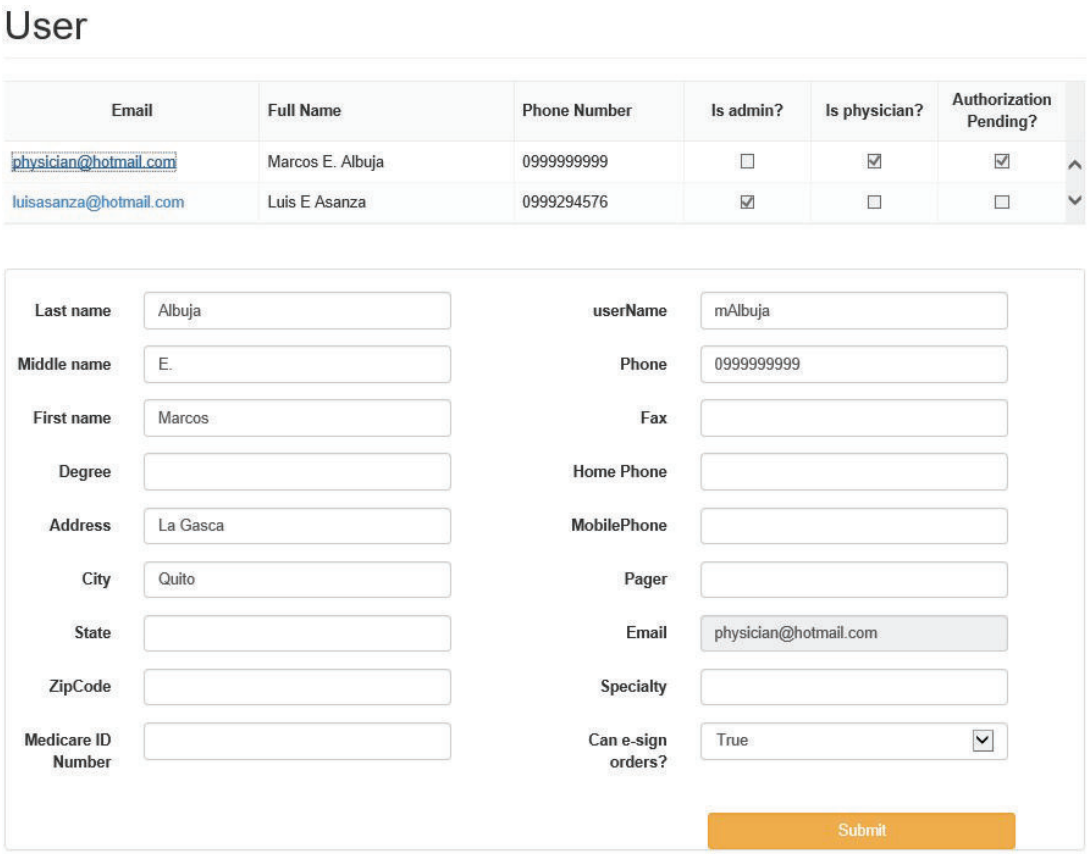


Figura 3.11 Prueba de modificación de un registro de usuario de sistema

3.2.2.3.3 Comprobar que el sistema sea soportado en los exploradores web Chrome e Internet Explorer

En esta prueba se verifica el funcionamiento de la aplicación web desplegada en los exploradores web Google Chrome e Internet Explorer.

Debido a que las anteriores pruebas fueron realizadas con el explorador web Internet Explorer, en esta prueba solo se verificará el funcionamiento en Chrome.

La **Tabla 3.11** muestra la información de la prueba de compatibilidad con el explorador web Google Chrome.

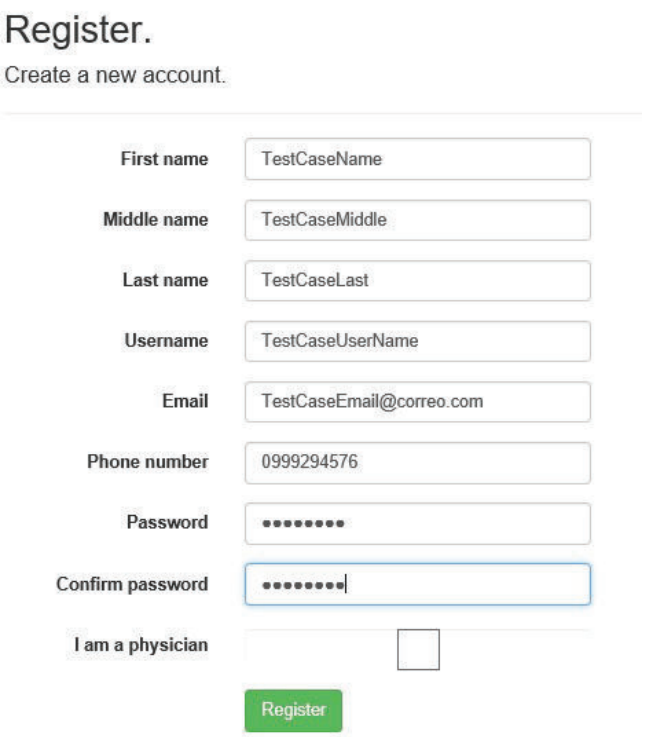


Figura 3.12 Prueba de registro de usuario

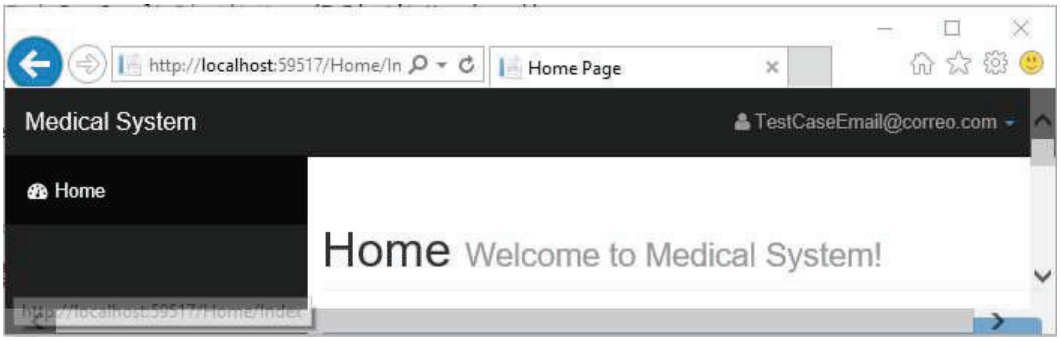


Figura 3.13 Prueba de registro de usuario luego del registro

Tabla 3.11 Prueba de compatibilidad con explorador web Google Chrome

Caso de prueba	Compatibilidad con exploradores web Chrome e Internet Explorer	
Número de caso de prueba	12	
Variables	**Se espera**	**Se obtuvo**
	Aplicación web se despliega en los exploradores Chrome e Internet Explorer.	Aplicación web se despliega en los exploradores Chrome e Internet Explorer.

La **Figura 3.14** y la **Figura 3.15** muestran capturas de diferentes interfaces de usuario de la aplicación corriendo sobre Google Chrome versión 58.0.3029.110 (64-bit).

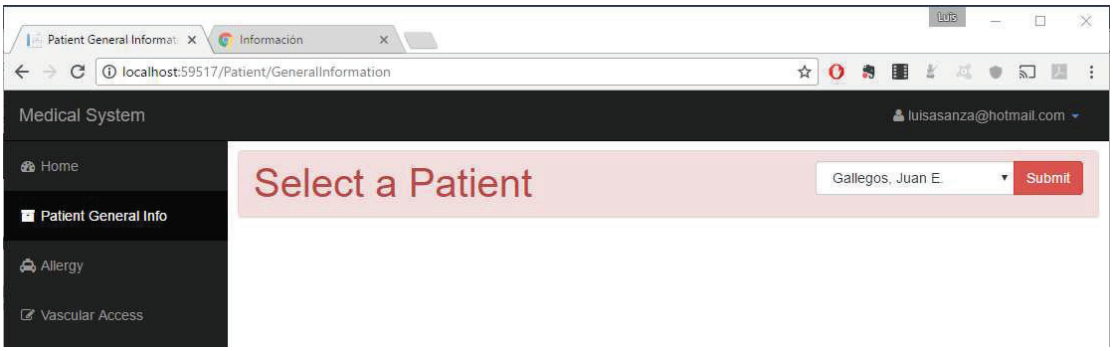


Figura 3.14 Prueba 1 de compatibilidad en explorador web Google Chrome

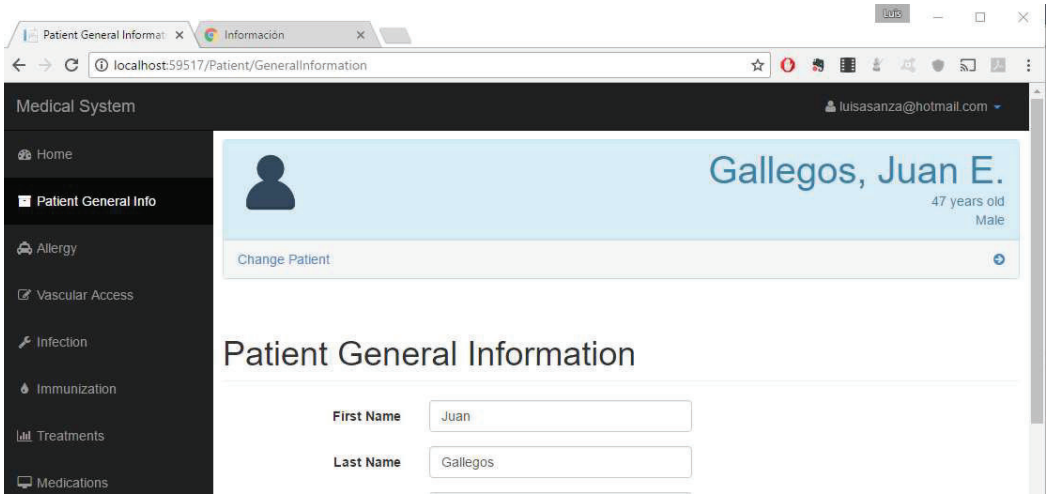


Figura 3.15 Prueba 2 de compatibilidad en explorador web Google Chrome

3.3 ERRORES ENCONTRADOS

3.3.1 Errores al momento de consumir los servicios WCF desde la aplicación web

Mientras se construía la interfaz de usuario para ver la información general del paciente, se presentó el siguiente error:

"An error occurred while receiving the HTTP response to http://localhost:52763/PatientService.svc. This could be due to the service endpoint binding not using the HTTP protocol. This could also be due to an HTTP request context being aborted by the server (possibly due to the service shutting down). See server logs for more details.]"

El mensaje de error no daba ninguna información sobre el problema real de lo que estaba sucediendo en el lado de los servicios web. Siguiendo con la metodología SCRUM, se añadió un nuevo *bug* dentro del sprint backlog.

Para poder encontrar la fuente del problema, se configuró el proyecto de servicio WCF para habilitar *tracing*[51]. De acuerdo con [39], cuando se habilita *tracing*, WCF puede arrojar información de excepciones en el código, alertas y otros procesos significativos.

Se añadió en el archivo de configuración las siguientes fuentes de *trace*:

- `System.ServiceModel.MessageLogging` para registrar todos los mensajes que fluyen a través del sistema.
- `System.ServiceModel` para registrar las etapas del procesamiento WCF como por ejemplo leer un mensaje, procesar un mensaje, seguridad, transporte, etc.
- `System.Runtime.Serialization` para generar registros cuando los objetos son leídos o escritos.

Finalmente se configuró un oyente (*listener)* que captura los registros y los almacena en un archivo destino.

Cuando se levanta el servicio WCF, automáticamente se genera un archivo de *logs* y puede ser accedido a través del Microsoft Service Trace Viewer cuya interfaz puede ser vista en la **Figura 3.16**.

Desde el Service Trace Viewer de Microsoft se pudo determinar la excepción que provocaba el mal funcionamiento del servicio WCF:

'System.Data.Entity.DynamicProxies.Allergy' with data contract name 'Allergy: http://schemas.datacontract.org/2004/07/System.Data.Entity.DynamicProxies' is not expected. Consider using a DataContractResolver if you are using DataContractSerializer or add any types not known statically to the list of known

---

[51] Rastreo (*tracing*) es una configuración de WCF que permite el rastreo de mensajes WCF entre el cliente y el servicio web. Cuando un cliente web hace una petición al servicio web y en este último se produce un error, el rastreo también puede identificar que tipo de error sucedió en el lado del servicio web.

types - for example, by using the KnownTypeAttribute attribute or by adding them to the list of known types passed to the serializer.

Las entidades de negocio que se autogeneran usando Entity Framework no podían ser deserializadas desde los servicios web debido a que requieren de la etiqueta `[DataContract]`, por lo que se decidió que la mejor manera de solucionar este problema era creando objetos DTO para el paso de información desde los servicios WCF hacia el cliente web, para de esa manera evitar los problemas de serialización de objetos complejos.

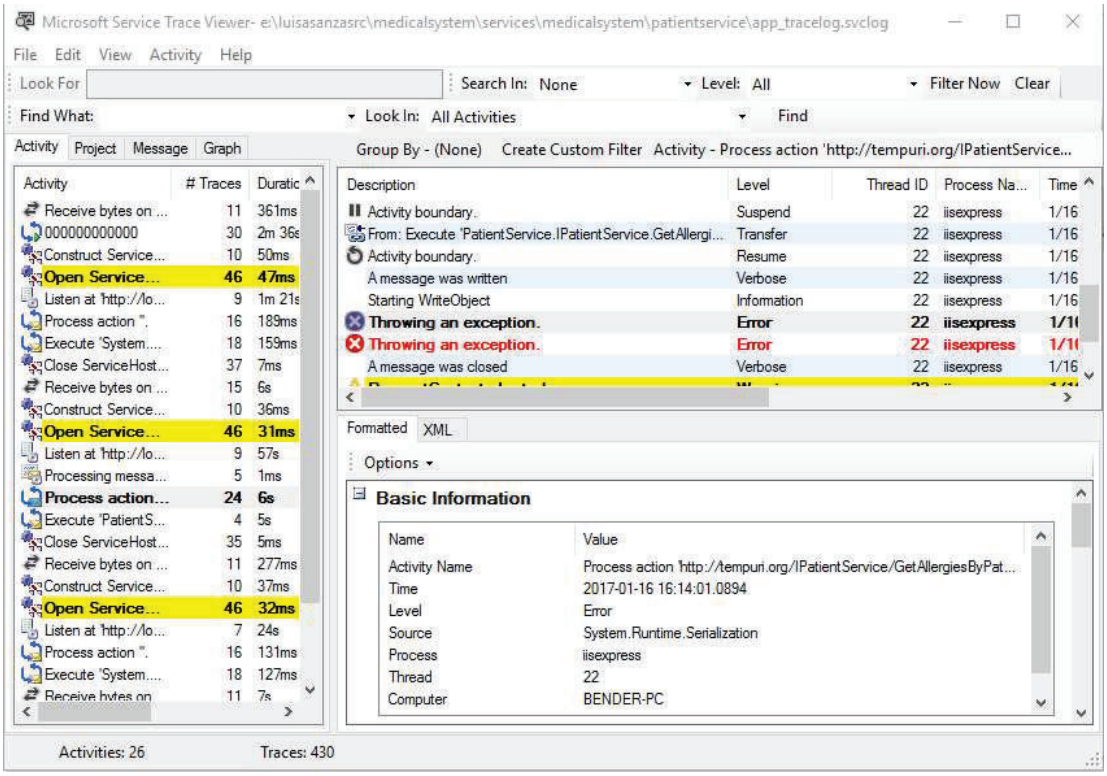


Figura 3.16 Microsoft Service Trace Viewer

3.3.2 Problemas de serialización en la comunicación entre el servicio WCF y el cliente

Otro problema presentado fue la imposibilidad de serializar un objeto al momento de comunicar un servicio WCF con el cliente web. Una propiedad es uno o dos bloques de código que representan un descriptor de acceso `get` y un descriptor de acceso `set` [40].

En la propiedad `Age` de la clase `TestPatientMinDTO` inicialmente solo había sido implementado el descriptor de acceso `get`, y debido a eso no se podía serializar al momento de mandar el objeto hacia el cliente web.

Para resolver el problema fue necesario definir el descriptor del acceso *set* a pesar de que no tuviera ninguna implementación dentro de este, tal como se ve en la línea 243 del **Segmento de código 3.3**.

Desde la línea 227 hasta la línea 242 se muestra el código del descriptor de acceso `get`, el cual permite obtener la edad de una persona en base a otra propiedad de la misma clase, denominada `Birthdate`. Cuando una propiedad se calcula en base a otra propiedad se la denomina propiedad calculada.

```
224.    [DataMember]
225.    public int Age
226.    {
227.        get
228.        {
229.            int age;
230.            age = DateTime.Now.Year - BirthDate.Value.Year;
231.
232.            if (age > 0)
233.            {
234.                age -= Convert.ToInt32(DateTime.Now.Date <
    BirthDate.Value.Date.AddYears(age));
235.            }
236.            else
237.            {
238.                age = 0;
239.            }
240.
241.            return age;
242.        }
243.        set { }
244.    }
```

Segmento de código 3.3 Propiedad Age de la clase `TestPatientMinDTO`

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Se diseñó un sistema distribuido basado en el estilo arquitectural SOA y el estilo arquitectural en capas. El sistema está conformado por 6 capas: capa de acceso a datos, capa de lógica de negocio, capa de servicios, capa de presentación, la capa de autorización y autenticación y la capa de *logs*. La capa de acceso a datos usa la tecnología Entity Framework para acceder a la base de datos con información médica. La capa de lógica de negocio como su nombre lo dice, contiene la lógica de negocio. La capa de servicios dispone de 3 servicios WCF que exponen métodos para retornar información médica de pacientes. La capa de presentación está conformada por una aplicación web de MVC.NET, donde se implementan las interfaces de usuario, la mayoría de ellas para desplegar información médica de pacientes, y otras para la generación de reportes y administrar usuarios del sistema y sus roles. La capa de autenticación y autorización controla el acceso de usuarios al sistema y la capa de *logs* se encarga de la generación de registros de *logs* a través de un cuarto servicio WCF.
- Se pudo evitar los errores de serialización al implementar objetos DTO para la comunicación entre el servicio y el cliente web, ya que estos objetos no poseen referencias con otras entidades de ida y vuelta.
- Se pudo aplicar el principio DRY al desacoplar la capa de autenticación y autorización, ya que esta podrá ser reutilizada desde diferentes partes del sistema.
- El *framework* Twitter Bootstrap demostró ser una opción óptima para la construcción de interfaces de usuario siguiendo el enfoque *mobile first*, ya que dispone de clases de estilos CSS que permiten diseñar la interfaz de usuario para dispositivos móviles, y sobre esa misma estructura, diseñar las interfaces para dispositivos de pantalla extra grande.
- La opción de cargado de datos *Eager loading* de Entity Framework evidenció ser una opción óptima para cargar entidades ya que se reduce el número de llamadas a la base de datos, volviendo más eficiente la comunicación entre el servicio web y los clientes.

- Al implementar el enfoque *code first* en la capa de Autenticación y Autorización, y el enfoque *database first* en la capa de acceso a datos, se evidenció que ambos enfoques pueden ser implementados simultáneamente en un mismo sistema.
- *Trace listener services* demostró ser una herramienta indispensable para poder rastrear los mensajes que se envían entre el servicio web y el cliente web y además visualizar la información de los errores que puedan generarse durante la comunicación.
- Las interfaces de usuario fueron implementadas de tal manera que el código de cliente JavaScript se cargara, en lo posible, como última prioridad, lo que demostró mejorar la experiencia de usuario, ya que los componentes de estructura y presentación de dichas interfaces se cargan con antelación, dando al usuario la percepción de que las páginas web son cargadas con mayor rapidez.
- La aplicación web fue desplegada en un dispositivo móvil de pantalla extra pequeña, y en PC, además se desplegó en diferentes exploradores web, demostrando así que cumplió con los requerimientos de responsividad y de compatibilidad.
- La herramienta de generación de reportes Doodle Report demostró ser una opción válida de reportería para fines prácticos dentro del alcance del presente trabajo de titulación, mas no como una solución empresarial, ya que la misma no dispone la funcionalidad de definir la estructura de un reporte.

4.2 RECOMENDACIONES

- Para trabajos futuros, se recomienda implementar algunas pruebas de seguridad como SQL *injection*, *Cross-Site Scripting* u otras, para añadir robustez a la aplicación.
- Se recomienda el uso de un certificado otorgado por una entidad certificadora que permita realizar la verificación de quienes están involucrados en la comunicación entre el servicio web y el cliente web.
- Al ser un sistema que se encuentra publicado en la nube, y que además posee información muy crítica, como lo es la información médica de

pacientes, se recomienda que se verifique los contratos de prestación de servicios en la nube, para asegurarse la integridad y confidencialidad de los datos.

- Para trabajos futuros, se recomienda el uso de una herramienta de generación de reportes diferente a Doodle Report, que disponga de características adicionales para la generación de reportes.
- Se recomienda, para trabajos futuros, revisar otras tecnologías de servicios web como REST debido a que los servicios WCF no son eficientes en la comunicación ya que cada mensaje SOAP incluye un sobrecoste (*overhead*) considerable causado por el XML.
- Se recomienda para una siguiente fase de este proyecto, implementar las operaciones de borrado y actualización de registros médicos, ya que estas no fueron parte del alcance.
- Como una mejora en la implementación del sistema, se recomienda reemplazar el manejo del ID del paciente seleccionado a través de una variable de sesión, por el manejo del mismo a través de la URL y enrutamiento de MVC, para así aprovechar de mejor manera las ventajas que MVC.NET ofrece.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Skonnard, «WCF Fundamentals | Pluralsight,» Pluralsight, 21 08 2008. [En línea]. Disponible: https://app.pluralsight.com/library/courses/wcf-fundamentals/table-of-contents. [Último acceso: 22 08 2016].

[2] C. House, «Architecting Applications for the Real World in .NET.,» Pluralsight, 07 01 2014. [En línea]. Disponible: https://app.pluralsight.com/library/courses/architecting-applications-dotnet/table-of-contents. [Último acceso: 15 01 2016].

[3] Microsoft Corporation, Microsoft Application Architecture Guide Patterns & Practices 2nd Edition, 2009.

[4] P. Eeles, «Capturing Architectural Requirements,» 15 11 2015. [En línea]. Disponible: http://www.ibm.com/developerworks/rational/library/4706.html#N100A7. [Último acceso: 25 11 2016].

[5] C. de la Torre, U. Zorrilla, M. Á. Ramos y J. Calvarro, Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0 (Beta), España: Krasis Press, 2010.

[6] P. Eeles, «Layering Strategies,» IBM Rational Software, 2001. [En línea]. Disponible: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct01/LayeringStrategiesOct01.pdf. [Último acceso: 25 11 2016].

[7] Microsoft Developer Network, «Chapter 5: Layered Application Guidelines,» Microsoft, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/ee658109.aspx. [Último acceso: 25 11 2016].

[8] Oracle Corporation, «SOA and Web Services,» 04 2005. [En línea]. Disponible: http://www.oracle.com/technetwork/articles/javase/soa-142870.html.

[9] W3C Working Group Note, «Web Services Architecture,» W3C, 11 02 2004. [En línea]. Disponible: https://www.w3.org/TR/ws-arch/#service_oriented_architecture. [Último acceso: 01 08 2016].

[10] W3C, «SOAP Version 1.2 Part 1: Messaging Framework (Second Edition),» 27 04 2007. [En línea]. Disponible: https://www.w3.org/TR/2007/REC-soap12-part1-20070427/. [Último acceso: 22 08 2016].

[11] W3C, «Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,» 10 11 2003. [En línea]. Disponible: https://www.w3.org/TR/2003/WD-wsdl20-20031110/. [Último acceso: 05 10 2016].

[12] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.

[13] A. Freeman, Pro ASP.NET MVC 5, Apress, 2013.

[14] Microsoft, «ASP.NET MVC | The ASP.NET Site,» Microsoft, 09 Agosto 2014. [En línea]. Disponible: https://www.asp.net/mvc. [Último acceso: 04 10 2016].

[15] E. Freeman, E. Freeman, B. Bates y K. Sierra, Head First Design Patterns, O'Reilly, 2004.

[16] MSDN Microsoft Developer Network, «ASP.NET Authentication,» MSDN, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/eeyk640h.aspx.

[17] MSDN Microsoft Developer Network, «ASP.NET Authorization,» MSDN, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/wce3kxhd.aspx.

[18] A. K. Talukder y M. Chaitanya, Architecting Secure Software Systems, Boca Ratón: CRC Press, 2009.

[19] W3C Recommendations, «What is the Document Object Model?,» World Wide Web Consortium (W3C), 03 11 2000. [En línea]. Disponible:

https://www.w3.org/TR/DOM-Level-2-Core/introduction.html. [Último acceso: 05 08 2016].

[20] Microsoft, Programming in HTML5 with JavaScript and CSS3 - Training Guide, Redmond, Washington 98052-6399: Microsoft Press, 2013.

[21] Bootstrap, «Bootstrap · The world's most popular mobile-first and responsive front-end framework.,» [En línea]. Disponible: http://getbootstrap.com/.

[22] B. Driscoll, N. Gupta, R. Vettor, Z. Hirani y L. Tenny, Entity Framework 6 Recipes - Second Edition, Apress, 2013.

[23] SCRUM Alliance, «The Scrum Guide | Scrum Alliance,» 24 09 2014. [En línea]. Disponible: https://www.scrumalliance.org/why-scrum/scrum-guide. [Último acceso: 02 11 2016].

[24] G. J. Myers, The Art of Software Testing, Second Edition, Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.

[25] Microsoft, «Create a New ASP.NET MVC Project,» Microsoft, 27 07 2010. [En línea]. Disponible: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/nerddinner/create-a-new-aspnet-mvc-project. [Último acceso: 21 03 2017].

[26] S. Smith, «Handling requests with controllers in ASP.NET MVC Core,» Microsoft, 14 10 2016. [En línea]. Disponible: https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions. [Último acceso: 28 12 2016].

[27] Start Bootstrap, "SB Admin - Free Bootstrap Admin Template - Start Bootstrap," [Online]. Disponible: https://startbootstrap.com/template-overviews/sb-admin/. [Accessed 01 10 2016].

[28] Microsoft Developer Network, «Entity Framework,» Microsoft, 16 10 2013. [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/gg696172(v=vs.103).aspx. [Último acceso: 15 02 2017].

[29] Microsoft Developer Network, «Entity Framework Loading Related Entities,» Microsoft, 23 10 2016. [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/jj574232(v=vs.113).aspx. [Último acceso: 25 12 2016].

[30] Oracle Help Center, «Handling Exceptions using SOAP Faults,» [En línea]. Disponible: https://docs.oracle.com/cd/E24329_01/web.1211/e24965/faults.htm#WSADV624. [Último acceso: 15 1 2017].

[31] Microsoft Developer Network, «Sending and Receiving Faults,» [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/ms732013(v=vs.110).aspx. [Último acceso: 20 11 2016].

[32] H. Sun, «Understanding OWIN Forms authentication in MVC 5,» 03 07 2013. [En línea]. Disponible: https://blogs.msdn.microsoft.com/webdev/2013/07/03/understanding-owin-forms-authentication-in-mvc-5/. [Último acceso: 15 02 2017].

[33] A. Tabalin , «Lightweight Grid jQuery Plugin,» [En línea]. Disponible: http://js-grid.com/. [Último acceso: 26 12 2016].

[34] Microsoft Developer Network, «Selecting a Credential Type,» Microsoft, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/ms733836(v=vs.110).aspx. [Último acceso: 02 02 2017].

[35] M. Hidinger, «GitHub - matthidinger/DoddleReport: Generate custom reports (PDF, Excel, etc) from any IEnumerable datasource.,» 13 05 2016. [En línea]. Disponible: https://github.com/matthidinger/DoddleReport. [Último acceso: 20 04 2017].

[36] S. Walther, «Creating Unit Tests for ASP.NET MVC Applications (C#),» Microsoft, 19 09 2008. [En línea]. Disponible: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/unit-testing/creating-unit-tests-for-asp-net-mvc-applications-cs. [Último acceso: 05 03 2017].

[37] Microsoft Developer Network, «ActionResult Class,» Microsoft, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/system.web.mvc.actionresult(v=vs.118).aspx. [Último acceso: 25 03 2017].

[38] Microsoft, «Assert Class (Microsoft.VisualStudio.TestTools.UnitTesting),» Microsoft, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx. [Último acceso: 12 04 2017].

[39] Microsoft Developer Network, «Configuring Tracing,» [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/ms733025(v=vs.110).aspx. [Último acceso: 25 12 2016].

[40] Microsoft, «Utilizar propiedades (Guía de programación de C#),» Microsoft, 24 03 2017. [En línea]. Disponible: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/using-properties. [Último acceso: 05 04 2017].

[41] Developer Express, [En línea]. Disponible: https://documentation.devexpress.com/#XtraReports/CustomDocument2162. [Último acceso: 02 02 2017].

[42] Microsoft, «Bundling and Minification,» 23 08 2012. [En línea]. Disponible: https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification. [Último acceso: 15 08 2016].

[43] J. Hasan y M. Duran, Expert Service-Oriented Architecture in C# 2005 2nd Edition, USA: Apress, 2005.

[44] Microsoft, «Introduction to ASP.NET Identity | The ASP.NET Site,» 17 10 2013. [En línea]. Disponible: https://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity. [Último acceso: 6 10 2016].

[45] L. Carvajal, Metodología de la Investgación Científica. Curso general y aplicado, 28 ed., Santiago de Cali: U.S.C., 2006, p. 139.

[46] IBM Corporation, Patterns: Service-Oriented Architecture and Web Services, 2004.

[47] N. Pathak, Pro WCF 4 Practical SOA Implementation 2nd Edition, USA: Apress.

[48] Telerik, «Reporting for HTML5, ASP.NET Ajax, MVC, WPF, Mobile, Touch, WinForms, Silverlight, Reports, Tool Software | Telerik,» Telerik, [En línea]. Disponible: http://www.telerik.com/products/reporting.aspx. [Último acceso: 11 11 2016].

[49] Start Bootstrap, «SB Admin - Free Bootstrap Admin Template - Start Bootstrap,» [En línea]. Disponible: https://startbootstrap.com/template-overviews/sb-admin/. [Último acceso: 1 10 2016].

[50] ISO/IEC/IEEE, «Systems and software engineering - Vocabulary - International Standard ISO/IEC/IEEE 24765:2010,» ISO/IEC/IEEE, 15 12 2010. [En línea]. Disponible: https://pascal.computer.org/sev_display/24765-2010.pdf. [Último acceso: 10 12 2016].

[51] Microsoft Developer Network, «Using Data Contracts,» Microsoft, [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/ms733127(v=vs.110).aspx. [Último acceso: 20 12 2016].

ANEXOS

ANEXO A: Código del sistema.

ANEXO B: Manual de usuario del sistema.

ANEXO C: Glosario.

ANEXO C: GLOSARIO

A

Acceso Vascular
: Es aquel que permite el ingreso y salida de sangre del cuerpo humano. Un acceso vascular permite a los doctores realizar los tratamientos de diálisis renal del paciente., 30

AJAX
: (*Asynchronous* JavaScript *and XML*) es una tecnología que permite cargar contenido adicional en la página web realizando peticiones al servidor web, sin necesidad de cargar un nuevo documento HTML., 18

Assert
: Clase estática usada para verificar condiciones en pruebas unitarias usando proposiciones de verdadero o falso., 157

C

Código de cliente
: Se refiere al código que forma parte de una interfaz de usuario el cual es dibujado en un explorador web. El código de cliente puede ser de tipo HTML, JavaScript o CSS., 56

Código de servidor
: Se refiere al código que se ejecuta del lado del servidor., 87

Cookie
: Es una pieza pequeña de información que es generada por el servidor web y enviada al explorador web. El explorador web almacena la *cookie* en el computador desde donde se accede el sitio web., 131

CSS
: (*Cascading Style Sheets*) es un lenguaje de programación para definir la presentación de documentos HTML., 16

D

Diálisis Renal
: Diálisis renal o tratamiento de diálisis renal es el proceso de retirar la sangre del cuerpo humano, y de una manera artificial, simular el proceso que realizan los riñones sobre la misma, para luego devolverla al torrente sanguíneo del paciente., 32

DRY
: (*Don't repeat yourself*) es un principio de arquitectura de *software* que promueve la reutilización del código desarrollado, de tal manera que se evita código repetido en diferentes secciones del sistema., 52

DTD
: (*Document Type Declaration*) es un documento que define la estructura, los elementos y los atributos válidos de un documento XML., 10

E

Ensamblado
: Ensamblado o *assembly* es un archivo o varios archivos que contienen el código y los recursos de una aplicación. Un ensamblado es reutilizable y versionable., 125

Entidad proveedora
: Persona o grupo de personas de la vida real (que probablemente representan a una organización) que posee un agente proveedor (servicio web) el cual realiza acciones en su nombre., 9

Entidad solicitante
: Persona o grupo de personas de la vida real (que probablemente representan a una organización) que requiere hacer uso de un servicio de una entidad proveedora. Por ejemplo, una organización accede a un servicio web de la entidad proveedora para obtener los tratamientos de un paciente., 9

F

G

H

I

J

L

M

W

X