ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

AUTOMATIZACIÓN DE LA RECOLECCIÓN DE DOCUMENTOS DE LOS PROFESORES DEL DETRI PARA ACREDITACIÓN DE CARRERAS

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

DAVID DE JESÚS CABRERA ORDÓÑEZ

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

Quito, julio 2018

DECLARACIÓN

Yo, David de Jesús Cabrera Ordóñez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado 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.

David de Jesús Cabrera Ordóñez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por David de Jesús Cabrera Ordóñez, bajo mi supervisión.

Raúl David Mejía Navarrete, MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

AGRADECIMIENTO

A Dios por todas sus bendiciones y por permitirme cumplir un objetivo más en mi vida.

A mis padres Félix y Anita por todo el apoyo incondicional que me han brindado a lo largo de mi vida.

A todos mis hermanos María, Mariana, Manuel, Juan, Antonio, Luis, y Daniel; quienes me han apoyado en las diferentes etapas de mi vida.

A Giss que es una persona muy especial en mi vida, por todo su cariño y el apoyo que me ha brindado.

A mis grandes amigos y compañeros de aulas, por todo su apoyo y por los buenos momentos que compartimos.

Al M.Sc. David Mejía por sus enseñanzas en cada una de sus clases magistrales, por su guía, apoyo, paciencia y tiempo en la dirección del presente Trabajo de Titulación.

A los profesores de la Escuela Politécnica Nacional que contribuyeron en mi formación profesional.

David Cabrera

DEDICATORIA

Este trabajo está dedicado mi amada familia que son la fuente de motivación y superación en la vida.

Al M.Sc. David Mejía por su guía en el desarrollo del presente Trabajo de Titulación.

A los profesores del DETRI quienes me brindaron información para el desarrollo del presente trabajo, entre ellos al M.Sc. Jorge Carvajal.

David Cabrera

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente Trabajo de Titulación tiene como objetivo el desarrollo de un prototipo de aplicación web que permita automatizar el proceso de recolección de los documentos de los profesores del Departamento de Electrónica, Telecomunicaciones, y Redes de Información (DETRI), para tener la documentación organizada y accesible de forma fácil y rápida, en el momento en que se requiera para el proceso de acreditación de las carreras que soporta el DETRI, lo cual llevará a cabo el Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad del Sistema de Educación Superior (CEAACES).

En el primer capítulo se realiza una descripción de la fundamentación teórica sobre el desarrollo web, de las tecnologías ASP.NET MVC, SQL SERVER y de la metodología de desarrollo ágil Kanban, la cual permite abordar adecuadamente el problema.

En el segundo capítulo se describen los requerimientos de la aplicación web que se obtuvieron aplicando una encuesta a un grupo de profesores del DETRI, al Coordinador de la carrera de Ingeniería en Electrónica y Redes de Información, al Coordinador de la carrera de Ingeniería en Electrónica y Telecomunicaciones, al Jefe del DETRI, a los miembros del Comité de Evaluación Interna (CODEI); y, analizando los documentos del modelo genérico de evaluación de carreras del CEAACES; además, se describe el diseño de la aplicación web a través de diagramas de casos de uso, diagramas de clases, diagrama entidad-relación, diagrama relacional y *mockups* que exponen la apariencia de la aplicación web.

En el tercer capítulo se describe la implementación de la aplicación web, se describen las pruebas de integración, validación y aceptación que se realizaron para garantizar el correcto funcionamiento de la misma.

En el cuarto capítulo se presentan las conclusiones y recomendaciones obtenidas en este Trabajo de Titulación.

En los anexos se presentan los diagramas de casos de uso, el *backlog*, el *script* para generar la base de datos, el código de la aplicación web y el manual de usuario.

PRESENTACIÓN

El Departamento de Electrónica, Telecomunicaciones, y Redes de Información (DETRI) no cuenta con un sistema automatizado que permita recolectar la documentación de sus profesores para el proceso de acreditación de las carreras, donde el Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad del Sistema de Educación Superior (CEAACES) es el organismo evaluador.

Actualmente el proceso de recolección de documentos de los profesores se lo realiza de forma manual, se debe solicitar la información a los profesores y organizarla en sus respectivas carpetas, se realizan informes de la información recopilada y las búsquedas de dicha documentación no es rápida, aun cuando la información se encuentre bien organizada, por tal motivo se propuso el desarrollo de una aplicación web que automatice el proceso de recolección de los documento de los profesores para la acreditación de las carreras que da soporte el DETRI.

La aplicación web cuenta con 5 módulos, un módulo para la gestión y autenticación de usuarios que permite registrar, eliminar y controlar el acceso de los usuarios a la aplicación web de acuerdo al rol asignado; un módulo para la recepción de documentos, el cual permite crear formularios con sus respectivos campos, en donde los profesores tienen que ingresar la información solicitada; un módulo para generar, visualizar e imprimir informes tanto por los profesores como por los administradores; un módulo para generar notificaciones que enviará recordatorios mediante el correos electrónicos a los profesores; y, un módulo para registrar los eventos o acciones realizadas por los usuarios dentro de la aplicación web, podrán ser visualizadas por cada usuario y por el administrador.

CAPÍTULO 1

1. MARCO TEÓRICO

1.1 INTRODUCCIÓN

El Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad del Sistema de Educación Superior (CEAACES) está encargado de realizar el proceso de acreditación de las carreras universitarias para asegurar y garantizar la calidad del sistema educativo de nivel superior. El Departamento de Electrónica, Telecomunicaciones, y Redes de Información (DETRI) no cuenta con un sistema automatizado que permita recolectar la documentación de sus profesores para el proceso de acreditación de carreras. Actualmente el proceso de recolección de documentos de los profesores se lo realiza de forma manual, se debe solicitar la información a los profesores y organizarla en sus respectivas carpetas, se realizan informes de la información recopilada y las búsquedas de dicha documentación no es rápida, aun cuando la información se encuentre bien organizada.

La documentación de los profesores debe permanecer almacenada por un cierto tiempo, además cada semestre se ingresa nueva información. Por otro lado, si se cambia de autoridades será complicado dar continuidad a este proceso, debido a que las nuevas autoridades deberán entender todo el proceso y continuar realizando la recolección de la documentación a mano, si no se realiza la automatización.

La información de los docentes es de mucha importancia para el proceso de acreditación de las carreras, por lo que debería estar bien organizada, siendo su accesibilidad de forma fácil y rápida en cualquier momento que se lo requiera; motivo por el cual se propone un prototipo que automatice el proceso de la recolección de la documentación de los profesores. Cabe recalcar que, si la carrera no acredita, el CEAACES determinará su suspensión, impidiendo a la institución la posibilidad de abrir nuevas promociones de dicha carrera por un periodo de diez años, a partir de la respectiva notificación [1].

1.2 OBJETIVOS

El objetivo general de este Proyecto Integrador es:

Desarrollar un prototipo de aplicación web que permita automatizar el proceso de recolección de documentos de los profesores del DETRI.

Los objetivos específicos de este Proyecto Integrador son:

- Analizar el proceso de recolección de la información de los profesores para la acreditación de carreras.
- Diseñar los módulos y la base de datos necesarios de acuerdo al análisis de los requerimientos.
- Implementar los módulos y la base de datos necesarios para el funcionamiento del prototipo.
- Evaluar el funcionamiento de la aplicación web una vez implementada mediante la realización de pruebas de integración, validación, y aceptación.

1.3 ALCANCE

Se pretende desarrollar un prototipo de aplicación web que permita automatizar el proceso de recolección de documentos de los profesores del DETRI, para contar con la documentación organizada y accesible de forma fácil y rápida, en cualquier momento que se requiera para el proceso de acreditación de las carreras del DETRI, por parte del CEAACES.

La aplicación web contará con 5 módulos, un módulo para la gestión y autenticación de usuarios que se encarga de registrar, eliminar y controlar el acceso de los usuarios a la aplicación web de acuerdo al rol asignado; un módulo para la recepción de documentos el cual permite crear formularios con sus respectivos campos, en donde los profesores tienen que ingresar la información solicitada; un módulo para generar, visualizar e imprimir informes tanto por los profesores como por los administradores; un módulo para generar notificaciones que enviará recordatorios a los profesores mediante mensajes de correo electrónico; y, un módulo para registrar los eventos o acciones realizadas por los usuarios dentro de la aplicación web, podrán ser visualizadas los usuario y por el administrador.

1.4 FUNDAMENTACIÓN TEÓRICA

1.4.1 SISTEMA DISTRIBUIDO

Un sistema distribuido es un: “sistema en el cual los componentes de hardware o software están localizados en computadores en red, el cual se comunica y coordina sus acciones solo con el paso de mensajes” [2]; es decir, se trata de un conjunto de computadoras independientes que cuentan con su respectivo hardware y software, las cuales están físicamente separadas y conectadas a través de una red de comunicaciones, para actuar como un solo sistema; se comunican y coordinan las acciones mediante el paso de mensajes. La principal motivación de implementar y usar sistemas distribuidos es poder compartir recursos informáticos de diferentes tipos, como son los componentes de hardware entre los que se encuentran los discos e impresoras o entidades definidas por software como archivos, bases de datos, audio, video y otros tipos de información.

1.4.1.1 Arquitectura de aplicaciones en N capas

Es importante manejar adecuadamente los conceptos de capas (*Layers*) y niveles (*Tiers*), debido que a menudo se confunden estos términos [3].

Capas: Las capas se encargan de la división lógica de los componentes y sus funcionalidades, sin considerar la ubicación física de estos componentes, pudiendo estar en diferentes servidores o en diferentes lugares.

Niveles: Los Niveles se encargan de la distribución física de los componentes y sus funcionalidades, colocándolos en servidores separados, para ello se debe considerar la topología de la red y las localizaciones remotas. Se utiliza el término “*Tier”* para referirse a patrones de distribución física como “*2-Tier*”, “*3-Tier*” y “*N-Tier*”.

Tanto las capas como los niveles usan nombres comunes para referirse a la presentación, servicios, negocios y datos. En la Figura 1.1 se presenta en la parte a) una arquitectura con los componentes y funcionalidades separados de forma lógica en capas; mientras que en la parte b) se visualiza una arquitectura con los componentes y funcionalidades separados físicamente en niveles.

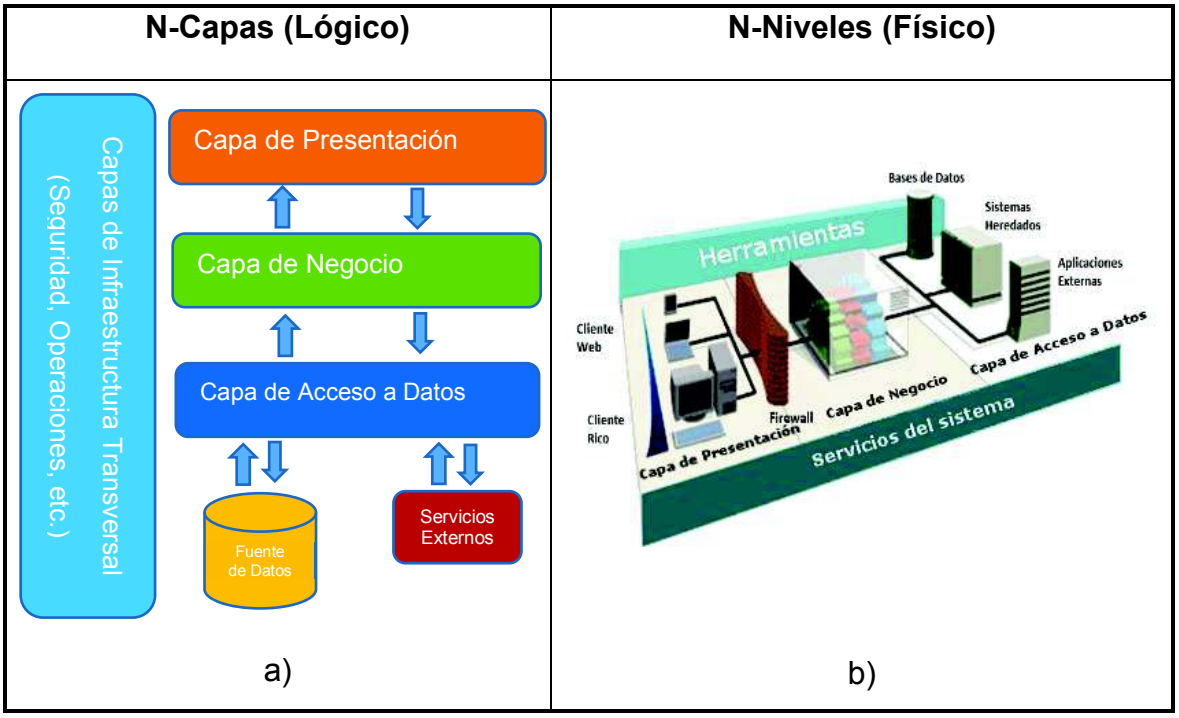


Figura 1.1. Arquitectura de aplicaciones en N-Capas y en N-Niveles [3]

Las aplicaciones con cierta complejidad se deben implementar con una arquitectura de N-Capas; sin embargo, existen aplicaciones que no requieren una separación física en N-Niveles, como es el caso de algunas aplicaciones web [3].

1.4.1.2 Programación en capas

La programación en capas es una técnica de la ingeniería del software que consiste en dividir el código fuente según su funcionalidad principal de una forma ordenada; a su vez, permite dividir el trabajo durante el desarrollo de la aplicación y se facilitan las modificaciones del software porque se trabaja en la capa correspondiente, sin tener que modificar todo el código. Se tienen tres capas principales que son [4], [5]:

Capa de presentación: Se encarga de interactuar con el usuario para presentar o recopilar información, es amigable y sencilla de utilizar. Esta capa se comunica únicamente con la capa de negocio.

Capa de negocio: Se encarga de la lógica de la aplicación para recibir peticiones del usuario y enviar las respectivas respuestas.

Capa de datos: Se encarga de realizar las transacciones de la base de datos o de otros sistemas de información; permite almacenar, recuperar, modificar o eliminar la información.

De acuerdo a la complejidad del sistema se pueden tener otras capas adicionales; a su vez, estas capas se pueden alojar en uno o varios servidores.

1.4.1.3 Aplicación web

Una aplicación web, es un programa informático que no necesita ser instalado; se almacena en un servidor remoto y el cliente accede a través de Internet por medio de un navegador web; para su desarrollo existen diferentes lenguajes de programación como: ASP.NET[1], PHP[2], JAVA[3], entre otros; además, se utiliza HTML[4] para estructurar el contenido, CSS[5] para controlar la apariencia y JavaScript[6] para generar contenidos dinámicos [6].

A menudo se suele confundir el término sitio web con aplicación web; sin embargo, se tratan de plataformas digitales con propósitos específicos que responden a necesidades diferentes, donde los usuarios cuentan con un conjunto distinto de funcionalidades respectivamente y todo dependerá de los productos o servicios que se quiera ofrecer al usuario. Los sitios web son fuentes de información estática, mientras que las aplicaciones web ofrecen un conjunto de acciones, funciones o tareas, con funcionalidades similares a las aplicaciones de escritorio o las aplicaciones para dispositivos móviles; por tal tanto, estas aplicaciones demandan de una lógica más compleja que la requerida para el desarrollo de un sitio web [7].

Las aplicaciones web se suelen utilizar para acceder al correo electrónico, ventas en línea, wikis, servicios de mensajería y otras funciones que dan respuesta a las diferentes necesidades o problemas del cliente. Entre las aplicaciones web más conocidas se encuentran los servicios de Google con Google Maps, Gmail, Docs, entre otros; los servicios de Amazon con la tienda *online*, Amazon audio y video; Microsoft con Office Online el cual incluye versiones de Word, Excel, Outlook, entre otros que cuentan con sus principales funciones para crear, editar y guardar

---

[1] ASP.NET: Es un entorno que proporciona todos los servicios necesarios para desarrollar aplicaciones web.
[2] PHP: Acrónimo recursivo de *Hypertext Preprocessor* (procesador de hipertexto), es un lenguaje de programación del lado del servidor muy utilizado para el desarrollo web.
[3] JAVA: Es un lenguaje de programación de propósito general y orientado a objetos.
[4] HTML (*HyperText Markup Language*): Es un lenguaje de etiquetado que se utiliza para estructurar y presentar el contenido de las páginas web.
[5] CSS (*Cascading Style Sheets*): Es un lenguaje utilizado controlar la apariencia de las páginas web.
[6] JavaScript: Es un lenguaje de programación utilizado para la creación de páginas web dinámicas.

documentos, siendo estas aplicaciones muy similares al Paquete de Microsoft Office de escritorio que requiere de una instalación [8].

1.4.2 ASP.NET MVC

ASP.NET MVC es un *framework*[7] de Microsoft que se utiliza para desarrollar aplicaciones web mediante el modelo general Modelo-Vista-Controlador (MVC), que es un patrón arquitectónico. Su arquitectura permite separar las responsabilidades de una aplicación web en componentes lógicos para manejar aspectos específicos en el desarrollo de las aplicaciones; permitiendo crear proyectos web basados en HTML5[8], CSS y JavaScript.

Las solicitudes del usuario son enrutadas a un controlador que es el responsable de responder a las acciones que solicita el usuario; el controlador es el único que interactúa con el modelo para realizar consultas a la base de datos; además, el controlador es el encargado de elegir la vista para mostrar la información al usuario, proporcionándole los datos del modelo que requiera.

En la Figura 1.2 se puede apreciar que la vista y el controlador dependen del modelo; mientras que el modelo no depende ni de la vista ni del controlador, siendo este uno de los principales beneficios de la separación de capas, de tal forma que el modelo se puede construir y probar independientemente de la presentación visual [9].

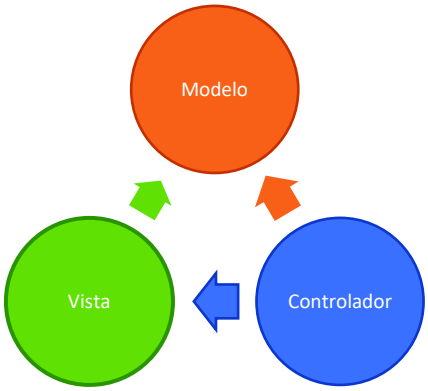


Figura 1.2. Modelo Vista Controlador

[7] *Framework*: Es un conjunto estandarizado de conceptos, prácticas y criterios para resolver nuevos problemas de índole similar.
[8] HTML5: *HyperText Markup Language* versión 5, es la última versión de HTML y cuenta con nuevos elementos, atributos y comportamientos para presentar el contenido de las páginas web.

La delimitación de responsabilidades ayuda a escalar la aplicación en términos de complejidad, porque es más fácil codificar, depurar, actualizar y probar un proyecto en MVC antes que en un solo trabajo que sigue el principio de responsabilidad única.

1.4.2.1 Ciclo de vida de una solicitud HTTP en ASP.NET MVC

El ciclo de vida de una aplicación web consta de una serie de pasos o eventos que se realizan para manejar algún tipo de solicitud o para cambiar el estado de una aplicación; por lo tanto, ASP.NET MVC tiene una secuencia de eventos que ocurren cada vez que se generar una petición HTTP[9], la cual es manejada por la aplicación web como se muestra en la Figura 1.3.

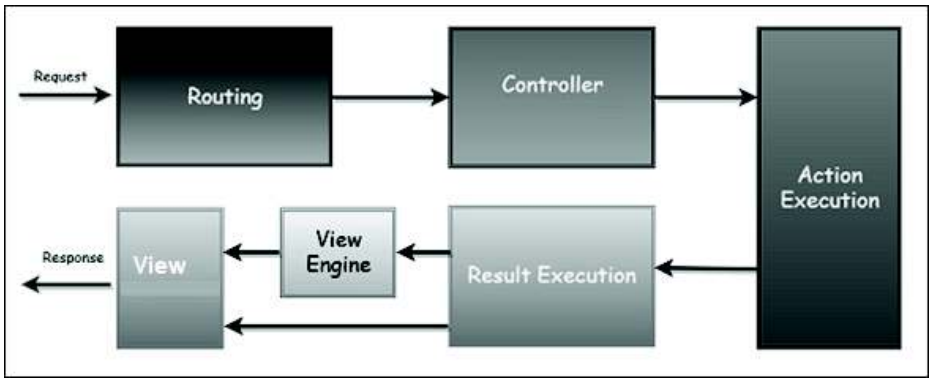


Figura 1.3. Ciclo de vida de una solicitud HTTP con ASP.NET MVC [10]

El ciclo de vida de una solicitud HTTP consta de una secuencia ordenada de eventos que se producen cada vez que la aplicación web recibe y maneja la solicitud HTTP; donde la solicitud pasa por el *Routing* a través del objeto *UrlRoutingModule*, que es un módulo HTTP en donde se analiza la solicitud y se realiza la selección del primer objeto de la ruta que coincida con la solicitud; para seleccionar un controlador (*controller)* que manejará la solicitud mediante la ejecución de una acción definida en un método de dicho controlador (*action*); posteriormente si el resultado corresponde a un tipo de vista, el motor de vistas (*View Engine*) será el responsable de encontrar una vista (*View*) adecuada y presentar la información solicitada, caso contrario el resultado de la acción se ejecutará por su cuenta [10].

[9] HTTP (*Hypertext Transfer Protocol*): Es un protocolo de comunicación que permite la transferencia de hipertexto.

1.4.2.2 Desarrollo de aplicaciones web con ASP.NET

El desarrollo de aplicaciones web con ASP.NET requieren de .NET Framework que contiene las librerías necesarias para su codificación, un IDE[10] para el desarrollo y un servidor IIS[11] para alojar la aplicación web [11].

.Net Framework: Es una plataforma de programación integral para desarrollar aplicaciones de Windows y aplicaciones web, las cuales son interoperables y seguras. Está integrado por el *Common Language Runtime* (CLR[12]) y las bibliotecas de clases de .NET Framework que cuenta con las interfaces, las clases y los tipos de valores que son compatibles con una amplia variedad de tecnologías. Provee un entorno de ejecución administrado que simplifica el desarrollo y la implementación al aportar independencia e interoperabilidad entre diferentes lenguajes como son: Visual Basic, C#, Visual F# y C++; además, tiene la capacidad para interactuar con otros componentes y aplicaciones de .NET Framework independientemente del lenguaje de programación con el que fueron desarrollados [12].

IDE: Microsoft Visual Studio es un Entorno de Desarrollo Integrado, desarrollado por Microsoft; cuenta con un conjunto de herramientas de productividad, servicios en la nube y extensiones para desarrollar un sinnúmero de aplicaciones; tiene soporte de múltiples lenguajes de programación como C++, Visual Basic .NET, C#, Java, Python[13], Ruby[14], PHP; al igual que entornos de desarrollo web, como ASP.NET MVC, Django[15], entre otros; además, cuenta con características para el desarrollo de aplicaciones en la nube de Microsoft Azure[16].

---

[10] IDE (*Integrated Development Environment*): Es un programa informático que cuenta con un editor de código fuente, herramientas de construcción automáticas, un depurador, entre otros; utilizado para facilitar el desarrollo de software.
[11] IIS (*Internet Information Services*): Son un conjunto de servicios para servidores de Microsoft Windows; es comúnmente utilizado como un servidor web.
[12] CLR (*Common Intermediate Language*): Es un entorno de ejecución para los códigos de programas que corren sobre la plataforma de Microsoft .NET; además, proporciona servicios que facilitan el proceso de desarrollo de software.
[13] Python: Es un lenguaje de programación de propósito general; interpretado, orientado a objetos y multiplataforma.
[14] Ruby: Es un lenguaje de programación interpretado, reflexivo y orientado a objetos; su elegante sintaxis se siente natural al leer y fácil de escribir.
[15] Django: Es un *framework* escrito en Phyton para el desarrollo de aplicaciones web.
[16] Microsoft Azure: Es un conjunto de servicios en la nube para crear, implementar y administrar diferentes tipos de aplicaciones.

Servidor IIS: Es un servidor web que cuenta con un conjunto de servicios para Microsoft Windows; soporta múltiples protocolos como: FTP[17], SMTP[18], NNTP[19], HTTP, HTTPS[20], HTTP/2[21] para publicar páginas web de forma local o remota. Tiene la capacidad para procesar diferentes tipos de páginas como ASP.NET, PHP, Perl[22], entre otros [13].

1.4.2.3 Modelo

El modelo se encarga de la representación de los datos que maneja la aplicación mediante objetos; es decir, se encarga de la lógica del negocio y cuenta con los mecanismos de persistencia. Las responsabilidades del modelo son: modelar objetos, recuperar, crear, actualizar, mantener el estado del objeto y el almacenamiento persistente en la base de datos. Para construir el modelo se utiliza Entity Framework.

ASP.NET MVC provee de algunas herramientas y características para generar los controladores y las vistas de las aplicaciones a partir de la definición del modelo; por lo que el método para construir las aplicaciones a partir del modelo se denomina *scaffolding*[23] [14].

1.4.2.3.1 Entity Framework

Entity Framework (EF) es un ORM[24] de .NET Framework que cuenta con un conjunto de tecnologías de ADO.NET utilizadas para desarrollar aplicaciones de software que están orientadas a datos; su objetivo es reducir la cantidad de código

---

[17] FTP (*File Transfer Protocol*): Es un protocolo de red utilizado para la transferencia de archivos.
[18] SMTP *(Simple Mail Transfer Protocol*): Es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico.
[19] NNTP (*Network News Transport Protocol*): Es un protocolo inicialmente creado para la lectura y publicación de artículos de noticias en Usenet (Red de usuarios).
[20] HTTPS (*HyperText Transfer Protocol Secure*): Es un protocolo seguro de transferencia de hipertexto.
[21] HTTP/2 (*HyperText Transfer Protocol* versión 2): Es un protocolo que presenta mejoras con respecto al protocolo HTTP que están relacionadas con el empaquetado de datos y el transporte de la información.
[22] Perl (*Practical Extracting and Reporting Languaje*): Es un lenguaje de programación de propósito general utilizado para múltiples propósitos como la administración de sistemas, el desarrollo web, la programación en red, entre otros.
[23] *Scaffolding*: Es una técnica que permite a un desarrollador definir y crear aplicaciones básicas automáticamente a partir de un modelo permitiendo crear, visualizar, actualizar y borrar información.
[24] ORM (*Object Relational Mapping*): Es una técnica para la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

en el desarrollo y el mantenimiento que requieren las aplicaciones orientadas a datos. Los datos son representados en forma de objetos y propiedades específicas del dominio sin tener que preocuparse de las tablas y columnas de la base de datos subyacente donde se almacena la información. Las aplicaciones de Entity Framework se ejecutarán en los equipos que tengan instalado .NET Framework a partir de la versión 3.5 SP1.

EF se divide en tres modelos: el modelo de dominio que corresponde a las entidades y las relaciones del sistema; el modelo lógico de una base de datos relacional que se encarga de normalizar las entidades y las relaciones de las tablas con restricciones de las claves externas o foráneas; y, el modelo físico que corresponde a las capacidades de un motor de datos determinado que especifica los detalles del almacenamiento en forma de particiones e índices.

EF cuenta con un asistente para el diseño y la generación del código *Entity Data Model*; que puede construirse a partir de una base de datos existente o a partir de un modelo vacío con la ayuda del asistente de EF que se denomina *Code First*, el cual crea el modelo a partir de diferentes clases que representan a una tabla en la base de datos o con la ayuda de *EF Designer* que permite crear el modelo gráficamente mediante un diagrama relacional [15].

El modelo cuenta con diferentes elementos y características, algunas de ellas se detallan a continuación:

1.4.2.3.2 Database Context

Es la clase principal del modelo que se encarga de coordinar la funcionalidad de Entity Framework; cuida la integridad del esquema del objeto relacional y representa un contenedor de entidades en memoria. Con la ayuda de otras clases e interfaces, es capaz de administrar la identidad de una entidad, el estado, los valores originales y los valores nuevos de las propiedades de la entidad; además, realiza el seguimiento de los cambios efectuados de cada entidad en la memoria caché. Las entidades se pueden asociar o desasociar de un contexto, de tal forma que solo las entidades asociadas al contexto recibirán un seguimiento y administración [16].

1.4.2.3.3 Estados de una entidad

La clase de contexto del modelo debe conocer el estado de cada una de las entidades del modelo para guardar los cambios en el origen de los datos. Se tienen los siguientes estados [16]:

Added: Una nueva entidad será agregado al contexto de entidades. Al utilizar el método `SaveChanges`, se cambia el estado del objeto a `Unchanged` y se ejecutará una sentencia `insert` sobre la base de datos.

Deleted: La entidad será eliminada del contexto de entidades. Al utilizar el método `SaveChanges`, se cambia el estado del objeto a `Unchanged` y se ejecutará una sentencia `delete` sobre la base de datos.

Detached: La entidad no se tomará en cuenta a la hora de aplicar cambios.

Modified: Modifica alguna de las propiedades escalares de la entidad, por lo que el estado de las propiedades modificadas cambia a `Modified` llamando al método `DetectChanges`. Al utilizar el método `SaveChanges`, se cambia el estado del objeto a `Unchanged` y se ejecutará la sentencia `update` sobre la base de datos para guardar los cambios.

Unchanged: La entidad no ha cambiado desde que se recuperó de la fuente de datos.

1.4.2.3.4 Data annotations

Permite llevar a cabo validaciones de los datos ingresados por el usuario y de todos los datos que se van a almacenar en la base de datos; se utilizan de acuerdo a los requerimientos de la aplicación, para lo cual se proporcionan clases con atributos que se utilizan para definir los metadatos que realizan los controles de los datos en ASP.NET MVC y ASP.NET. Algunos atributos utilizados para las anotaciones de datos son [17]:

Required: Especifica que el valor debe ser ingresado obligatoriamente y no puede contener un valor `null` o vacío.

Range: Especifica un rango de valores para un tipo de dato específico.

StringLength: Especifica la longitud de una cadena de texto, el cual se puede utilizar en conjunto con `MinimunLength` y `MaxLength` para especificar cadenas de longitud mínima y máxima.

RegularExpression: Se utiliza para validar expresiones regulares, que son formatos específicos definidos dentro de las cadenas de caracteres, como son las fechas, las direcciones de correo electrónico, entre otros.

DataType: Se utiliza para especificar un tipo de dato que está asociado al campo de datos o al de un parámetro. Algunos de estos son: `currency` (valor de moneda), `custom` (tipo de datos personalizado), `date` (valor de fecha), `dateTime` (tiempo expresado como una fecha y hora) `duration` (tiempo de la existencia de un objeto), `emailAddress` (dirección de correo electrónico), `html` (archivo HTML), `imageUrl` (URL de una imagen), `multilineText` (texto multilínea), `password` (valor de contraseña), `phoneNumber` (número de teléfono), `postalCode` (código postal), `text` (texto que se muestra), `Time` (valor de hora), `upload` (tipo de datos de carga de archivo) y `url` (dirección URL).

CustomValidation: Se utiliza para especificar validaciones personalizadas a los atributos de la entidad.

En la Figura 1.4 se presenta una entidad definida por una clase `Persona` que cuenta con anotaciones de datos; en la línea 12 se especifica que el valor nombre es obligatorio y deberá tener una longitud máxima de 50 caracteres; y, en la línea 16 se especifica que el valor de la edad deberá estar entre 0 y 100.

```
10      public class Persona
11      {
12          [Required, StringLength(50)]
            0 referencias
13          public string Nombre { get; set; }
14
15          [Range(0, 100)]
            0 referencias
16          public int Edad { get; set; }
17      }
```

Figura 1.4. Ejemplo de anotaciones de datos

1.4.2.4 Vistas

La vista o interfaz de usuario, se encarga de presentar la información al cliente y a su vez cuentan con mecanismos de interacción. Su responsabilidad es presentar la información al usuario, la cual es creada desde el modelo de datos [14].

Las vistas están compuestas por sintaxis Razor, que se encargan de generar HTML; además en las vistas se puede agregar etiquetas HTML para estructurar la página web, hojas de estilo para mejorar la apariencia del contenido y JQuery para generar funcionalidades dinámicas en el lado del cliente; algunas características mencionadas sobre las vistas se detallan a continuación.

1.4.2.4.1 Layout

Es una plantilla que contiene el código HTML con todos los componentes que se repiten en varias páginas web, como es el caso del encabezado, el pie de página, menús de navegación, entre otros. Dentro del *layout* se encontrará la sentencia `RenderBody()`, que es el encargado de renderizar todas las vistas que utilicen este archivo de *layout* mediante la sentencia `Layout =_layout`; un ejemplo de la renderización de vistas se puede visualizar gráficamente en la Figura 1.5 [18].

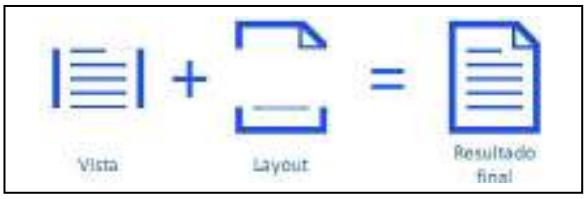


Figura 1.5. Renderización de vistas

1.4.2.4.2 HTML5

HTML5 es la última versión de HTML (*Hyper Text Markup Language*), se trata de un lenguaje de etiquetado que se utiliza para estructurar y presentar el contenido de las páginas web; además, el desarrollo de este lenguaje de marcado es regulado por el Consorcio W3C[25]. Provee tres características importantes que son: estructura, estilo y funcionalidad; cuenta con elementos, atributos y comportamientos para la reproducción interna de vídeos, audio y juegos. [19], [20].

[25] W3C: Es un consorcio internacional que genera recomendaciones y estándares para asegurar el crecimiento de la *World Wide Web* a largo plazo.

En la Figura 1.6 se presentan algunas etiquetas de HTML5 que permiten organizar el contenido de una página web.

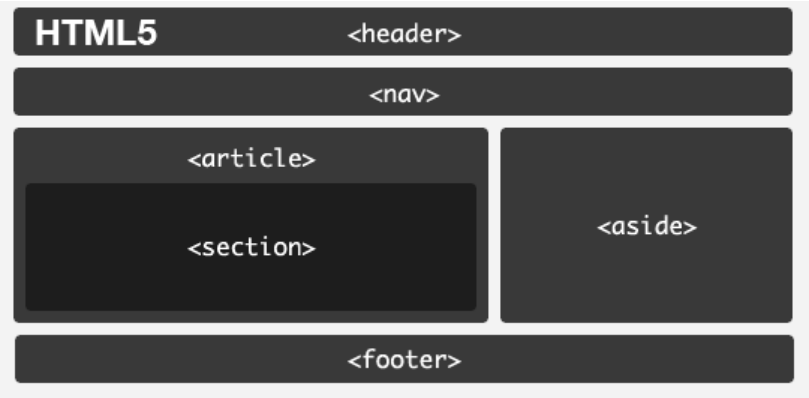


Figura 1.6. Etiquetas de HTML5 [21]

En la Tabla 1.1 se detallan las características de algunas etiquetas utilizadas en HTML5 [22].

Tabla 1.1. Etiquetas de HTML5

ELEMENTOS	CARACTERÍSTICAS
<!doctype html>	La página web está escrita bajo el estándar HTML 5.
<html>	Representa la raíz de la página web.
<head>	Contiene una colección de metadatos sobre la página, se incluye referencias a *scripts* y hojas de estilo.
<title>	Contiene el título de la página que se muestra en la barra de título del navegador o en una pestaña.
<body>	Contenedor donde se escribe el contenido de la página web, es único dentro de un documento HTML.
<header>	Define la cabecera de una página o sección, generalmente contiene el título, un logotipo y una tabla de navegación de contenidos.
<nav>	Es una sección que contiene los menús de navegación.
	Se utiliza para especificar un bloque de contenido o una sección de la página, de esta forma se puede aplicar diferentes estilos o realizar alguna acción específica.
<article>	Para definir contenido autónomo que puede existir independientemente del resto del contenido.
<section>	Define secciones en un documento como capítulos, encabezados, pies de página, entre otros.
<h1>, …,<h6>	Son elementos de cabecera para describir brevemente el tema de una sección.
<footer>	Especifica el pie de una página o sección, generalmente contiene un mensaje donde se indican los derechos del autor, enlaces para visualizar cierta información legal o para incluir direcciones con información de retroalimentación.
<address>	Se utiliza para ingresar la información de contacto.
<!-- -->	Se utiliza para escribir comentarios en el documento HTML, todo lo que se encuentra dentro de estos caracteres no se visualizará en el navegador web.

1.4.2.4.3 CSS3

CSS (*Cascading Style Sheets*) se encarga de controlar la apariencia de la página web; define reglas para presentar la información en diferentes dispositivos, como es el caso del tamaño de texto, los tipos de fuentes, los colores, el espaciado entre textos, los recuadros y el lugar en donde se disponen los textos e imágenes dentro de la página web. CSS3 es la versión actual, tiene opciones para sombreado, redondeado, funciones avanzadas de movimiento y transformación [23], [24].

Los elementos de las sentencias que se escriben en el archivo CSS para definir las reglas son:

Selector: Es el nombre del elemento HTML que se selecciona de la hoja de estilos al momento aplicar un estilo en concreto en la página web.

Atributos: Son las reglas que se aplicarán a las etiquetas HTML, mediante diferentes características que tiene el lenguaje CSS para cambiar el fondo (*background*), el tipo de letra (*font*), entre otros.

Valores: CSS tiene valores concretos que definen los comportamientos, que son especificados en los atributos; algunos de los valores pueden ser el color, el tamaño, entre otros.

CSS permite incluir comentarios con los caracteres `/*  */`, se utiliza para documentar el código y entenderlo; a su vez, los comentarios son ignorados por el navegador.

En la Figura 1.7 se presenta un ejemplo de código de estilo aplicado a la etiqueta `h1`; en la parte a) se define un color de texto y un color de fondo para esta etiqueta; mientras que en la parte b) se presenta el resultado obtenido donde el texto de la etiqueta `h1` tendrá un color rojo con un fondo verde.

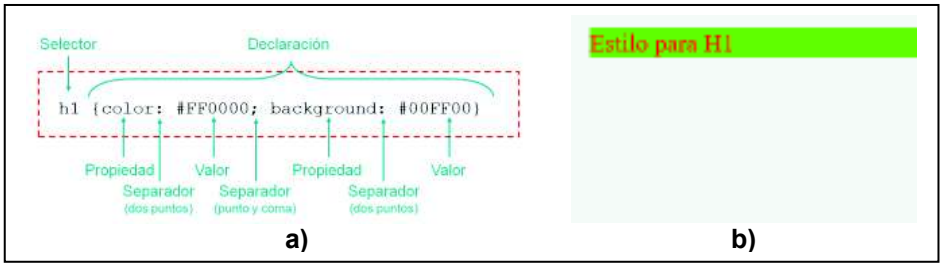


Figura 1.7. Ejemplo de CSS [23]

1.4.2.4.4 Razor

Es un motor de vistas (*View Engine*) o un motor de plantillas que es utilizado para incrustar el código HTML del servidor en las vistas de una forma limpia, ligera y simple. El servidor ejecuta el código que está marcado con la sintaxis de Razor; luego se envía la página resultante al explorador, de tal forma que el código de Razor se ejecuta antes que la página web se envíe al explorador. Las vistas que tienen código Razor se deben guardar con la extensión .CSHTML o .VBHTML.

Razor proporciona la sintaxis simplificada para minimizar la cantidad de código y caracteres adicionales en las vistas, las reglas que se definen en Razor se presentan a continuación [25]:

1. Las sintaxis en Razor inician con `@` y se utiliza para escribir expresiones o bloques de código.
   Expresión: se escriben de la siguiente forma: `@expresión`, son utilizadas para escribir palabras reservadas, variables o funciones.
   Bloque de código: Se escribe de la siguiente forma: `@{código}`, el código se tiene que escribir dentro de las llaves.
2. Las sentencias de código terminan con punto y coma (`;`).
3. Las variables son declaradas con la palabra clave `var`.
4. Las cadenas de caracteres (`string`) se escriben entre comillas.
5. *Correos electrónicos:* Razor identifica cuando se trata de una dirección de correo electrónico sin confundirse con las sintaxis utilizadas en las expresiones.
6. *Escape a la arroba:* Para indicarle a Razor que se trata de una simple arroba, de tal forma que no realice nada específico, se debe escribir doble arroba. Ejemplo: `@@media screen`.
7. *Comentarios:* Los comentarios se escriben entre arrobas de la siguiente forma: `@* comentario *@`
8. *Estructuras de control:* Las sintaxis de las estructuras de control son parecidas a las utilizadas en .NET, con la particularidad que se debe agregar `@` al inicio de la estructura de control; cabe señalar que el uso de llaves es obligatorio. En la Figura 1.8 se muestra un ejemplo de una estructura de

control; para lo cual se definirá una variable llamada `mensaje` (línea 10), a través de la estructura de control `if` (línea 12) la variable tomará uno de los valores definidos que son administrador del sistema o gestor (líneas 13-19).

```
8   @{
9
10      var mensaje = "";
11
12      if (User.Identity.GetType().ToString() == "admin")
13      {
14          @mensaje = "Administrador del sistema";
15      }
16      else
17      {
18          @mensaje = "Gestor";
19      }
20
21  }
```

Figura 1.8. Ejemplo de una estructura de control

1.4.2.4.5 Helpers

Son componentes reutilizables de las vistas que incluyen código (métodos) para generar el etiquetado HTML de forma simple y parametrizada, evitando repetir el código al momento de crear etiquetas, formularios, enlaces, entre otros. Los principales h*elpers* se presentan a continuación, y se escriben de la siguiente forma: `@Html.NombreDelHelper`

- `Label`: Inserta una etiqueta.
- `Hidden`: Inserta un atributo que contiene información que no se visualiza, ni se puede modificar en la página web.
- `Display`: Inserta un texto.
- `BeginForm`: Inserta un formulario web.
- `ActionLink`: Inserta un enlace.
- `TextArea`: Inserta un cuadro de texto con múltiples líneas.
- `TextBox`: Inserta un cuadro de texto.
- `DropDownList`: Inserta una lista de opciones.
- `CheckBox`: Inserta una casilla de verificación.
- `RadioButton`: Inserta un botón de selección.
- `ListBox`: Inserta un menú de opciones.

En Figura 1.9 se presenta la codificación del *helper* `Label`, en la parte a) se muestra la sintaxis usada en la vista; en la parte b) se muestra el HTML generado por el *helper*; y, en la parte c) se presenta lo que se visualizará en el navegador web.

a)
```
@Html.Label("Nombre", "Nombre:")
```
b)
```
<label for="Nombre">Nombre:</label>
```
c)
```
Nombre:
```

Figura 1.9. Ejemplo de un *helper*

Se pueden utilizar *helpers* personalizados de acuerdo a las necesidades, pero se tienen que implementar; para lo cual, se crea un archivo con la extensión .CSHTML dentro de la carpeta `App_Code`, en este archivo se realizará la implementación del *helper* y se utilizará en la vista escribiendo de la siguiente forma: `estruct@MyHelpers.[nombre_archivo_cshtml](atributos)`, donde el *helper* puede tener uno o varios atributos.

Los *helpers* también se utilizan para acceder a la información de un modelo, siempre y cuando en la vista se incluya la referencia del modelo de la siguiente forma: `@model NombreProyecto.NombreModelo`; además, se utilizan las operaciones *lambda* para acceder a la información de las propiedades del modelo: `m=>m.propiedad`. En la Figura 1.10 se presentan varios *helpers* que son utilizados en los formularios de las vistas: en la parte a) se presenta el *helper* `LabelFor` para mostrar una etiqueta de la propiedad título; en la parte b) se presenta el *helper* `TextBoxFor` que mostrará un cuadro de texto con el valor de la propiedad título; y, finalmente se presenta el *helper* `ValidationMessageFor` que se encarga de mostrar un mensaje de validación de la propiedad título que está especificada en la entidad del modelo.

a)
```
@Html.LabelFor(l => l.Titulo)
```
b)
```
@Html.TextBoxFor(l => l.Titulo)
```
c)
```
@Html.ValidationMessageFor(l => l.Titulo)
```

Figura 1.10. *Helpers* con el modelo

ASP.NET MCV también cuenta con *helpers* que muestran mensajes de error en los formularios, para lo cual se utiliza la propiedad `AddModelError` dentro del controlador y se utiliza `ModelState` para validar la información recibida de un formulario; cuando el campo de la propiedad `AddModelError` se deja vacío, por lo que se lanzará un error a nivel del modelo. Para que se muestren los mensajes de error se deben incluir las siguientes sentencias dentro del formulario: `ValidationSummary()`, el cual se encargará de mostrar una lista no ordenada con todos los errores que se producen al validar el formulario; para validar cada uno de los elementos del formulario se tiene que escribir la siguiente sentencia: `ValidationMessage(s:[Nombre del campo], o:[Mensaje de error])`, donde el mensaje de error se mostrará a la derecha del control [26].

1.4.2.4.6 JQuery

JQuery es un conjunto de librerías JavaScript que simplifica la forma de interactuar con los documentos HTML mediante la manipulación de los elementos del DOM (*Document Object Model*). Se encarga de ejecutar eventos, modificar estilos, realizar animaciones e interacciones. JQuery es de código abierto y se basa en una Licencia MIT[26] [27].

En la Figura 1.11 se presenta un pequeño ejemplo de código JQuery, en muchos casos esta es la estructura básica para comenzar a escribir las sentencias de JQuery, de tal forma que todo lo que se encuentra dentro del código que se presentase, se ejecutará una vez que se cargue la página en el navegador.

```
$(document).ready(function() {

 //Aquí se escribe el código

});
```

Figura 1.11. Código de JQuery

[26] Licencia MIT (*Massachusetts Institute of Technology*): Es una licencia de software libre permisiva, que impone muy pocas limitaciones en la reutilización para licenciar software libre como software propietario; lo único que exige es que los derechos de autor sean incluidos en todas las copias.

El DOM es un API[27] de programación para documentos que está basada en una estructura jerárquica de objetos, que genera el navegador web en el momento en que se carga un documento HTML; está definido y administrado por el W3C. El DOM define la estructura lógica de los documentos HTML, el modo en que se accede y la forma en la que se manipula; permite construir documentos HTML, navegar por su estructura, añadir, modificar, o eliminar elementos y contenido [28].

En la Figura 1.12 se presenta un ejemplo de la estructura lógica del DOM, que es muy parecida a un árbol en el cuál los elementos se ramifican desde el tronco principal (`<html></html>`). Las dos ramas más grandes son el `<head>` y el `<body>`, luego las ramas se multiplican y se hacen más delgadas a medida que se introducen más elementos como los `<div>`, `<table>`, el texto de los encabezados y de los párrafos [29].

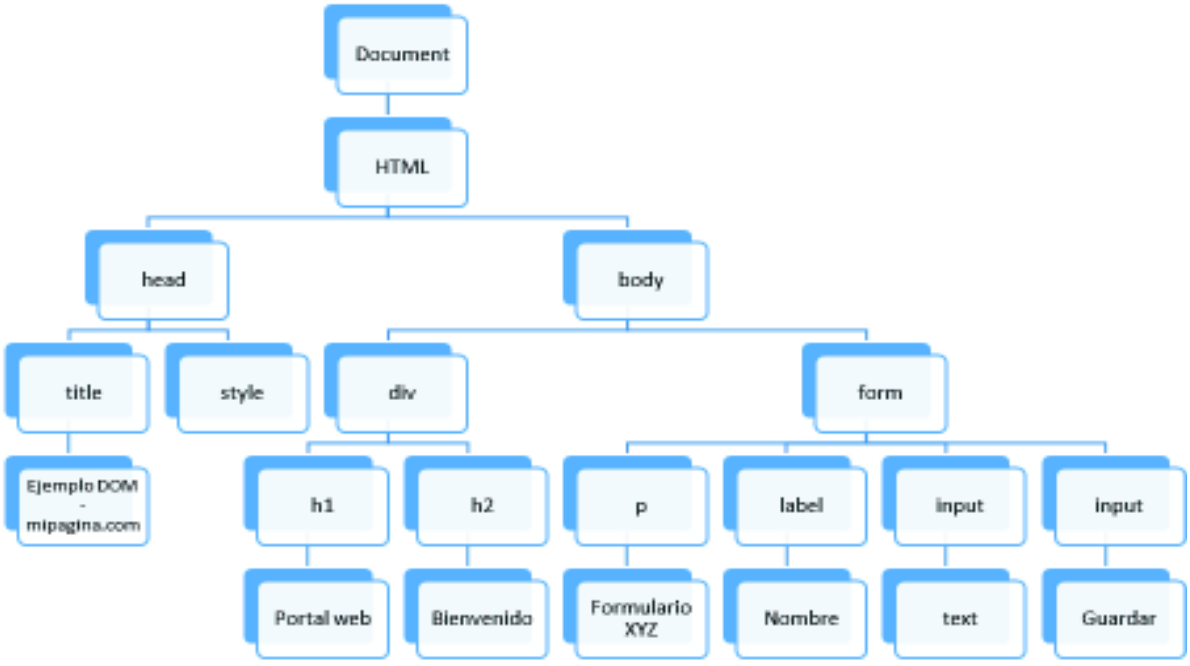


Figura 1.12. Estructura del DOM

1.4.2.4.7 Bootstrap

Bootstrap es un *framework* que fue desarrollado y liberado por Twitter con el objetivo de facilitar el diseño web mediante librerías CSS que incluyen botones,

[27] API (*Application Programming Interface*): Es un conjunto de reglas y especificaciones que provee acceso a funciones de un determinado software; de tal forma que se puede hacer uso de funciones ya existentes en otro software o en la infraestructura de otras plataformas.

tipografías, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web; además, permite crear de forma sencilla páginas web con contenido adaptable a cualquier dispositivo y tamaño de pantalla. Este *framework* fue liberado por Twitter bajo la licencia MIT en el año 2011, por lo que se puede utilizar de forma gratuita y sin restricciones [30].

1.4.2.5 Controlador

El controlador es una clase con un grupo de funciones similares que se denominan métodos de acción; los cuales son utilizados para responder a las solicitudes HTTP; además, actúa como un intermediario entre el modelo y la vista, encargándose de decidir a qué modelo debe consultar para obtener la información solicitada; también se encarga adaptar la información solicitada a la respectiva vista.

La responsabilidad del controlador es: manejar la lógica de la aplicación web, manejar las solicitudes del usuario, trabajar con el modelo y seleccionar la vista para representar la interfaz del usuario [14].

Existen dos tipos de controladores que son: los controladores sincrónicos y los controladores asincrónicos.

1.4.2.5.1 Tipos de controladores

Controlador sincrónico: Normalmente los flujos de las aplicaciones web eran sincrónicos, es decir, al realizar una petición al servidor, se esperaba sin liberar el hilo del *pool* de hilos hasta obtener la respuesta, esto ocasionaba que no se puedan atender otras peticiones mientras no se termine de procesar la primera petición.

Controlador asíncrono: Permite escribir métodos con acciones asincrónicas que se utilizan para liberar el hilo de una solicitud cuando el procesamiento se tarda y de esta forma se pueda atender a otras solicitudes evitando bloqueos; una vez finalizado el procesamiento anterior se le asignará un hilo para dar respuesta a dicha solicitud. El controlador genera un hilo del *pool* de hilos para atender las peticiones, mientras que el servidor web IIS mantiene un grupo de hilos en lugar de crear nuevos hilos para responder cada solicitud; por lo tanto, se consigue beneficios de rendimiento. Este tipo de controlador es ideal para aplicaciones que tengan mucha concurrencia de usuarios [31].

1.4.2.5.2 Filtros de acción

Es un atributo que puede asociarse a una acción o controlador para modificar la forma en la que se ejecuta una acción [32].

En la Tabla 1.2 se detallan algunos filtros de acción que actúan en los controladores.

Tabla 1.2. Filtros de Acción

Nombre	Descripción
`Authorize`	Restringe el acceso a la acción de un controlador, permitiendo el acceso únicamente a los roles o usuarios especificados.
`HandleError`	Especifica una vista que se mostrará en el caso que se produzca una excepción no controlada.
`OutputCache`	Almacena en la memoria caché una acción de un controlador o un controlador completo.
`ValidateAntiForgeryToken`	Evita la falsificación de una petición (*pishing*[28]).
`ValidateInput`	Desactiva las validaciones de solicitudes, pero pone en peligro la seguridad de la aplicación.

1.4.2.5.3 Tipos de acción

Los métodos de acción utilizan diferentes tipos de acción para devolver los resultados a la vista, la clase `ActionResult` es la base de todos los resultados de la acción. En la Tabla 1.3 se presenta una lista con los diferentes tipos de resultados de acción y su comportamiento [33].

[28] *Pishing*: Es uno de los métodos más utilizados de ingeniería social por delincuentes cibernéticos para estafar y obtener información confidencial de forma fraudulenta.

Tabla 1.3. Tipos de Acción

Tipo de acción	Método	Descripción
`ContentResult`	`Content()`	Devuelve una cadena de texto, puede ser tanto de texto plano como de código HTML.

`FileContentResult`
`FilePathResult`
`FileStreamResult`
	`File()`	Devuelve el contenido de un archivo, puede devolver desde un archivo físico o un objeto *stream*.

`EmptyResult`	`EmptyResult()`	No devuelve nada.
`JavaScriptResult`	`JavaScript()`	Devuelve el *script* para su ejecución del lado del cliente, también servirá para intercambiar datos entre el servidor y el cliente, formateando los datos adecuadamente.
`JsonResult`	`Json()`	Devuelve datos con formato JSON[29].
`RedirectResult`	`Redirect()`	Redirecciona la URL especificada.
`HttpUnauthorizedResult`		Devuelve un error de autorización (403).
`RedirectToRouteResult`	`RedirectToRoute()` `RedirectToAction()`	Direcciona a otra ruta/acción del controlador.
`ResultView`	`View()`	Devuelve una vista.
`PartialView`	`PartialView()`	Devuelve una vista parcial.

1.4.2.6 Estructura de la aplicación web

Cuando se crea un proyecto en Visual Studio, automáticamente se generan carpetas y archivos del proyecto. En la Tabla 1.4 se muestran los principales archivos de configuración de la aplicación web; mientras que en la Tabla 1.5 se presentan los directorios del proyecto [14].

[29] JSON (*JavaScript Object Notation*): Es un formato ligero para el intercambio de datos y nació como una alternativa al XML.

Tabla 1.4. Archivos de configuración del proyecto

`Startup.cs`	Es una clase que permite iniciar la aplicación web.
`Global.asax`	Es una clase utilizada para declarar y manejar eventos; objetos a nivel de aplicación; el inicio y fin de la sesión; y, peticiones web, entre otros.
`web.config`	Archivo XML que posee toda la configuración de la aplicación web.

Tabla 1.5. Directorios del proyecto

`/Content`	Almacena los recursos de la aplicación como imágenes, CSS, entre otros.
`/App_Start`	Contiene los archivos de código que se ejecutan para realizar todas las configuraciones necesarias al inicio la aplicación.
`/Controllers`	Permite almacenar los controladores del proyecto.
`/Models`	Permite almacenar las clases relacionadas a la capa de datos.
`/Views`	Permite almacenar las vistas y la plantilla (*layout*) del proyecto.
`/App_Data`	Para almacenar archivos de datos o de bases de datos.
`/Filters`	Contiene las clases que permite incluir comportamientos antes y después de las acciones de los controladores.

1.4.2.7 Contenedores de datos

ASP.NET MVC cuenta con tres métodos principales para enviar la información entre controladores y vistas; cabe recalcar que el controlador es quien accede al modelo para obtener los datos, a continuación se detallan los métodos para enviar datos [31]:

ViewData: Permite pasar información entre un controlador y una vista, solo sobrevive durante una solicitud; además, tiene dos puntos débiles que se deben considerar:

- Las claves que se utilizan son cadenas de texto `ViewData["clave"]`, por lo que si se escribe mal la clave, el compilador no podrá mostrar un error en tiempo de compilación[30].
- Siempre devuelve un objeto, por lo tanto, se debe hacer *casting* para obtener el tipo real de lo que se almacenó.

ViewBag: Usa propiedades dinámicas en vez de utilizar cadenas de texto para especificar las claves, su funcionalidad es semejante a `ViewData`, pero no es necesario realizar *casting*.

TempData: Almacena objetos de corta duración, se utiliza una cadena de texto como clave y un objeto como valor; permite pasar información de controlador a controlador.

Sesión: Se almacena las variables de sesión en memoria haciendo que estos datos sean persisten entre solicitudes hasta que la sesión se caduque; en aplicaciones pequeñas es una solución efectiva y razonable.

En la Tabla 1.6 se realiza una comparación en la que se pasa información entre controladores y vistas, de tal forma que se pueda conocer entre que elementos se puede mantener una persistencia de datos; por lo tanto, se realiza la comparación con los siguientes contenedores de ASP.NET MVC: `ViewData`, `ViewBag`, `TempData` y `Sesion`.

Tabla 1.6. Persistencia de datos

MANTIENE DATOS ENTRE	**ViewData/ViewBag**	**TempData** (solicitudes simples)	**Sesion**
Controlador a Controlador	NO	SI	SI
Controlador a Vista	SI	SI	SI
Vista a Controlador	NO	NO	SI

[30] Tiempo de compilación: Se refiere al intervalo de tiempo en el que un compilador compila el código escrito en un determinado lenguaje de programación a una forma de código ejecutable por una máquina.

1.4.2.8 Tabla de rutas

Las rutas son modelos de direcciones URL, donde se puede especificar marcadores de posición para pasar datos de variables al controlador sin necesidad de utilizar cadenas de consulta; ASP.NET MVC utiliza la ruta por defecto: `http://host/{controlador}/{acción}/{id}`.

La tabla de rutas se utiliza para asociar las URL de la aplicación web con sus correspondientes controladores y los métodos de acción que se encuentran dentro de los controladores; por lo que se define la tabla de rutas en un archivo dentro de la carpeta `App_Start`, llamado `RouteConfig.cs`. Sigue el patrón `{controller}/{action}/{id}` que se define para ejecutar una acción de un controlador; por lo que se debe especificar el controlador, la acción y por último el identificador de los parámetros del método de acción, donde el parámetro por defecto es `{id}`. Para agregar una nueva ruta se define una nueva sobrecarga del método `MapRoute` que recibe un identificador de la ruta, las URL que asocia a esta ruta y los parámetros del método de acción [34].

En la Figura 1.13 se muestra un ejemplo en la que se agrega una nueva ruta a la tabla de rutas denominada `Ventas`; por lo que se inicia realizando una nueva sobrecarga del método `MapRoute` (líneas 24-33); se especifica el nombre de la ruta (línea 25); se especifica la URL con los parámetros (línea 27); se especifica la ruta por defecto que mostrará la página de *index*; y, finalmente se agregan las restricciones, para este ejemplo el `id` puede estar únicamente entre los valores de 1 a 5.

```
22          //url: http://dominio/Ventas/x/y
23
24          routes.MapRoute(
25              name: "Ventas", // Nombre de la ruta
26
27              url: "{controller}/{action}/{id}", // URL con parámetros
28
29              defaults: new { controller = "Ventas", action = "Index", id = UrlParameter.Optional }
30              // Parametros por defecto
31
32              constraints: new { id = @"\d{1,5}" } //Restricciones
33          );
```

Figura 1.13. Nueva ruta en la tabla de rutas

1.4.2.9 Expresiones Lambda

Son funciones anónimas (función literal, expresión Lambda) para crear delegados[31] o expresiones de tipo árbol. Con el uso de expresiones Lambda se puede escribir funciones locales que se pueden pasar como argumentos o devolver como el valor de las llamadas de función. Las expresiones de consulta en las que se utiliza LINQ[32], pueden ser escritas con expresiones Lambda.

Para definir una expresión lambda, se especifican los parámetros de entrada (si existen) en el lado izquierdo del operador lambda, seguido de `=>`, y del lado derecho se coloca el bloque de expresión o de declaración. Como ejemplo se tiene: `@Html.TextBoxFor(x=>x.Nombre)`, donde el parámetro con la sintaxis `x=>x.Nombre` es una expresión lambda. Cabe señalar que las expresiones lambda son evaluadas en el tiempo de compilación y no en el tiempo de ejecución; por lo que, si hay errores en el nombre de la propiedad del modelo, el código no compilará.

1.4.2.10 Optimización de la aplicación web

ASP.NET MVC cuenta con la librería `System.Web.Optimization` que tiene métodos para optimizar los elementos de la aplicación web; permitiendo agilizar la carga, la renderización y disminuir el tiempo de respuesta de la aplicación web. Se tiene dos técnicas que usa ASP.NET que son [35]:

Bundle: Se encarga de empaquetar múltiples archivos JavaScript o CSS en un solo objeto en tiempo de ejecución, con lo cual se reduce el número de solicitudes al servidor; usualmente las páginas web necesitan de varios archivos CSS y JavaScript para su correcto funcionamiento, por lo que se realiza una petición al servidor por cada archivo que se requiera; esto ocasiona que la carga de la página web sea lenta.

Minificación: se eliminan los espacios en blanco, los comentarios y se acortan los nombres de las variables a un carácter; de tal forma que se pueda reducir el tamaño de los archivos CSS y JavaScript.

---

[31] Delegados: Representan referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto, un delegado es semejante un puntero seguro a una función en C y C++.
[32] LINQ (*Language Integrated Query*): Utiliza expresiones de consulta que son parecidas a las sentencias SQL, se utiliza para extraer y procesar convenientemente datos de arreglos, clases enumerables, documentos XML, bases de datos relacionales, entre otros.

1.4.2.11 NuGet

Es un sistema de administración de paquetes de .NET y Visual Studio para facilitar el uso al momento de agregar, modificar o eliminar librerías externas con sus respectivas dependencias en la aplicación web. También es posible crear paquetes propios y compartir con desarrolladores a nivel mundial [14].

1.4.3 BASE DE DATOS

Las bases de datos son conjuntos de datos pertenecientes a un mismo contexto que están almacenados sistemáticamente para su posterior uso; utilizan un Sistema Gestor de Base de Datos (DBMS), que es un software para administrar la base de datos y está compuesto por el DDL (*Data Definition Language*), DML (*Data Manipulation Language*) y SQL (*Structured Query Language*). Existen diferentes gestores de bases de datos, uno de ellos es SQL Server.

1.4.3.1 SQL Server

SQL Server es un Sistema para la Gestión de Bases de Datos Relacionales (RDBMS) desarrollado por Microsoft y diseñado para el entorno empresarial. Utiliza el lenguaje Transact-SQL (T-SQL), que es una implementación del estándar ANSI del lenguaje SQL que se utiliza para manipular y para recuperar datos (DML), para crear tablas y para definir las relaciones entre ellas (DDL), a través de la línea de comandos o de su interfaz gráfica denominada SQL Server Management Studio (SSMS). Este lenguaje se puede utilizar embebido en aplicaciones desarrolladas en lenguajes de programación como Visual Basic, C#, Java, entre otros.

Microsoft SQL Server 2016 está disponible desde junio del 2016, se desarrolló como parte de una estrategia de tecnología primero móvil y primero en la nube; que fue adoptada por Microsoft dos años antes. Se agregaron funciones para optimizar el rendimiento, para realizar análisis operacionales en tiempo real, para visualizar y para generar informes de datos en dispositivos móviles; además, es una base de datos escalable, la cual cuenta con el soporte híbrido en la nube que permite a los DBA[33] ejecutar bases de datos sobre una combinación de sistemas locales y servicios *cloud* públicos [36].

[33] DBA (*Database Administrator*): Es un profesional que dirige o lleva a cabo todas las actividades relacionadas con el mantenimiento de un entorno de base de datos.

1.4.4 DESARROLLO DE SOFTWARE

El desarrollo de software sigue una serie de actividades relacionadas que conducen a la elaboración de un producto de software. Estas actividades se refieren al desarrollo de software desde cero o el desarrollo de software empresarial en el que se extienden o se modifican los sistemas existentes.

Existen varios procesos para el desarrollo de software; sin embargo, incluyen cuatro actividades fundamentales en la ingeniería de software que se indican a continuación [37]:

- La especificación del software que consiste en la definición de la funcionalidad y las restricciones para su operación.
- El diseño e implementación del software para que cumpla con las especificaciones.
- La validación para asegurarse que se cumplen con los requerimientos del cliente.
- La evolución del software para satisfacer necesidades cambiantes de los clientes.

1.4.4.1 Proceso de desarrollo del software

Es un conjunto de actividades, acciones y tareas para desarrollar algún producto de software; los procesos son adaptables de tal forma que el equipo de software busca y elije un conjunto apropiado de acciones y tareas para su trabajo.

Actividad: Permite lograr un objetivo amplio como es el caso de la especificación del software, el diseño e implementación, la validación y la evolución del software.

Acción: Son un conjunto de tareas para desarrollar un producto, como ejemplo se tiene el diseño de la arquitectura.

Tarea: Son objetivos pequeños y definidos, como ejemplo se tiene la realización de pruebas de la aplicación.

Existe un enfoque del proceso prescriptivo en el desarrollo de software para conseguir un orden y una consistencia del proyecto; dado que todos los modelos del proceso del software (cascada, proceso incremental, proceso evolutivo,

concurrente, entre otros) pueden incluir actividades estructurales generales, pero en cada modelo se hace mayor énfasis en alguna fase del proceso; además, varían el flujo de trabajo en lo referente a las acciones y tareas; por lo tanto de una forma general como fases de desarrollo del software se tiene [37], [38]:

Requerimientos: Se debe identificar a los actores que interactúan con el sistema para conseguir los requerimientos que ayuden a definir las características y las funcionalidades del software.

Planeación: Se elabora un plan de proyecto de software para describir las tareas técnicas, los riesgos probables, los recursos requeridos, las actividades y el producto que se obtendrá.

Diseño: Se crea y se documenta un modelo mediante el diseño de los componentes que tendrá dicho software; ayudándose de los modelos arquitectónicos, de componentes, de objetos y de secuencias; todo ello con el fin de entender de mejor forma los requerimientos y como solventarlos.

Implementación: Consiste en generar el código del sistema de forma parcial o total siguiendo los parámetros del diseño; luego se entregará al cliente para ser evaluado y permitirá retroalimentarse de dicha información.

Pruebas: Es un proceso iterativo que se realiza en conjunto con la implementación y continúa luego de la implementación.

Mantenimiento: Permitirá evolucionar el software a través de actualizaciones para mejorar el sistema y corregir errores.

1.4.4.2 Pruebas en el desarrollo de software

Las pruebas son un conjunto de actividades dentro del desarrollo de software que pueden ser implementadas en cualquier momento del desarrollo de software, de acuerdo con la metodología utilizada [37], [39].

Pruebas de integración: Es la verificación del correcto funcionamiento de todos los componentes del sistema trabajando en conjunto; sin embargo, algunos componentes funcionan de forma aislada pero no significa que no encajen correctamente con los demás componentes.

Pruebas de validación: Es un proceso en el cual se verifica que el software cumpla con las especificaciones planteadas para cumplir con su objetivo.

Prueba de aceptación: El sistema se pone a prueba con los datos suministrados por el cliente del sistema, de esta forma se obtendrán problemas de requerimientos, de rendimiento del sistema o que no se cumpla con las necesidades del cliente.

1.4.4.3 Metodología

Las metodologías son un conjunto de procedimientos, herramientas y técnicas para el desarrollo de software; de acuerdo a la metodología elegida se tendrá diferentes etapas con sus respectivas tareas y restricciones. Existen una gran cantidad de metodologías que han sido desarrolladas durante el transcurso del tiempo, las cuales se diferencian por sus fortalezas y debilidades.

Las metodologías de desarrollo ágil se enfocan en el software en lugar del diseño y la documentación; a través del método incremental para obtener las especificaciones, realizar el desarrollo y la entrega del software; es indicado cuando los requerimientos del sistema cambian durante el desarrollo del software. Esta filosofía hace énfasis en la satisfacción del cliente y la entrega rápida; se puede aplicar a cualquier proceso de software. Las metodologías ágiles no son aplicables a todos los proyectos, productos, personas y situaciones.

Se tienen varias metodologías ágiles como: Programación Extrema (XP), *Scrum*, *Crystal*, desarrollo adaptativo, *Kanban*, entre otras [37].

1.4.4.4 Kanban

Kanban se define como “un sistema de producción altamente efectivo y eficiente”, esta metodología se origina de los procesos de producción “*just in time*” ideado por Toyota. En la actualidad es parte de las metodologías ágiles, su objetivo es gestionar de manera general el desarrollo de las tareas. Kanban procede de una palabra japonesa que significa “tarjetas visuales”, donde Kan es “visual” y Ban corresponde a “tarjeta” [40].

El método Kanban que originalmente se aplicó a la automatización de la industria, fue adaptado por David J. Anderson siguiendo la filosofía original en el desarrollo del software, la cual fue utilizada por primera vez en Microsoft.

Kanban es una metodología muy fácil de utilizar, actualizar y asumir por parte del equipo de trabajo; permite efectuar entregas en cualquier instante, facilita el cambio de las prioridades de las tareas y cuenta con una visualización perfecta del flujo de trabajo. La gestión de las tareas es visual, donde se destacará el estado de cada tarea para desarrollar el proyecto de manera efectiva.

Kanban es una metodología especialmente indicada para las organizaciones que requieren de flexibilidad fundamentalmente en las tareas de entrada, durante el seguimiento, la priorización o en la supervisión del equipo de trabajo mediante informes de dedicación.

1.4.4.4.1 Principios de Kanban

Kanban utiliza 5 propiedades principales para mejorar el desarrollo de software siguiendo esta metodología [41]:

1. Visualiza el flujo de trabajo a través de un tablero que se utiliza para entender el proceso de trabajo; conocer y tomar decisiones frente a los problemas que puedan surgir.
2. Limita el trabajo en curso, que radica en acordar anticipadamente la cantidad de tareas que pueden abordarse por cada proceso; es decir, por columnas del tablero para evitar cuellos de botella.
3. Administra el flujo de trabajo para optimizar el proceso.
4. Define las políticas del proceso explícitas a seguir.
5. Utiliza modelos para reconocer oportunidades de mejora consiguiendo un desarrollo estable, continuo y previsible.

1.4.4.4.2 Tarjetas de Kanban

Las tarjetas Kanban se utilizan para transmitir visualmente el progreso de un elemento de trabajo a medida que fluye a través de un sistema o proceso; las tarjetas se elaboran de acuerdo al entorno de trabajo en el que se encuentra; por lo tanto, se tiene diversos tipos de tarjetas Kanban; algunas de ellas se describen a continuación:

Kanban urgente: Se asigna en caso de escasez de una pieza o de un elemento para que sea atiendo de forma inmediata.

Kanban de emergencia: Se emite de modo temporal para resolver algún problema.

Kanban orden de trabajo: Se utiliza para la fabricación de una línea específica y se emite con cada orden de trabajo.

Kanban único: Se emite cuando los procesos están vinculados y pueden verse como un único proceso, también se utiliza una ficha Kanban común para varios procesos.

Kanban común: Se utiliza como Kanban de movimiento o Kanban de producción cuando la distancia entre dos procesos es muy corta y ambos cuentan con el mismo supervisor.

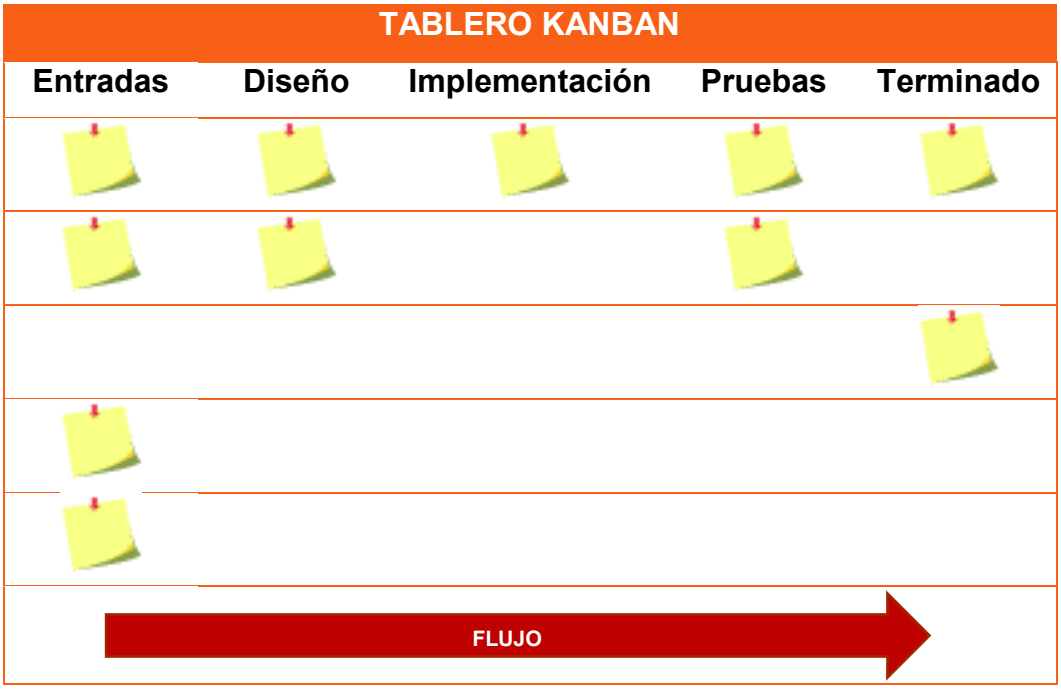
1.4.4.4.3 Aplicación de Kanban

La aplicación de Kanban necesitará de un tablero de tareas el cual permitirá visualizar el flujo de trabajo. Para implementar esta metodología, se deben tener claros los siguientes aspectos [40]:

1. Definir las tareas y etapas del proyecto para visualizar el flujo de trabajo en el tablero Kanban.
2. Utilizar columnas con su respectivo nombre para ilustrar el lugar en donde se encuentra cada uno de los elementos en el flujo de trabajo; a su vez, las columnas corresponderán a un estado en concreto del flujo de trabajo; de tal forma que el tablero Kanban tendrá tantas columnas como estados, de acuerdo a las necesidades del proyecto.
3. Dividir el trabajo en bloques o tareas, donde elemento se escribe en una tarjeta y se coloca en el tablero.
4. Kanban permite priorizar el trabajo que está en curso, antes de dar inicio a nuevas tareas.
5. Monitorear continuamente el proceso para asegurar mejoras continuas en el desarrollo del software.

En la Tabla 1.7 se presenta un tablero simbólico de Kanban con varios estados y con varias tareas que se encuentran en diferentes estados.

Tabla 1.7. Tablero Kanban



1.4.4.4.4 Software para Kanban

Algunos profesionales implementan Kanban físicamente utilizando notas adhesivas o tableros con ranuras; sin embargo, es común utilizar software para dar seguimiento a los proyectos, por lo que se tiene:

Trello [42]**:** Es de pago, pero también se puede crear una cuenta gratuita con limitaciones; es flexible y visualmente se organiza todo con cualquier persona.

Kanban Tool [43]**:** Es un tablero de Kanban en la nube de pago; cuenta con la opción de crear una cuenta gratuita para 2 usuarios y 2 tableros.

Atlassian JIRA Software [44]**:** Herramienta de desarrollo de software para equipos ágiles que se ofrece en la nube o para implementar en servidores propios.

Virtual Kanban [45]**:** Tablero de Kanban gratuito que se puede acceder online o se puede tener de forma *offline* en un fichero HTML.

Targetprocess [46]**:** Es un Software de Gestión de Proyectos Visuales de pago, pero se puede adquirir una cuenta gratuita con limitaciones.

CAPÍTULO 2

2. ANÁLISIS DE REQUERIMIENTOS Y DISEÑO DE LA APLICACIÓN

2.1 ANÁLISIS DE REQUERIMIENTOS

Los requerimientos de la aplicación web se han obtenido mediante la técnica de entrevista con el instrumento de cuestionario que se aplicó en el DETRI a un grupo de profesores, a los Coordinadores de las carreras, al Jefe del DETRI y a miembros del CODEI; también se mantuvieron reuniones durante el primer semestre del año 2017 con el Coordinador de la carrera de Ingeniería en Electrónica y Redes de Información M.Sc. David Mejía y con el Coordinador de la carrera de Ingeniería en Electrónica y Telecomunicaciones M.Sc. Jorge Carvajal.

Para la obtención de requerimientos también se revisó el documento del modelo genérico de evaluación y acreditación de carreras del CEAACES, donde se indica lo que se evaluarán [47]:

- Las actividades de docencia que comprenden los programas de las asignaturas con sus instrumentos que hacen operativo el desarrollo de la asignatura y el logro de los objetivos propuestos como el sílabo, las prácticas de la asignatura, entre otros.
- La producción académica y científica que se refiere a los trabajos e investigaciones científicas que están relacionados con el área de conocimiento o con la asignatura que imparte el profesor, entre estos están los artículos científicos de los profesores de la carrera que han sido publicados en revistas que figuran en las bases de datos como SCIMAGO (Scopus) o en las bases del ISI Web of Knowledge.
- La actualización científica o pedagógica del profesor relacionada con su formación y actualización que guarde relación con las asignaturas que imparte o con su producción investigativa.

- Los libros o capítulos de libros de los trabajos e investigaciones científicas que fueron elaborados y publicados por los profesores de la carrera.
- Las ponencias que son las presentaciones de los avances o los resultados de una investigación sobre un tema específico, desarrolladas en eventos académicos o científicos nacionales e internacionales que estén publicadas en las memorias de dichos eventos.
- La gestión académica que se encarga de la dirección y la gestión de los procesos de docencia e investigación que está representada por un equipo de académicos, con el objetivo de mejorar la calidad educativa en todos los niveles de la organización académica e institucional.

2.1.1 METODOLOGÍA

El presente Trabajo de Titulación inicia con una revisión bibliográfica de los aspectos fundamentales para el desarrollo de la aplicación web, que se encuentra descrito en el Marco Teórico.

La aplicación web se desarrolló utilizando la metodología Kanban, la cual forma parte de las metodologías ágiles para el desarrollo de software.

Los requerimientos de la aplicación web se obtuvieron mediante la técnica de entrevista, la cual se aplicó en el DETRI a un grupo de profesores; además se mantuvieron reuniones con los Coordinadores de carreras del DETRI y se analizaron los documentos del modelo genérico de evaluación de carreras del CEAACES. Los requerimientos de la aplicación web están relacionados con la documentación de los profesores que debe ser ingresada, visualizada e informada tanto a los profesores como a los coordinadores; además, la aplicación web será administrada.

El diseño de la aplicación web se realizó mediante diagramas de casos de uso y diagramas de clases, de acuerdo con los requerimientos obtenidos; a su vez, para el diseño de la base de datos se realizó el diagrama entidad-relación que está fundamentado en el modelo relacional.

La implementación se realizó utilizando las tecnologías de ASP.NET MVC y SQL Server; siguiendo los pasos de la metodología de desarrollo ágil Kanban.

Una vez implementada la aplicación web se realizaron las pruebas de integración, validación y aceptación para garantizar el correcto funcionamiento de la misma; se realizaron las respectivas correcciones requeridas que se detectaron durante las pruebas para el correcto funcionamiento de la aplicación web. Además, se fue documentando el trabajo realizado.

2.1.2 SITUACIÓN ACTUAL

Como antecedente se puede mencionar que la documentación de los profesores es almacenada manualmente cada semestre; hay ocasiones en que se extravía cierta documentación y en otras ocasiones los profesores se olvidan de entregar a tiempo la documentación solicitada.

La búsqueda de dicha documentación no es rápida aun cuando la información se encuentre bien organizada. Cuando se cambia de autoridades es complicado dar continuidad a este proceso, debido a que las nuevas autoridades deberán entender todo el proceso y continuar realizando la recolección de la documentación a mano.

Por los inconvenientes presentados es importante realizar la automatización del proceso de recolección de los documentos de los profesores.

2.1.3 REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales que se obtuvieron son:

1. La aplicación web permitirá al administrador crear, modificar y eliminar usuarios.
2. La aplicación web permitirá gestionar roles de usuario para asignar permisos dentro de la misma.
3. La aplicación web permitirá la autenticación de usuarios.
4. La aplicación web permitirá al administrador gestionar categorías para organizar la documentación de los profesores recolectada en los formularios establecidos por el administrador.
5. La aplicación web permitirá al administrador crear, modificar o eliminar los formularios con sus respectivos campos, donde los profesores ingresarán la información solicitada.

6. Los profesores ingresarán la información en cada uno de los formularios generados por el administrador, además la información se podrá visualizar, modificar o eliminar.
7. El administrador tendrá la capacidad de validar la información ingresada por el profesor, también podrá habilitar la edición de dicha información una vez aprobada.
8. La aplicación web permitirá imprimir informes de la información ingresada tanto a los profesores como a los administradores.
9. La aplicación web generará notificaciones a los profesores sobre la información que deberán ingresar en los plazos señalados.
10. La aplicación web generará eventos (*logs*) que se podrán visualizar por los profesores y por los administradores.
11. La aplicación permitirá agregar, modificar o eliminar los periodos académicos con sus respectivas fechas.

2.1.4 REQUERIMIENTOS NO FUNCIONALES

1. Las contraseñas serán cifradas para su posterior almacenamiento en la base de datos.
2. El contenido de la aplicación web será responsivo, de tal forma que se adapte a cualquier tamaño de pantalla de los dispositivos.
3. Se utilizarán conexiones seguras mediante SSL.
4. Se empleará ASP.NET MVC para el desarrollo de la aplicación web.
5. La aplicación web se alojará en un servidor IIS.
6. La base de datos será montada con SQL Server.
7. Se utilizará una cuenta de correo electrónico para el envío de mensajes con notificaciones a los profesores.

2.1.5 PROPUESTA DEL PROTOTIPO

De acuerdo con los requerimientos obtenidos para el desarrollo de la aplicación web, se ha dividido la aplicación web en diferentes módulos los cuales se mencionan a continuación:

- Módulo para la administración y autenticación de usuarios.
- Módulo para la recepción de la documentación de los profesores.

- Módulo para generar, visualizar e imprimir informes.
- Módulo para generar notificaciones a los profesores.
- Módulo para el registrar y visualizar los eventos (*logs*).

2.2 DISEÑO DE LA APLICACIÓN

Para realizar el diseño de la aplicación web se partió de los requerimientos obtenidos previamente, se utilizó la metodología Kanban para definir las tareas y las etapas que fueron empleadas para desarrollar la aplicación web. Se realizó la descripción de la arquitectura, se realizaron diagramas de casos de uso y diagramas de clases, así como diagramas de entidad relación y diagramas relacionales; a su vez, se realizaron *mockups* que son vistas previas de la interfaz gráfica la aplicación web.

2.2.1 PLANIFICACIÓN KANBAN

Se realizó la planificación del proyecto siguiendo la metodología Kanban, por lo que se elaboró un tablero Kanban que contiene el flujo de trabajo que se seguirá para llevar a cabo el desarrollo de la aplicación web, se definieron las columnas que corresponden a cada una de las etapas por las que pasará el desarrollo de la aplicación web y finalmente se dividió el trabajo en diferentes tareas de acuerdo a los requerimientos obtenidos.

Las columnas del tablero Kanban son las siguientes: entradas que corresponden a las tareas de la aplicación web; diseño que corresponden a los diagramas realizados para el diseño de la aplicación web; implementación que corresponde a la codificación de la aplicación web; pruebas que corresponden a las actividades realizadas para comprobar que se cumplan con los requerimientos; y, terminado que corresponde a las tareas concluidas.

Para una mejor visualización del tablero Kanban de la aplicación web, se ha procedido a dividir en tableros Kanban más pequeños que corresponden a cada uno de los módulos, donde se indican las respectivas tareas que se llevarán a cabo para el desarrollo de la aplicación web, las cuales se detallan a continuación: Se inicia el proyecto de la aplicación web realizando varias tareas iniciales, las cuales se presentan en la Tabla 2.1. Las tareas del módulo de administración y

autenticación de usuarios se presentan en la Tabla 2.2. Las tareas para la recepción de los documentos de los profesores se presentan en la Tabla 2.3. Las tareas del módulo para generar, visualizar e imprimir los informes se presentan en la Tabla 2.4. Las tareas para generar las notificaciones a los profesores se presentan en la Tabla 2.5. Las tareas del módulo para registrar y visualizar los eventos se presentan en la Tabla 2.6. Finalmente se van a llevar a cabo tareas adicionales que se denominan de esta forma porque no constan en el Plan del Trabajo de Titulación, pero son necesarias para el correcto funcionamiento de la aplicación web, estas tareas se presentan en la Tabla 2.7.

Tabla 2.1. Tablero Kanban de las tareas iniciales

TAREAS INICIALES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Creación de la base de datos con SQL Server Management Studio.	Desarrollo de los diagrama entidad relación y relacional.	Codificación de los *scripts* para crear la base de datos.	Prueba de ingreso y eliminación de los registros desde el SQL Server Management Studio.	Base de datos de la aplicación web.
Creación del proyecto con ASP.NET MVC para la aplicación web.	Desarrollo del diagrama de la arquitectura de la aplicación web.	Creación de un proyecto `Acreditacion` y de un proyecto `Datos`.	Ejecución del proyecto base que genera Visual Studio.	El proyecto de la aplicación web.
Generación de las entidades del modelo con Entity Framework.	Se parte de la base de datos creada con sus respectivos diagramas.	Integración de Entity Framework al proyecto `Datos`.	Acceso desde el proyecto `Acreditacion` a las entidades del modelo.	Entidades del modelo de la aplicación web.
Creación de los métodos CRUD en las entidades del modelo.	Diagramas de clases.	Codificación de los métodos CRUD en las entidades del modelo.	Acceso a los métodos CRUD desde los controladores.	Métodos CRUD de las entidades del modelo.
Integración de una plantilla HTML para las vistas de la aplicación web.	No se realiza ningún diseño.	Integración de las librerías y de los estilos al proyecto `Acreditacion`.	Visualización de la plantilla HTML y sus estilos en el navegador web.	La apariencia y los estilos de la aplicación web.

Tabla 2.2. Tablero Kanban del módulo de autenticación de usuarios

MÓDULO DE ADMINISTRACIÓN Y AUTENTICACIÓN DE USUARIOS				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para registrar usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del registro de usuarios.	Registrar usuarios.	Registro de usuarios.
Desarrollar un método para cambiar de rol a los usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para cambiar de rol a los usuarios.	Cambiar de rol a un usuario.	Cambiar de roles a los usuarios.
Desarrollo de un método para activar y desactivar usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para activar y desactivar usuarios.	Activar y desactivar usuarios.	Activar y desactivar usuarios.
Desarrollar una vista para la edición y la visualización de la información del usuario.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para la edición y la visualización de la información del usuario.	Visualizar y editar la información de un usuario.	Vista para la edición y la visualización de la información del usuario.
Desarrollar un método para eliminar usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para eliminar usuarios.	Eliminar un usuario.	Método para eliminar usuarios.
Desarrollar una vista para presentar la lista de usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista que muestra la lista de los usuarios.	Visualizar la lista de los usuarios registrados.	Vista que muestra la lista de usuarios registrados.
Desarrollar métodos para generar un archivo en formato de Excel y PDF con la lista los usuarios registrados.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de los métodos para generar los archivos en formato de Excel y PDF con la lista de los usuarios registrados.	Generar un archivo en formato de Excel y un archivo en formato PDF con la lista de los usuarios registrados.	Método para generar archivos en formato de Excel y PDF con la lista de los usuarios registrados.
Desarrollar un método para la autenticación de usuarios en la aplicación web.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para la autenticación de usuarios en la aplicación web.	Autenticación de los usuarios.	Autenticación de usuarios en la aplicación web.

Tabla 2.3. Tablero Kanban del módulo de recepción de documentos (Parte 1 de 3)

MÓDULO PARA LA RECEPCIÓN DE LA DOCUMENTACIÓN DE LOS PROFESORES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para mostrar la lista de categorías.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para mostrar la lista de categorías.	Visualizar la lista de categorías.	Vista para mostrar la lista de categorías.
Desarrollar una vista para crear o modificar las categorías.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para crear o modificar las categorías.	Crear y editar una categoría.	Vista para crear o modificar las categorías.
Desarrollar un método para eliminar categorías.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar categorías.	Eliminar una categoría.	Método para eliminar categorías.
Desarrollar una vista para mostrar la lista de formularios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para mostrar la lista de formularios.	Visualizar la lista de formularios.	Vista para mostrar la lista de formularios.
Desarrollar una vista para crear o modificar los formularios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para crear o modificar un formulario.	Crear y editar un formulario.	Vista para crear o editar un formulario.
Desarrollar un método para eliminar formularios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar formularios.	Eliminar un formulario.	Método para eliminar formularios.
Desarrollar una vista para mostrar la lista de campos de un formulario.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para mostrar la lista de los campos de un formulario.	Visualizar la lista de los campos de un formulario.	Vista para mostrar la lista de los campos de un formulario.

Tabla 2.3 Tablero Kanban del módulo de recepción de documentos (Parte 2 de 3)

MÓDULO PARA LA RECEPCIÓN DE LA DOCUMENTACIÓN DE LOS PROFESORES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para crear o modificar los campos de un formulario.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para crear o modificar los campos de un formulario.	Crear y modificar los campos de un formulario.	Vista para crear o modificar los campos de un formulario.
Desarrollar un método para eliminar los campos de un formulario.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar los campos de un formulario.	Eliminar los campos de un formulario.	Método para eliminar los campos de un formulario.
Desarrollar una vista para mostrar la lista de los registros de la documentación ingresada por los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para mostrar la lista de los registros de la documentación ingresada por los profesores.	Visualizar lista de los registros de la documentación ingresada por los profesores.	Vista para mostrar la lista de los registros de la documentación ingresada por los profesores.
Desarrollar una vista para ingresar o modificar la documentación de los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para ingresar o modificar la documentación de los profesores.	Ingresar y modificar los registros de la documentación de los profesores.	Vista para ingresar o modificar la documentación de los profesores.
Desarrollar un método para eliminar los registros de la documentación ingresada.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar los registros de la documentación ingresada.	Eliminar un registro de la documentación ingresada de un profesor.	Método para eliminar registros de la documentación ingresada por los profesores.
Desarrollar una vista para validar la documentación ingresada por los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para validar la documentación ingresada por los profesores.	Aprobar o rechazar la documentación ingresada por los profesores.	Vista para validar la documentación ingresada por los profesores.

Tabla 2.3. Tablero Kanban del módulo de recepción de documentos (Parte 3 de 3)

MÓDULO PARA LA RECEPCIÓN DE LA DOCUMENTACIÓN DE LOS PROFESORES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para visualizar la lista de los portafolios académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para visualizar la lista de los portafolios académicos.	Visualizar la lista de los portafolios académicos.	Vista para visualizar la lista de los portafolios académicos.
Desarrollar una vista para crear o modificar los portafolios académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para crear o modificar los portafolios académicos.	Crear y modificar un portafolio académico.	Vista para crear o modificar los portafolios académicos.
Desarrollar un módulo para eliminar los portafolios académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar los portafolios académicos.	Eliminar un portafolio académico.	Método para eliminar los portafolios académicos.
Desarrollar una vista para ingresar la documentación a los portafolios académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para ingresar la documentación de los portafolios académicos.	Ingresar la documentación en un portafolio académico.	Vista para ingresar la documentación a los portafolios académicos.
Desarrollar una vista para validar los documentos de los portafolios académicos de los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para validar los documentos de los portafolios académicos de los profesores.	Aprobar o rechazar los documentos ingresados en los portafolios académicos de los profesores.	Vista para validar los documentos de los portafolios académicos de los profesores.

Tabla 2.4. Tablero Kanban del módulo de informes

MÓDULO PARA GENERAR, VISUALIZAR E IMPRIMIR INFORMES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para establecer el encabezado y el pie de página de los informes para los profesores y para los administradores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para configurar el encabezado y el pie de página de los informes.	Verificar que el informe tenga el encabezado y el pie de página configurado.	Vista para configurar el encabezado y el pie de página de los informes.
Desarrollar una vista para mostrar los informes y su impresión.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para visualizar los informes y su impresión.	Visualización de los informes y su posterior impresión.	Vista para visualizar los informes y su impresión.
Desarrollar un método para generar un archivo en formato de Excel de cada informe.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para generar un archivo en formato de Excel de un informe.	Generar un informe en formato de Excel, tanto para el profesor como para el administrador.	Generar informes en formato de Excel.
Desarrollar un método para generar un informe en formato PDF.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para generar un informe en formato PDF.	Generar un informe en formato PDF, tanto para el profesor como para administrador.	Generar informes en formato PDF.

Tabla 2.5. Tablero Kanban del módulo de notificaciones

MÓDULO PARA GENERAR NOTIFICACIONES A LOS PROFESORES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar un método para generar las notificaciones a los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para generar las notificaciones a los profesores.	Revisar el correo electrónico recibido de la notificación del profesor.	Método para enviar las notificaciones a los profesores.
Desarrollar una vista para configurar las notificaciones de los profesores.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para configurar las notificaciones de los profesores.	Realizar la configuración de las notificaciones de los profesores.	Vista para configurar las notificaciones de los profesores.

Tabla 2.6. Tablero Kanban del módulo de eventos

MÓDULO PARA EL REGISTRAR Y VISUALIZAR LOS EVENTOS				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar un método para generar y almacenar los eventos de los usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación del método para almacenar los eventos de los usuarios.	Realizar algunas acciones para generar eventos en la aplicación web.	Método para almacenar los eventos de los usuarios.
Desarrollar una vista para visualizar los eventos de los usuarios.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para visualizar los eventos de los usuarios.	Visualización de los eventos de los usuarios.	Vista para mostrar los eventos de los usuarios.

Tabla 2.7. Tablero Kanban de las tareas adicionales

TAREAS ADICIONALES				
ENTRADAS	**DISEÑO**	**IMPLEMENTACIÓN**	**PRUEBAS**	**TERMINADO**
Desarrollar una vista para configurar los mensajes de los correos electrónicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para configurar los mensajes de correo electrónico.	Visualización del correo electrónico recibido con el respectivo formato establecido.	Vista para la configuración de los mensajes de los correos electrónicos.
Desarrollar una vista para configurar una cuenta de correo electrónico para el envío de mensajes.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de la vista para configurar una cuenta de correo electrónico para el envío de mensajes.	Comprobar que se almacene y se utilice la información de la cuenta de correo electrónico ingresado.	Vista para configurar una cuenta de correo electrónico para el envío de mensajes.
Desarrollar una vista para visualizar la lista de los periodos académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para visualizar la lista de los periodos académicos.	Visualizar los periodos académicos.	Vista para visualizar la lista de los periodos académicos.
Desarrollar una vista para crear o modificar un periodo académico.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de una vista para crear o modificar un periodo académico.	Crear y modificar la información de un periodo académico.	Vista para crear o modificar un periodo académico.
Desarrollar un método para eliminar los periodos académicos.	Diagrama de casos de usos, de clases, de entidad-relación y relacional.	Codificación de un método para eliminar los periodos académicos.	Eliminar un periodo académico.	Método para eliminar los periodos académicos.

2.2.2 ARQUITECTURA DE LA APLICACIÓN WEB

El desarrollo del presente proyecto se realizó con la utilización de tecnologías de Microsoft como: ASP.NET MVC y SQL SERVER, motivo por el cual la aplicación web se alojará en un servidor web IIS; cabe mencionar, que al tratarse de un prototipo se puede alojar tanto la aplicación web como la base de datos en el mismo servidor, de acuerdo a la Figura 2.1.

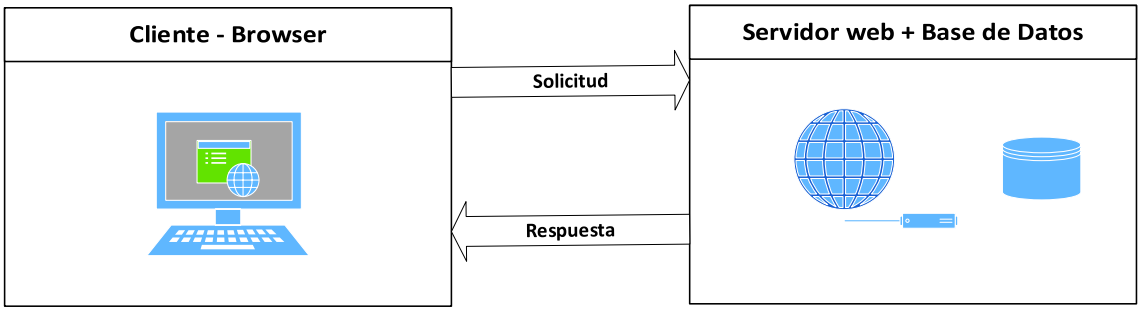


Figura 2.1. Arquitectura cliente - servidor

ASP.NET MVC permite la separación en capas lógicas las cuales cumplen funciones específicas, como se observa en la Figura 2.2. La aplicación web está desarrollada para que sea extensible y escalable en el futuro, por tal motivo se realizó una separación de capas, donde el controlador y la vista se desarrollaron en un mismo proyecto de Visual Studio; mientras que en otro proyecto se desarrolló el modelo que es el encargado de interactuar con la base de datos; de esta forma se pueden desarrollar y ejecutar otros proyectos simultáneamente como una API, o una aplicación de escritorio de ser necesario.

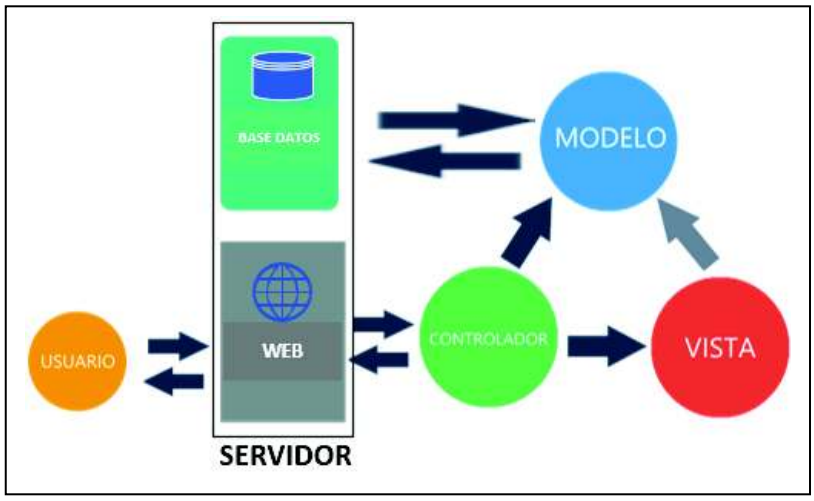


Figura 2.2. Arquitectura de la aplicación web

2.2.3 FUNCIONAMIENTO DEL PROTOTIPO

La aplicación web contará con 3 roles de usuario: un rol denominado Profesor que será asignado a todos los profesores que tienen que ingresar la documentación solicitada en la aplicación web; un rol de Administrador que se encargará de aprobar o rechazar la información ingresada por los profesores; y, un rol de Súper Administrador que tendrá la capacidad de realizar las configuraciones de la aplicación web. Cabe señalar que tanto el administrador como el súper administrador podrán ingresar la documentación en la aplicación web como lo hacen los profesores. En la aplicación web se van a tener dos tipos de usuarios: los usuarios activos que son quienes tendrán que ingresar la documentación solicitada y los usuarios inactivos que son los usuarios registrados, pero que por alguna circunstancia no tendrán que ingresar la documentación de algún periodo académico.

La aplicación web se ha dividido en varios módulos que tienen diferentes funcionalidades, las cuales se describen a continuación:

1. **Módulo para la administración y autenticación de usuarios:** En este módulo el administrador y el súper administrador podrán registrar, visualizar y eliminar usuarios; podrán asignar roles; podrán poner a los usuarios como activos o inactivos; cabe señalar que los usuarios inactivos no podrán ingresar la documentación en la aplicación web; además, en este módulo se define el mecanismo de autenticación y autorización de los usuarios en la aplicación web, que se encarga de controlar el acceso de los usuarios a los respectivos módulos de la aplicación web, de acuerdo su rol asignado.
2. **Módulo para la recepción de la documentación de los profesores:** Este módulo permite crear formularios con sus respectivos campos para que los profesores ingresen la documentación solicitada, además se podrán organizar los formularios en diferentes categorías según sea necesario.

   Este módulo permite crear categorías que son utilizadas para organizar la información de los formularios creados en la aplicación web, además este módulo crea carpetas con los nombres de cada categoría en el directorio que se especifique en la aplicación web para poder almacenar la documentación de los profesores que se ingresa en cada formulario.

Los archivos de la documentación ingresados tendrán un nombre específico que permitirá identificar rápidamente a que profesor le corresponde y saber de qué tipo de documento se trata, por lo que se utiliza el siguiente formato: `nombre del formulario_[nombres del profesor]_periodo académico número de archivo`, como ejemplo se tiene: `INVESTIGACIÓN_[ALVARADO RAMIREZ KARLA MARIA]_2016A_1.pdf`. Los documentos ingresados en el portafolio de cada asignatura impartida por el profesor, se almacenarán en una carpeta denominada `portafolio` y tendrán el siguiente formato: `código de la asignatura_Grupo_[nombres del profesor]_periodo académico`, como ejemplo se tiene: `ADM413_GR1_[ALVARADO RAMIREZ KARLA MARIA]_2016A.pdf`; además, la documentación de los portafolios debe ingresarse dentro de un plazo establecido por el administrador o por el súper administrador.

3. **Módulo para generar, visualizar e imprimir informes:** Este módulo permitirá a los profesores, a los administradores y a los súper administradores visualizar e imprimir un informe de la información ingresada; de acuerdo al rol del usuario se generará la información apropiada con el respectivo formato de encabezado y de pie de página que estableció el súper administrador, con la finalidad para que se pueda imprimir o almacenar de forma digital.
4. **Módulo para generar notificaciones a los profesores:** Este módulo permitirá generar y enviar correos electrónicos de recordatorio a los usuarios sobre la documentación que tienen que ingresar en la aplicación web; el texto del mensaje del correo electrónico podrá ser personalizado únicamente por el súper administrador.
5. **Módulo para el registrar y visualizar los eventos (*logs*):** Este módulo permite registrar las acciones realizadas por los usuarios, almacenar y visualizar posteriormente; cabe señalar que se almacenarán únicamente los 50 últimos eventos de los usuarios, por el motivo en que si se almacenan los eventos de forma ilimitada se consumirán los recursos del sistema de forma innecesaria; además, el administrador y el súper administrador podrán visualizar los eventos de todos los usuarios registrados en la aplicación web.

En la Figura 2.3 se presenta un mapa de navegación de las vistas de la aplicación web a las que tendrán acceso los usuarios de acuerdo a su rol asignado; cabe señalar que tanto las categorías como los formularios pueden ser agregados, modificados o eliminados según sea necesario por el administrador o por el súper administrador.

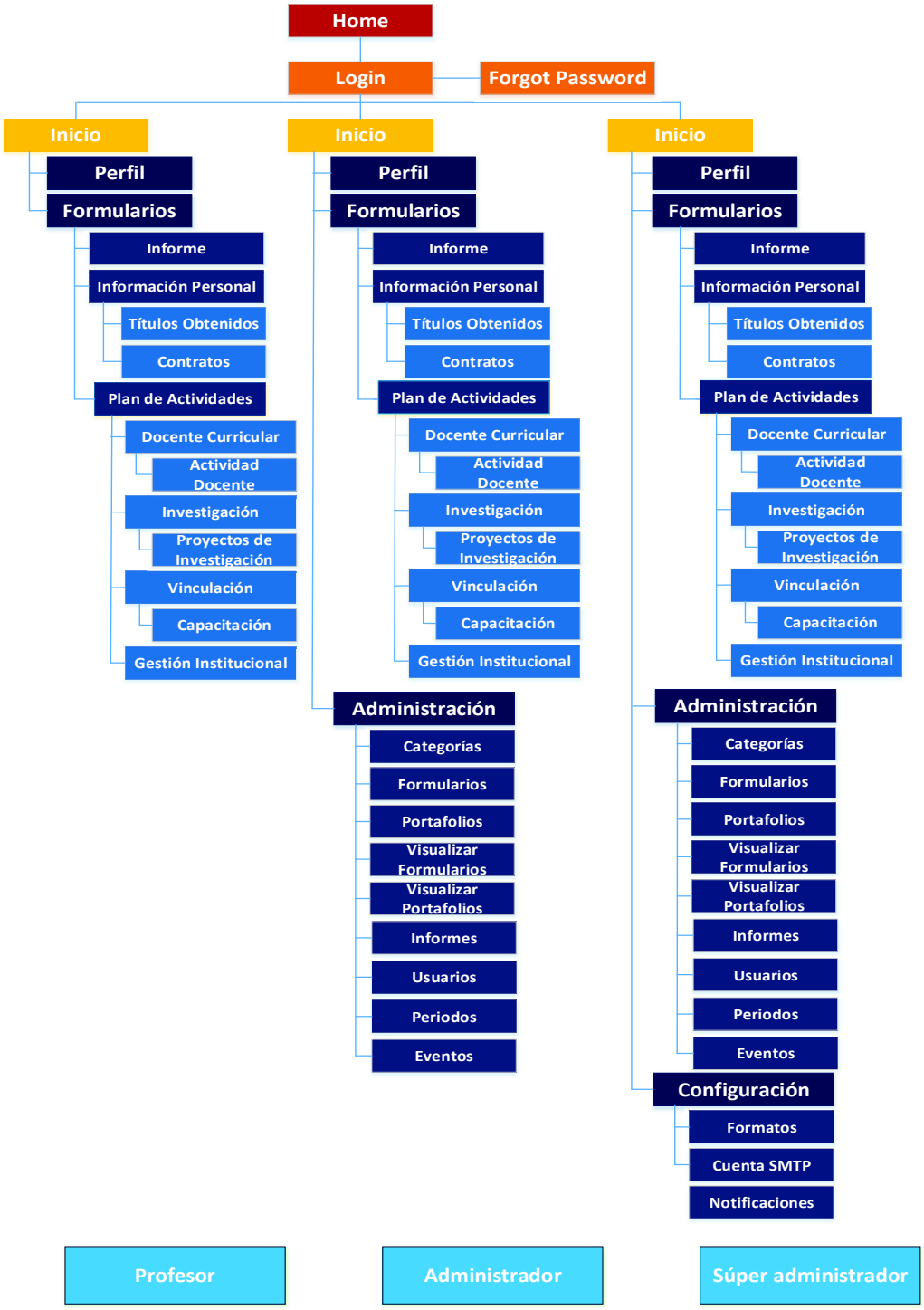


Figura 2.3. Mapa de navegación y el rol de los usuarios en la aplicación web

2.2.4 DIAGRAMAS

El diseño de la aplicación web se ha realizado a través de diferentes diagramas, los cuales son representaciones que definen el funcionamiento del sistema.

2.2.4.1 Diagrama de casos de uso

Los diagramas de casos de uso son una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso.

En la Figura 2.4 se presenta el diagrama de casos de uso de la aplicación web, donde se puede observar las principales funcionalidades del sistema, en donde el administrador y el súper administrador contarán con la capacidad de gestionar los usuarios, crear formularios que estarán organizado en categorías para recolectar la documentación de los profesores. Los profesores ingresarán la documentación solicitada en los respectivos formularios para que el administrador o el súper administrador puedan validar dicha información. Se cuenta con la función de imprimir informes de la documentación ingresada, tanto por los administradores como por los súper administradores y los profesores. Los administradores y súper administradores podrán gestionar los periodos académicos que serán utilizados para ingresar y visualizar organizadamente la documentación de los profesores. Todos los usuarios de la aplicación web podrán visualizar los eventos que se generan cada vez que realizan alguna acción en la aplicación web; cabe señalar que los administradores y los súper administradores podrán visualizar los eventos de todos los usuarios de la aplicación web.

Los súper administradores podrán realizar algunas configuraciones en la aplicación web como: establecer los encabezados y pies de páginas de los informes que se imprimirán; establecer los mensajes de correo electrónico que se enviarán con notificaciones indicando que la documentación ingresada fue rechazada, indicando que se debe ingresar la documentación requerida en los formularios y en los portafolios académicos establecidos, entre otros; configurar la hora y el periodo de envío de las notificaciones a los profesores. También se podrá configurar una cuenta de correo electrónico que es utilizada para el envío de mensajes de correo electrónico a los profesores. El diagrama de casos de uso con su descripción se encuentra en el Anexo A.

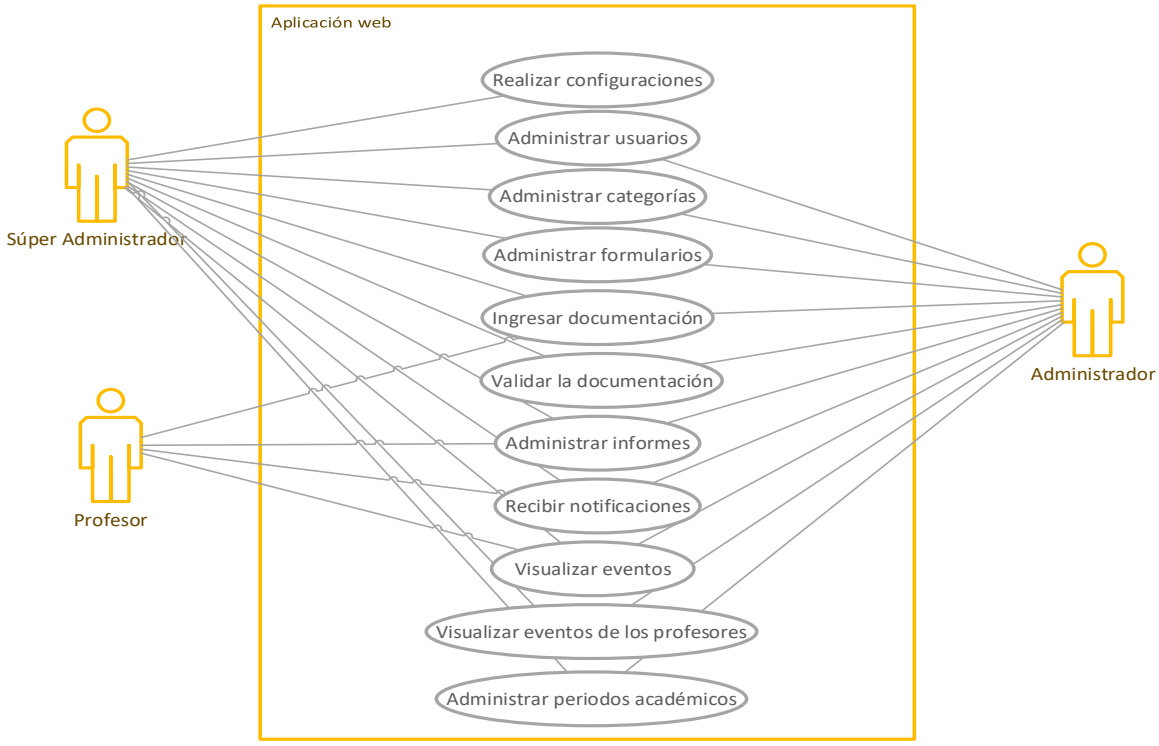


Figura 2.4. Diagrama de casos de uso

2.2.4.2 Diagrama de clases

El diagrama de clases describe la estructura de la aplicación web mediante atributos, métodos y relaciones. En este Proyecto cada controlador y cada entidad del modelo están representados por una clase.

En la Figura 2.5 se presentan todas las clases del controlador de la aplicación web, estas clases se denominan controladores en ASP.NET MVC, heredan diferentes funcionalidades de la clase `controller`; además, tienen sus respectivos atributos y métodos (acciones) utilizados para la lógica de la aplicación web.

Se tiene los siguientes controladores: `InicioController` que es utilizado para presentar una vista pública. `AreaPersonalController` que es utilizado para gestionar la información de los usuarios. `FormularioController` que es utilizado para gestionar las categorías, los formularios, el ingreso de documentación, la validación de la documentación ingresada y la generación de informes. `PortafolioController` que se encarga administrar los portafolios de las asignaturas, ingresar información, visualizar y validar dicha información. `AdministracionController` se utiliza para realizar la administración de la

aplicación web. Finalmente, `ConfiguracionController` permite realizar algunas configuraciones en la aplicación web.

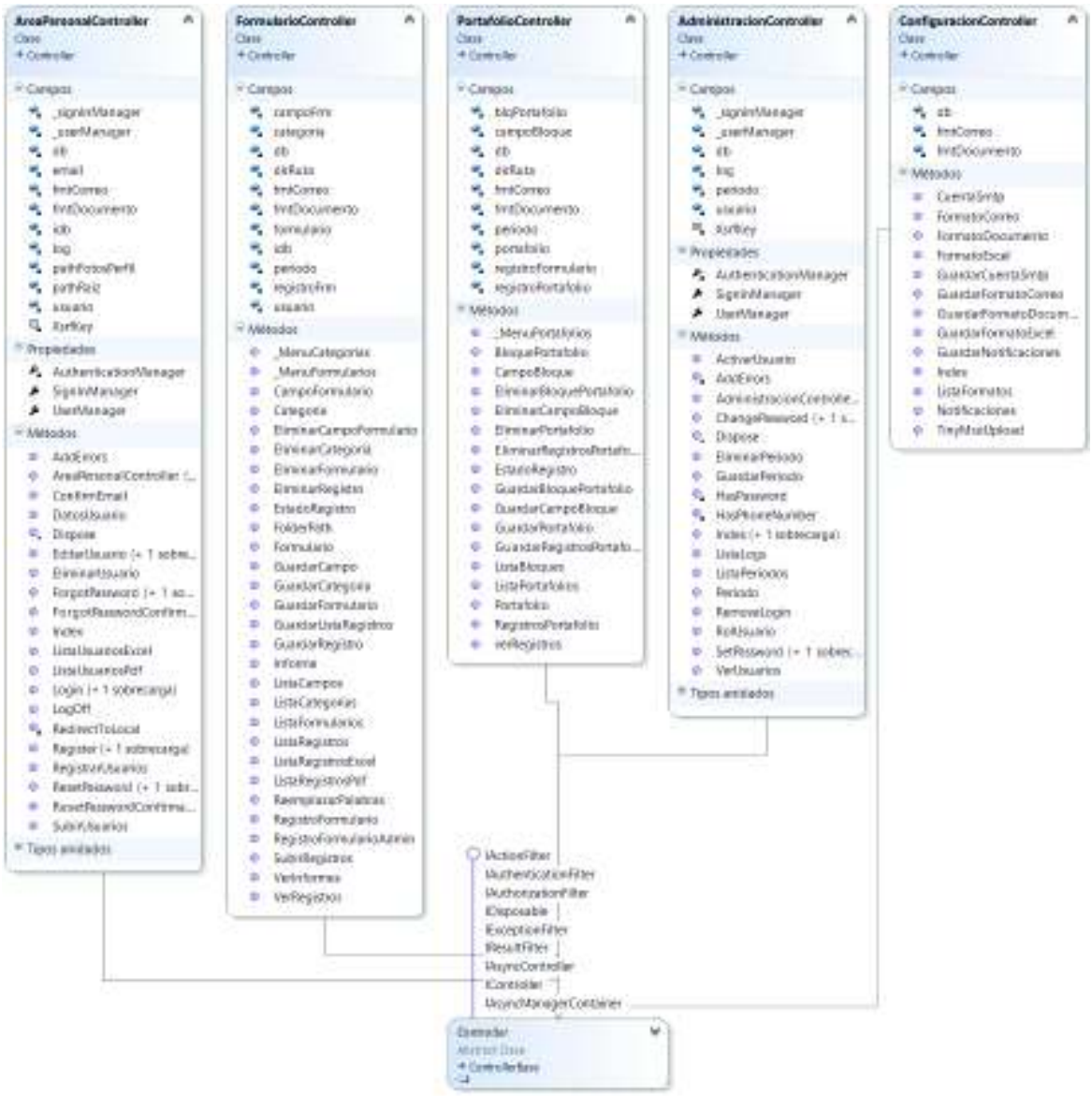


Figura 2.5. Clases del Controlador

El modelo de la aplicación web tiene diferentes clases denominadas entidades, cada entidad representa a una tabla en la base de datos y está compuesta por diferentes atributos los cuales corresponden a las columnas de la base de datos, también se escribieron métodos en cada una de las entidades para consultar, agregar, modificar o eliminar registros. Los métodos agregados son: `Obtener` para conseguir información de un registro específico, `Listar` para obtener una lista de registros, `Guardar` para modificar o agregar un registro y `Eliminar` para eliminar

un registro específico. El modelo también tiene una clase de contexto denominada `AcreditacionContext`, que es la encargada de relacionar las entidades con sus respectivas tablas de la base de datos; además, realiza diferentes tareas como actualizar, crear, obtener y eliminar registros de la base de datos.

A continuación, se presenta una descripción y el diagrama de clase de cada uno de los controladores de la aplicación web:

EL controlador `InicioController`, cuenta con un solo método denominado `Index`, el cual se encarga de presentar la vista pública del inicio de la aplicación web, esta página conducirá a la página del *Login* para la autenticación de los usuarios; este controlador no utiliza ninguna entidad del modelo y no tiene asociaciones con otras clases, por lo tanto, no se presenta su diagrama de clases.

En la Figura 2.6 se presenta el diagrama de clases del controlador `AreaPersonalController`, que cuenta con los métodos, propiedades y atributos necesarios para la autenticación de usuarios, para la gestión de la información personal de los usuarios, para el registro de nuevos usuarios, para el restablecimiento de las contraseñas y para la confirmación de las cuentas. Este controlador tiene los siguientes métodos:

`AddError` se utiliza para agregar un mensaje de error personalizado a la propiedad del modelo.

`Dispose` se utiliza para liberar los recursos no administrados de forma oportuna, que son los objetos que contienen los recursos del sistema operativo como archivos, ventanas, conexiones de red o conexiones de bases de datos, entre otros.

`RedirectToLocal` se utiliza para direccionar a una URL[34].

`Login` y `LogOff` para realizar tareas relacionadas con la autenticación y autorización de los usuarios; y, para cerrar la sesión del usuario.

`DatosUsuario`, `EditarUsuario`, `GuardarUsuario` son utilizados respectivamente para visualizar, modificar y guardar la información de un usuario.

[34] URL (*Uniform Resource Locator*): Permite nombrar los recursos en Internet con un formato estándar (páginas web, archivos, carpetas, entre otros), de tal forma que se puedan localizar mediante una dirección única.

`SubirUsuarios` se utiliza para mostrar una lista de usuarios en una vista los cuales serán registrados y se parte de una lista de usuarios en formato de Excel.

`Register`, `RegistrarUsuarios` y `EliminarUsuario` se utilizan respectivamente para registrar un usuario, para registrar múltiples usuarios a partir de una lista en formato Excel y para eliminar un usuario.

`ForgotPassword` y `ForgotPasswordConfirmation` se utiliza para reestablecer la contraseña de acceso a la cuenta de un usuario y visualizar el mensaje de confirmación.

Finalmente, `ListaUsuariosExcel` y `ListaUsuariosPdf` se utilizan para generar archivos en formato de Excel y PDF con la lista de usuarios registrados en la aplicación web.

El controlador `AreaPersonalController` utiliza las entidades del modelo con sus métodos para crear, recuperar, modificar y eliminar la correspondiente información de la base de datos, por consiguiente, las entidades utilizadas son:

`AcreditacionContext` es la clase de contexto del modelo que se encargada de relacionar las entidades con sus respectivas tablas de la base de datos. `ApplicationDbContext` es otra clase de contexto, utilizada en la autenticación de usuarios, el registro de usuarios, entre otros; y, su funcionalidad es semejante a la clase `AcreditacionContext`, cabe señalar que esta clase se crea automáticamente cuando se crea el proyecto en ASP.NET MVC.

`AspNetUser` es una entidad con los atributos de la información de los usuarios como nombres, apellidos, cédula de identidad, correo electrónico, entre otros.

`FormatosDocumento` es una entidad con los atributos necesarios para establecer/recuperar el formato de los encabezados y los pies de página que se utilizan para generar la lista de usuarios registrado en la aplicación web.

`FormatosCorreo` es una entidad con los atributos para establecer y recuperar el formato del mensaje de correo electrónico de las notificaciones que se envían al momento de registrar una nueva cuenta de un usuario, para reestablecer la contraseña, entre otros.

Finalmente, `Logs` es una entidad con los atributos necesarios que se utilizada para almacenar y recuperar la información de los eventos producidos por los usuarios en la aplicación web.

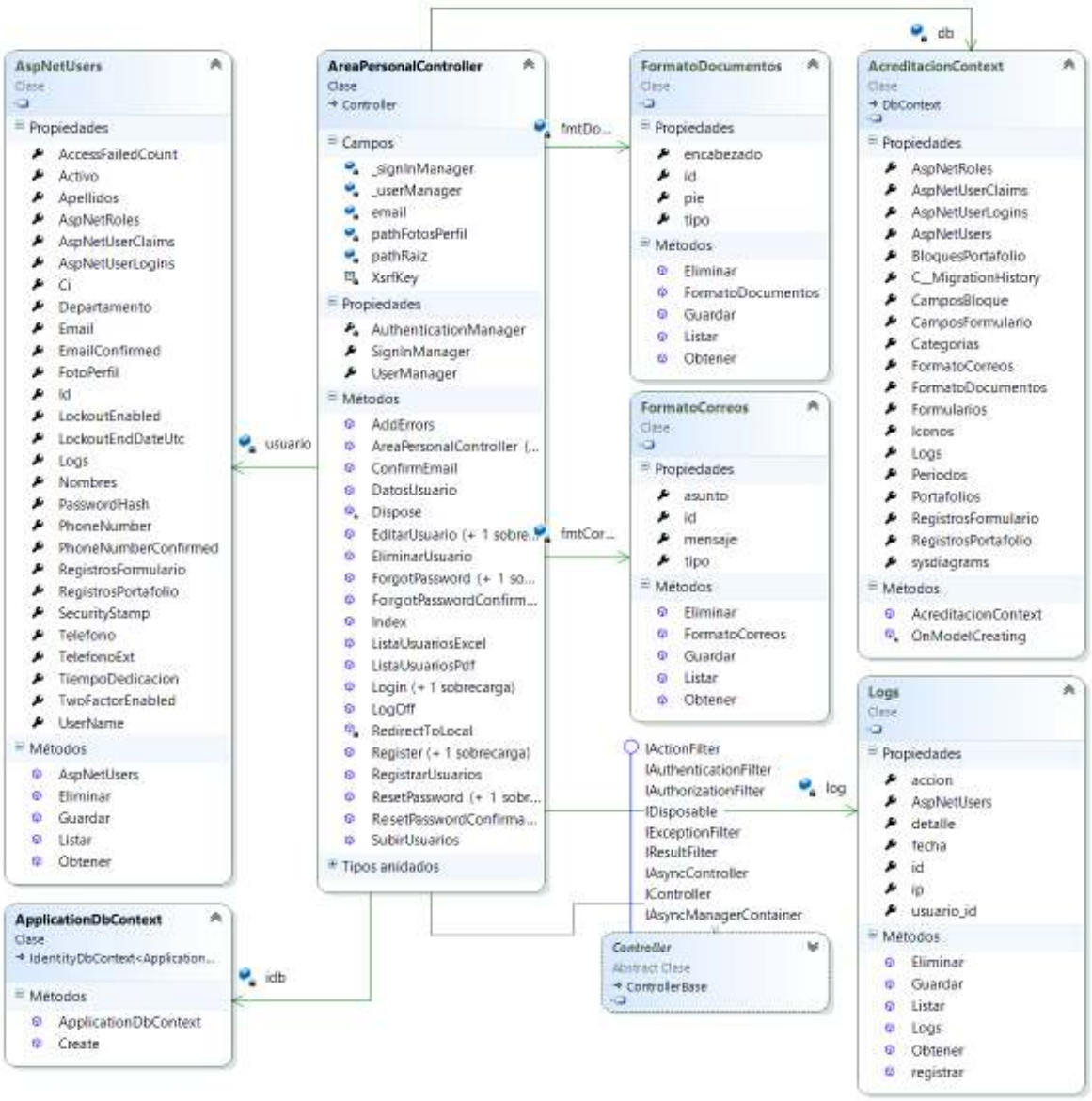


Figura 2.6. Diagrama de clases del controlador `AreaPersonalController`

En la Figura 2.7 se presenta el diagrama de clases del controlador `FormularioController` que cuenta con los métodos y atributos necesarios para gestionar las categorías y los formularios. Las categorías son utilizadas para organizar la información que se solicitará a los profesores mediante formularios. Los formularios permiten recopilar la documentación requerida y son totalmente personalizables permitiendo agregar, modificar y eliminar sus campos a

conveniencia. Se tiene diferentes tipos de campos de tal forma que se pueda ingresar información de varios tipos como es el caso de textos, fechas, archivos, entre otros; la información ingresada por los profesores podrá ser visualizada, aprobada e impresa. Este controlador tiene los siguientes métodos:

`_MenuFormularios` y `_MenuCategorias`, los cuales permiten generar menús y submenús que se ubican en la parte lateral de la página para que los usuarios puedan acceder a cualquier formulario.

`ListaCategorias`, `Categoria`, `GuardarCategoria` y `EliminarCategoria` se encargan de listar las categorías, modificar/agregar, guardar y eliminar.

`ListarFormularios`, `Formulario`, `GuardarFormulario` y `EliminarFormulario` se encargan de mostrar una lista de los formularios creados al administrador, modificar/agregar, guardar y eliminar un formulario; los campos de los formularios disponen de métodos como `ListaCampos`, `CampoFormulario`, `GuardarCampoFormulario` y `EliminarCampoFormulario` para mostrar los campos del formulario, agregar/modificar, guardar y eliminar campos en un formulario.

`ListaRegistros`, `RegistroFormulario`, `GuardarRegistro` y `EliminarRegistro` para mostrar la información ingresada por el profesor, agregar/modificar, guardar y eliminar los registros de dicha información.

`FolderPath` permite localizar la ruta del directorio de una categoría específica.

`RegistroFormularioAdmin` y `SubirRegistros` permiten al administrador ingresar y guardar la información de las asignaturas de los profesores.

`VerRegistros`, `EstadoRegistro` y `HabilitarRegistro` permiten al administrador visualizar los registros ingresados por los profesores, aprobar o rechazar dichos registros; y, habilitar la edición de un registro que previamente fue aprobado.

`Informe` se utiliza para mostrar un resumen con la información que ha ingresado un profesor durante el semestre en cada uno de los formularios, a su vez se podrá imprimir esta información.

`ReemplazarPalabras` se utiliza para reemplazar los códigos o las palabras clave que se escriben en el diseño del formato de los encabezados y de los pies de página de cada uno de los informes que generará la aplicación web, tanto para los profesores como para los administradores; estas palabras clave permitirán agregar información específica y como ejemplo se presentan algunas de estas palabras clave:

- `$Fecha` se reemplazará por la fecha del instante en la que se generó el informe.
- `$Usuario` se reemplazará por los nombres y apellidos del usuario que genera el informe.
- `$Ci` se reemplazará por la cédula de identidad del usuario.
- `$Correo` se reemplazará por la dirección de correo electrónico del usuario.
- `$Categoria y $Formulario` se reemplazarán por el nombre de la categoría y del formulario de los cuales se están imprimiendo los informes.

`GenerarExcel` y `GenerarPdf` para exportar informes de la información ingresada por los profesores en formato Excel o PDF con su respectivo encabezado y pie de página que fue especificado por el súper administrador.

El controlador `FormularioController` utiliza las entidades del modelo en sus métodos para crear, recuperar, modificar y eliminar la correspondiente información de la base de datos, las principales entidades utilizadas son:

`Categorias` es una entidad utilizada para almacenar/recuperar categorías.

`Formularios` y `CamposFormulario` son entidades empleadas para crear/recuperar formularios.

`RegistrosFormulario` es una entidad empleada para almacenar y recuperar los registros de la documentación del profesor.

`FormatosDocumento` y `FormatosCorreo` se utilizan para establecer/recuperar los formatos de los informes y de los mensajes de correo electrónico.

Finalmente, `Periodos` se utiliza para recuperar la información de los periodos académicos.

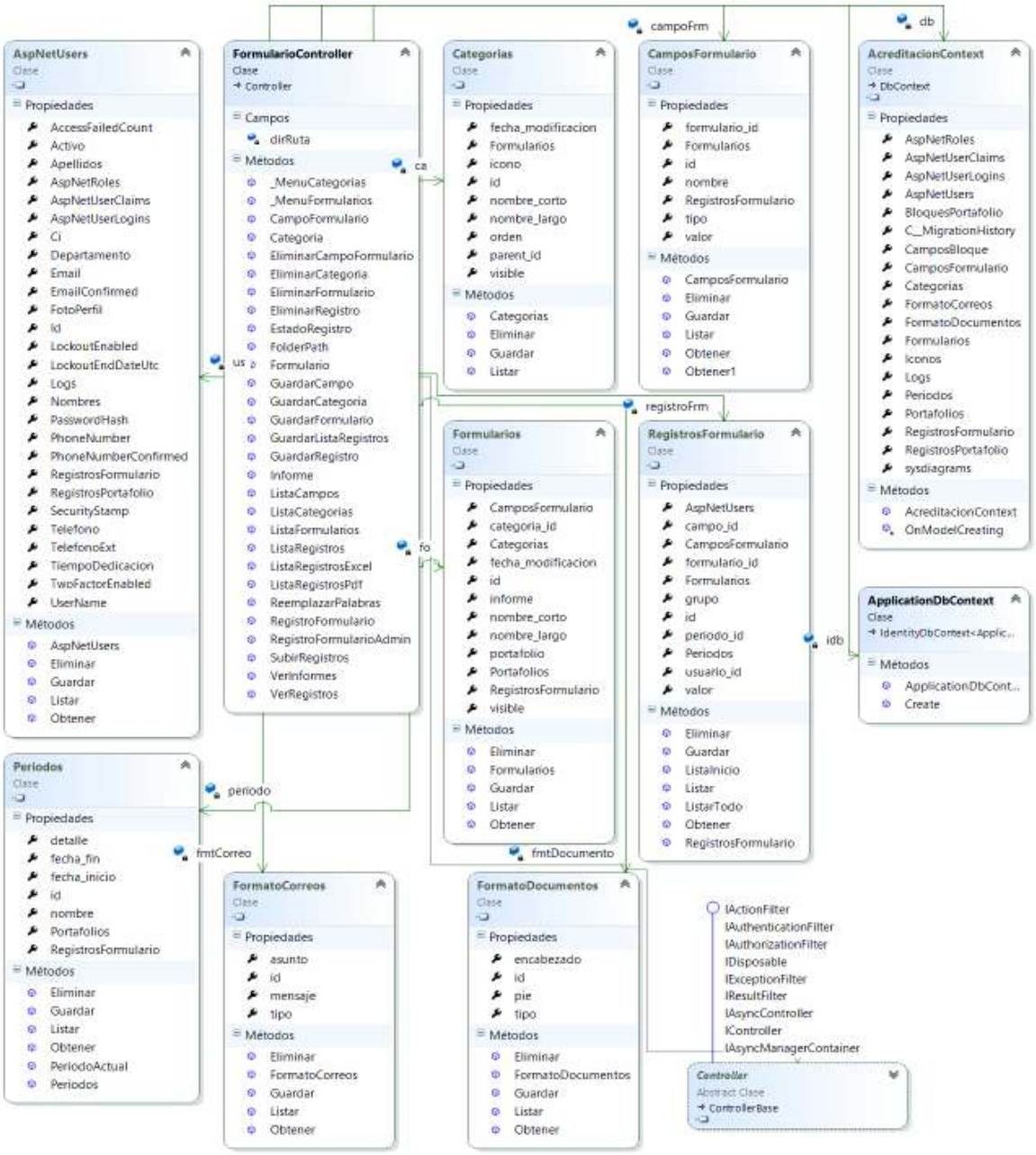


Figura 2.7. Diagrama de clases del controlador `FormularioController`

En la Figura 2.8 se presenta el diagrama de clases del controlador `PortafolioController` que cuenta con los métodos y atributos necesarios para gestionar la documentación requerida de cada una de las asignaturas que imparten los profesores, por lo que cada asignatura tendrá su propio portafolio académico que está compuesto por varios bloques, los cuales tendrán diferentes campos de

acuerdo a la información requerida por el administrador. Este controlador tiene los siguientes métodos:

`ListaPortafolios`, `Portafolio`, `GuardarPortafolio` y `EliminarPortafolio`, los cuales se encargan de listar los portafolios existentes, modificar/agregar, guardar y eliminar un portafolio.

`ListarBloques`, `BloqueFormulario`, `GuardarBloquePortafolio` y `EliminarBloquePortafolio` se encargan de mostrar los bloques de cada portafolio, modificar/agregar, guardar y eliminar cada uno de estos bloques; los bloques están formados por diferentes campos que tienen los métodos: `CampoBloque`, `GuardarCampoBloque` y `EliminarCampoBloque` para agregar/modificar, guardar y eliminar campos en un bloque específico.

`RegistrosPortafolio`, `GuardarRegistroPortafolio` y `EliminarRegistroPortafolio` que son utilizados por los profesores para mostrar la documentación ingresada por ellos, agregará/modificará, guardará o eliminará cada uno de los registros correspondientes a la documentación ingresada.

El controlador `PortafolioController` utiliza las entidades del modelo con sus métodos para crear, recuperar, modificar y eliminar la correspondiente información de la base de datos, por lo que las entidades utilizadas son:

`AcreditacionContext` y `ApplicationDbContext` son las clases de contexto.

`Portafolios`, `BloquesPortafolio` y `CamposBloque` son utilizadas para crear y recuperar los portafolios de las asignaturas.

`RegistrosPortafolio` se utiliza para almacenar y recuperar los registros de un portafolio.

`Periodos` se utiliza para recupera los periodos académicos.

`FormatosDocumento` y `FormatosCorreo` se utilizan para establecer/recuperar los formatos de informes y de correos.

`Logs` que se utiliza para almacenar la información de los eventos de cada usuario.

Figura 2.8. Diagrama de clases del controlador `PortafolioController`

En la Figura 2.9 se presenta el diagrama de clases del controlador `AdministracionController` que cuenta con los métodos y atributos

necesarios para gestionar las cuentas de los usuarios y para gestionar los periodos académicos. Dispone de los siguientes métodos:

`ListaUsuarios` se utiliza para mostrar todos los usuarios registrados en la aplicación web con su respectiva información como son nombres, apellidos, cédula de identidad, correo electrónico, rol, estado, entre otra información.

`RolUsuario` permite cambiar el rol de un usuario; los roles que se tienen son: profesor, administrador o súper administrador.

`ActivarUsuario` que permite activar/desactivar a un usuario de la aplicación web, de tal forma que solo los usuarios que estén activos podrán subir la información requerida por el administrador, tanto en los formularios como en los portafolios.

`ListaPeriodos`, `Periodo`, `GuardarPeriodo`, `EliminarPeriodo` para mostrar una lista de los periodos académicos existentes, agregar/modificar, guardar y eliminar un periodo académico.

`ListaLogs` para presentar al administrador una lista con las actividades realizadas por cada usuario en la aplicación web.

Finalmente, `ChangePassword` y `SetPassword` son utilizados para que el usuario pueda modificar su clave personal de acceso a la aplicación web.

El controlador `AdministracionController` utiliza las entidades del modelo para crear, recuperar, modificar y eliminar la correspondiente información de la base de datos, las entidades utilizadas son:

`AcreditacionContext`, `ApplicationDbContext` son las clases de contexto.

`AspNetUser` es utilizado para recuperar o almacenar la información de los usuarios registrados.

`Periodo` es una entidad utilizada para almacenar o recuperar los periodos académicos de la base de datos.

Finalmente, `Logs` se utiliza para almacenar o para recuperar los eventos de las actividades realizadas por los usuarios en la aplicación web.

Todas las entidades tienen métodos para listar, obtener, guardar/modificar o eliminar registros de la base de datos; sin embargo, a la entidad `Logs` tiene el método `Registro` que es utilizado para almacenar los eventos de las actividades realizadas por los usuarios en la aplicación web, se agrega este método en su entidad con la finalidad de reutilizar su código, evitando agregar el método en cada uno de los controladores o evitando desarrollar una clase con este método para registrar los eventos de los usuario.

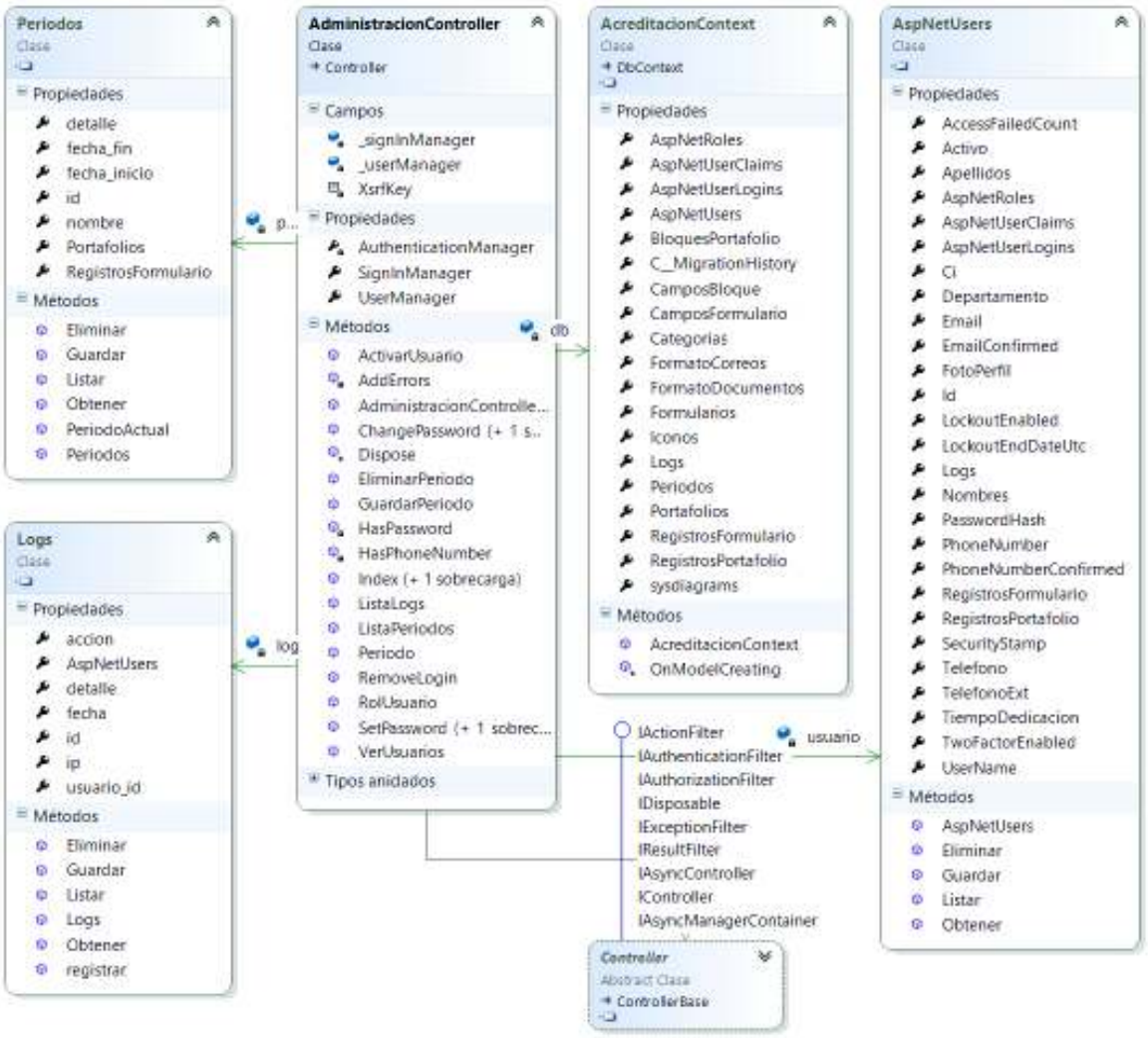


Figura 2.9. Diagrama de clases del controlador `AdministracionController`

En la Figura 2.10 se presenta el diagrama de clases del controlador `ConfiguracionController` que tiene los métodos y atributos necesarios para realizar ciertas configuraciones dentro de la aplicación web, por lo que se tienen los siguientes métodos:

`ListaFormatos` muestra una lista de los formatos que tienen los mensajes de correo electrónico y los documentos de informes que se generarán.

`FormatoCorreo` y `GuardadFormatoCorreo` permiten modificar los mensajes de los correos electrónicos y guardar los cambios.

`FormatoDocumento` y `GuardarFormatoDocumento` permiten modificar los encabezados y los pies de las páginas de los documentos de informes que se generarán; y, guardar los cambios.

`Notificacion` y `GuardarNotificacion` permitirán establecer los días y la hora en las que se enviarán los mensajes de notificaciones; y, guardar los cambios.

`CuentaSmtp` permitirá modificar la información de la cuenta de correo electrónico utilizada para enviar mensajes de correo electrónico a través del protocolo SMTP[35].

Finalmente, `TinyMceUpload` se utiliza para almacenar las imágenes que se insertan en el editor de texto para establecer los formatos del correo electrónico y de los informes.

El controlador `ConfiguracionController` utiliza las entidades del modelo para crear, recuperar, modificar y eliminar la correspondiente información de la base de datos, por lo que las entidades utilizadas son:

`AcreditacionContext`, `ApplicationDbContext` son las clases de contexto.

`Logs` se utiliza para almacenar los eventos de las actividades realizadas por los usuarios.

`FormatoDocumentos` se utiliza para establecer los encabezados y pies de página en los informes.

Finalmente, `FormatoCorreos` se utiliza para establecer los formatos de los correos electrónicos que se enviarán como notificaciones a los profesores con la información sobre la documentación que deben ingresar y sus registros de la documentación que fue rechazada. A su vez, permite establecer los encabezados

[35] SMTP (*Simple Mail Transfer Protocol*): Es un protocolo de red que se utiliza para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos.

de los mensajes que se envían con el registro de nuevas cuentas de usuarios, cuando se solicita el restablecimiento de la contraseña, entre otros.

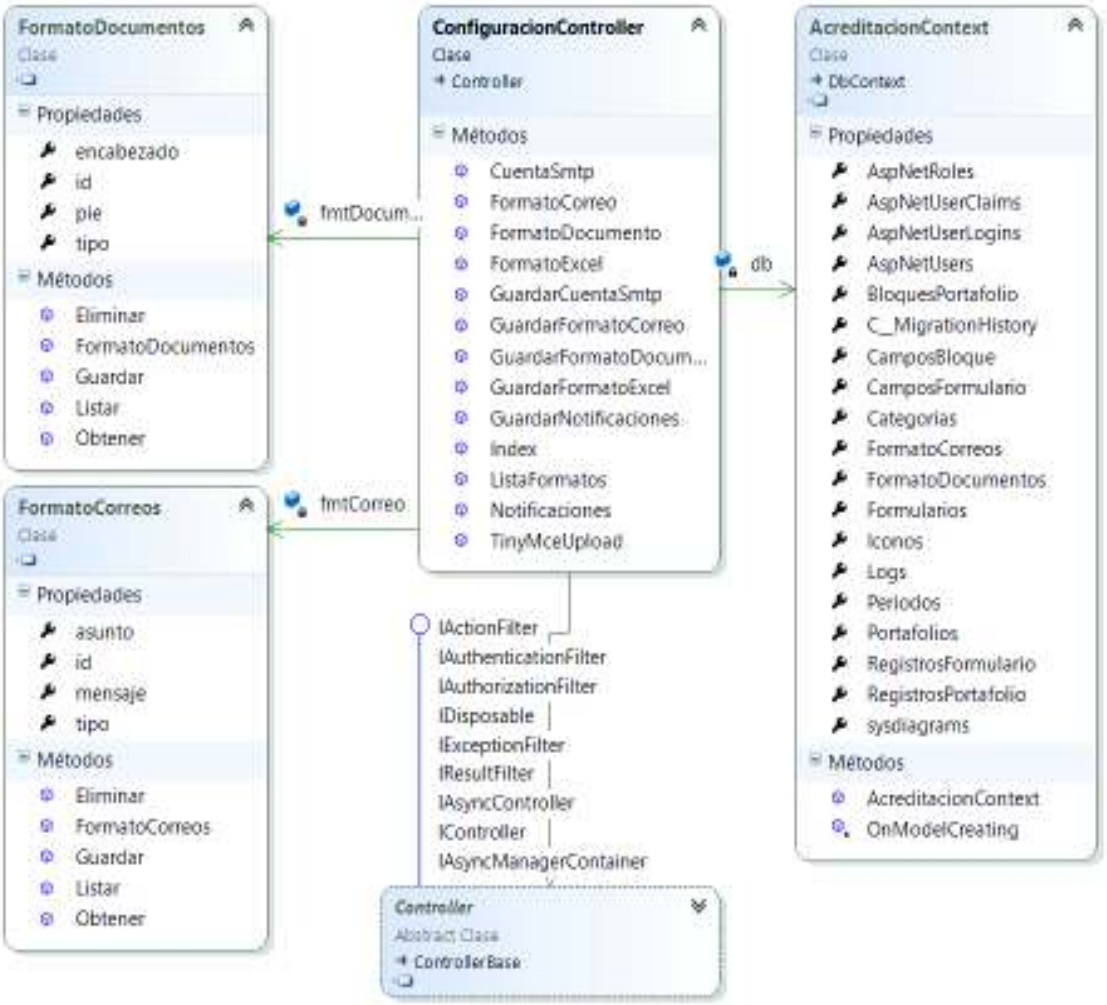


Figura 2.10. Diagrama de clases del controlador `ConfiguracionController`

2.2.4.3 Diagrama entidad – relación

Es un modelo de datos que permite representar cualquier abstracción, percepción y conocimiento de un sistema de información mediante un conjunto de objetos denominados entidades que se relacionan y tienen una lista de sus atributos.

En la Figura 2.11 se tiene el diagrama entidad-relación de la aplicación web, que cuenta con varias entidades, cada una con sus respectivos atributos y sus relaciones como se puede observar. A continuación, se realiza una descripción del diagrama entidad – relación.

La aplicación web cuenta con las entidades `AspNetUser` para almacenar o recuperar información de los usuarios, `AspNetRoles` para almacenar los roles de los usuarios y `AspNetUserRoles` para asociar a cada usuario con un rol específico; estas entidades son generadas automáticamente por ASP.NET Identity[36], pero se deben hacer algunas modificaciones en la tabla `AspNetUser` para incluir más columnas con la información necesaria de cada uno de los usuario como: nombres, apellidos, número de cédula, entre otros.

La entidad `Categoría` se asocia con la entidad `Formulario`, donde cada categoría puede tener muchos formularios, estableciéndose una cardinalidad de uno a muchos.

La entidad `Formulario` se asocia con la entidad `Campo` con una cardinalidad uno a muchos por lo que cada formulario puede estar compuesto por varios campos; la entidad `Campo` se asocia con la entidad `RegistroFormulario` que tienen una cardinalidad de uno a muchos y cada `RegistroFormulario` se asocia con la entidad `AspNetUser` con una cardinalidad de muchos a uno, por lo que cada usuario puede ingresar varios registros con la documentación solicitada en cada formulario.

La entidad `Portafolio` se asocia con la entidad `BloquePortafolio` con una cardinalidad de uno a muchos; cada `BloquePortafolio` se asocia con la entidad `CampoBloque` con una cardinalidad de uno a muchos, por lo que cada bloque puede estar compuesto por varios campos; la entidad `CampoBloque` se asocia con la entidad `RegistroPortafolio` que tienen una cardinalidad de uno a muchos y cada `RegistroPortafolio` se asocia con la entidad `AspNetUser` con una cardinalidad de muchos a uno.

La entidad `RegistroPortafolio` se asocia con la entidad `RegistroFormulario` con una cardinalidad de muchos a uno, debido a que cada profesor puede ingresar varios registros en el portafolio de cada una de las asignaturas que tiene a su cargo.

---

[36] ASP.NET Identy: Es un sistema de autenticación y autorización que se puede utilizar para *Web Forms*, *Web Pages*, MVC, API, entre otros.

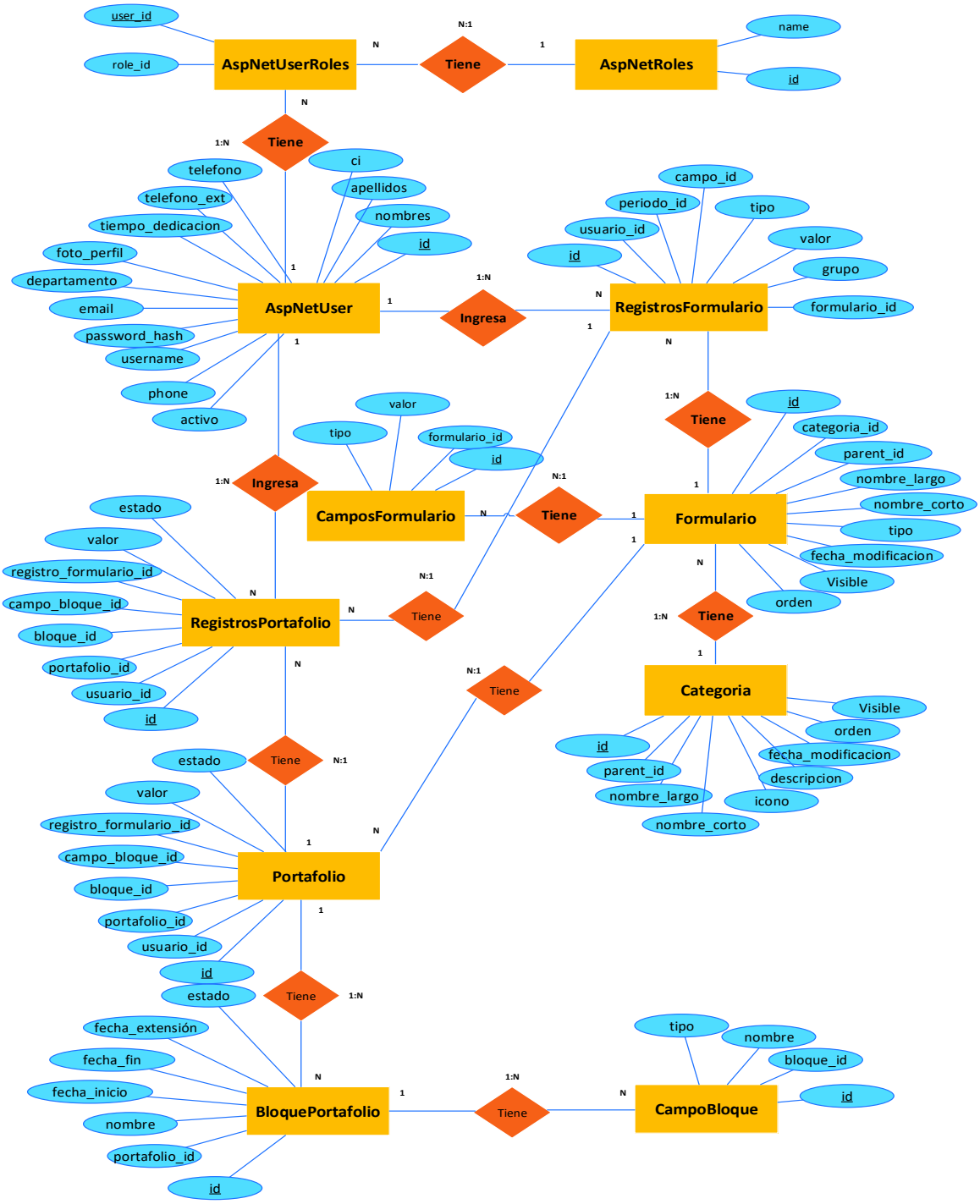


Figura 2.11. Diagrama de entidad – relación

2.2.4.4 Diagrama relacional

Es un modelo de organización y gestión de bases de datos que consistente en el almacenamiento de datos en diferentes tablas que se relacionan, están compuestas por filas y columnas. En la Figura 2.12 se presenta el diagrama relacional de la aplicación web que cuenta con diferentes tablas que se desarrollaron de acuerdo a

los requerimientos obtenidos. Cabe recalcar que las tablas `AspNetUser`, `AspNetRoles` y `AspNetUserRoles` son generadas automáticamente por ASP.NET Identity, pero se agregaron columnas en la tabla `AspNetUser` para almacenar cierta información importante sobre los usuarios.

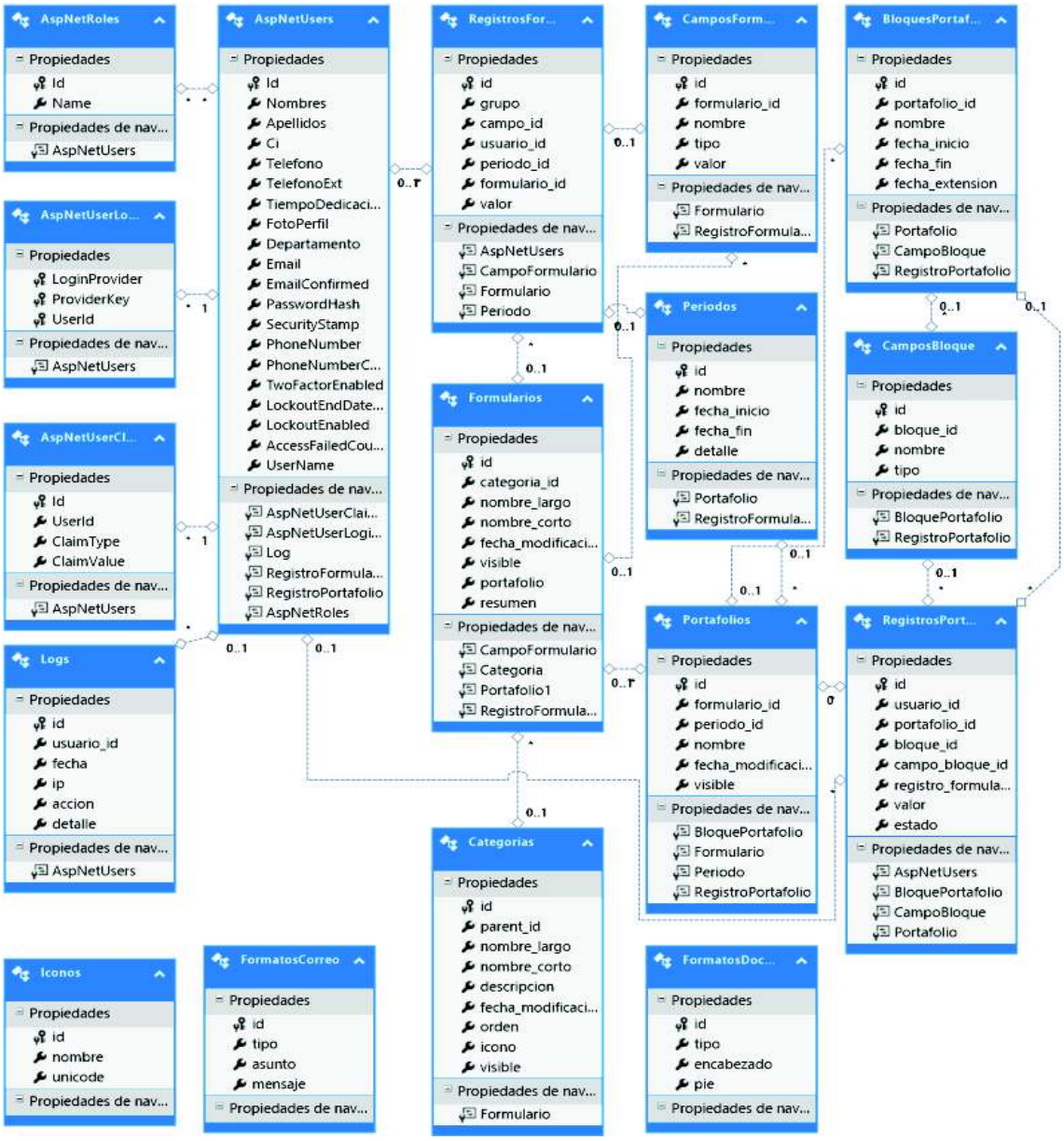


Figura 2.12. Diagrama relacional

A continuación, se realiza una breve descripción de los atributos de las diferentes tablas.

`AspNetUser`: `Id` es su clave primaria con un tipo de dato `nvarchar` de 128 caracteres; `Nombres`, `Apellidos` y `FotoPerfil` son de tipo de dato `nvarchar`

de 50 caracteres; Ci y Telefono son de tipo de dato nvarchar de 10 caracteres; TelefonoExt es de tipo de dato nvarchar de 6 caracteres; TiempoDedicacion es de tipo de dato nvarchar de 40 caracteres; Departamento es de tipo de dato nvarchar de 200 caracteres; Email y UserName son de tipo de dato nvarchar de 256 caracteres; PasswordHash y PhoneNumber son de tipo de dato nvarchar máximo; y, Activo es de tipo de dato bit.

AspNetRoles: Id es su clave primaria con un tipo de dato nvarchar de 128 caracteres y Name tiene un tipo de dato nvarchar de 256 caracteres.

AspNetUserRoles: es una tabla intermedia con sus atributos UserId y RoleId que son de tipo de dato nvarchar de 128 caracteres.

Categorias: id es su clave primaria con un tipo de dato entero que se autoincrementa, parent_id es de tipo de dato entero, nombre_largo es de tipo de dato varchar de 200 caracteres, nombre_corto es de tipo de dato varchar de 100 caracteres, fecha_modificación es de tipo de dato date, orden es de tipo de dato entero, icono es de tipo de dato varchar de 50 caracteres y visible es de tipo de dato booleano.

Formularios: id es su clave primaria con un tipo de dato entero que se autoincrementa; categoria_id es una clave foránea que relaciona la tabla Formularios con la tabla Categorias; nombre_largo y nombre_corto son de tipo de dato varchar de 200 y 100 caracteres; fecha_modificacion es tipo de dato date; y, visible, portafolio e informe son de tipo de dato booleano.

CamposFormulario: id es su clave primaria con un tipo de dato entero que se autoincrementa; formulario_id es una clave foránea de tipo de dato entero que se relaciona con la tabla Formularios; y, nombre, tipo y valor son de tipo de dato varchar de 100, 50 y 200 caracteres.

RegistrosFormulario: id es su clave primaria con un tipo de dato entero que se autoincrementa; grupo es de tipo de dato entero; campo_id, periodo_id y formulario_id son claves foráneas de tipo de dato entero utilizadas para

relacionar con las tablas CamposFormulario, AspNetUser, Periodos y Formularios; y, usuario_id es una clave foránea de tipo de dato nvarchar de 128 caracteres para relacionar con la tabla AspNetUser.

Portafolios: id es su clave primaria con un tipo de dato entero que se autoincrementa; formulario_id y periodo_id son claves foráneas de tipo de dato entero utilizadas para relacionarse con las tablas Formularios y Periodos; fecha_modificacion que es de tipo de dato date; y, visible es de tipo de dato booleano.

BloquesPortafolio: id es su clave primaria con un tipo de dato entero que se autoincrementa; portafolio_id es una clave foránea de tipo de dato entero; nombre es de tipo de dato varchar de 100 caracteres; y, fecha_inicio, fecha_fin y fecha_extension son columnas de tipo de dato date.

CamposBloque: id es su clave primaria con un tipo de dato entero que se autoincrementa; bloque_id es una clave foránea de tipo de dato entero; y, nombre y tipo son de tipo de dato varchar de 50 caracteres.

RegistrosPortafolio: id es su clave primaria con un tipo de dato entero que se autoincrementa; usuario_id es una clave foránea de tipo de dato nvarchar de 128 caracteres; portafolio_id, bloque_id, campo_bloque_id y registro_formulario_id son claves foráneas de tipo de dato entero; y, valor y estado son de tipo de dato varchar de 400 y 15 caracteres.

Periodos: id es su clave primaria con un tipo de dato entero que se autoincrementa; nombre es de tipo de dato varchar de 50 caracteres; y, fecha_inicio y fecha_fin son columnas de tipo de dato date.

Logs: id es su clave primaria con un tipo de dato entero que se autoincrementa; usuario_id es una clave foránea de tipo de dato nvarchar de 128 caracteres; fecha que es de tipo de dato date; e, ip, accion y detalle son de tipo de dato varchar de 40, 50 y 500 caracteres.

Iconos: id es su clave primaria con un tipo de dato entero que se autoincrementa; nombre y unicode son de tipo de dato varchar de 50 y 15 caracteres.

`FormatoDocumentos`: `id` es su clave primaria con un tipo de dato entero que se autoincrementa; `tipo` es de tipo de dato `varchar` de 100 caracteres; y, `encabezado` y `pie` son de tipo de dato `varchar` de tamaño máximo.

`FormatoCorreos`: `id` es su clave primaria con un tipo de dato entero que se autoincrementa; `tipo` y `asunto` son de tipo de dato `varchar` de 100 y 500 caracteres; y, `mensaje` es de tipo de dato `varchar` de tamaño máximo.

2.2.5 MOCKUPS DE LA APLICACIÓN WEB

Los *mockups* son bosquejos de la interfaz de usuario de la aplicación web. A continuación, se presentarán los principales *mockups* con una pequeña descripción de cada uno de ellos, además los *mockups* se han clasificado de acuerdo al rol del cada usuario.

2.2.5.1 Mockups de las páginas públicas

La aplicación web cuenta con varias páginas públicas, las cuales pueden ser visualizadas por cualquier persona; se tendrá una página principal que permitirá ir a la página del *Login*, en donde se autenticarán los usuarios.

En la Figura 2.13 se presenta el *mockup* de la página principal de la aplicación web, la cual tendrá una imagen de fondo, en el centro de la pantalla estará ubicado un botón que llevará a la página de *Login*.

Figura 2.13. *Mockup* de la página principal

En la Figura 2.14 se muestra el *mockup* de la página del L*ogin*, está compuesto por dos cuadros de texto en donde se tiene que ingresar el usuario (correo electrónico) y la contraseña respectivamente; se tiene una opción para mantener iniciada la sesión del usuario; se tiene un enlace que lleva a la página para recuperar la contraseña; finalmente, se tiene un botón que permitirá autenticar las credenciales ingresadas del usuario.

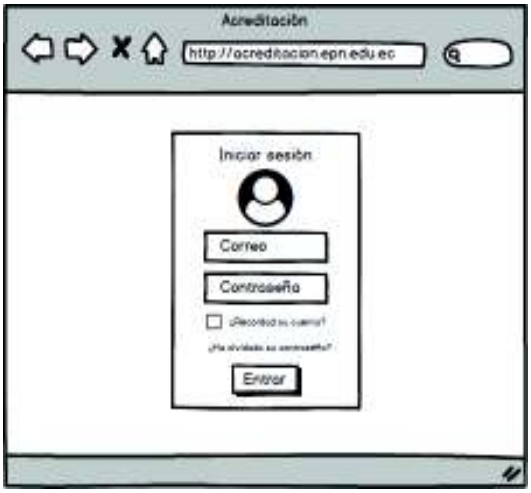


Figura 2.14. *Mockup* de la página del *Login*

En la Figura 2.15 se muestra el *mockup* de la página para reestablecer la contraseña de un usuario; la cual está compuesta por un cuadro de texto para ingresar el correo electrónico y un botón para validar la identidad del usuario, a su vez, se enviará un correo electrónico con las instrucciones y enlace que conduce a una página para ingresar una nueva contraseña.

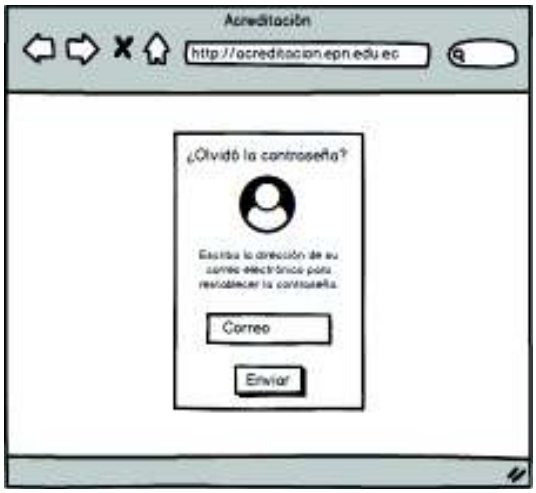


Figura 2.15. *Mockup* de la página para reestablecer la contraseña

2.2.5.2 Mockups de las páginas de los profesores

Los usuarios una vez autenticados como profesores tendrán acceso a diferentes páginas, en donde podrán visualizar y modificar su información personal; así como podrán ingresar, visualizar e imprimir sus informes de la documentación solicitada por los administradores que fue ingresada y aprobada.

En la Figura 2.16 se presenta el *mockup* de la página de inicio de un usuario autenticado, aquí se visualizan algunos bloques que tienen un resumen de la documentación ingresada en cada formulario, se visualizará un indicador en la parte superior izquierda de los registros ingresados y de los registros aprobados por el administrador; además, se tendrá un enlace que llevará al respectivo formulario en el cual se tendrá que ingresar la información solicitada.

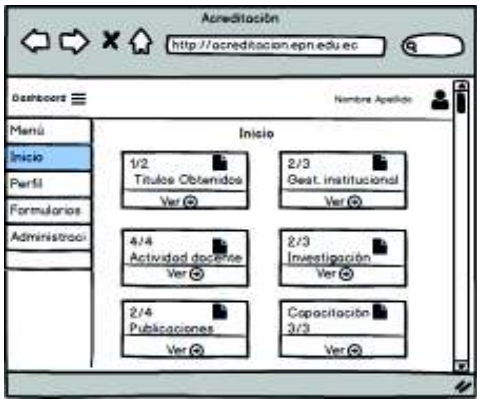


Figura 2.16. *Mockup* de la vista inicial del profesor

En la Figura 2.17 se presenta el *mockup* de la página que muestra la información personal de cada usuario, tiene botones que permitirá modificar su información personal o cambiar la contraseña.

Figura 2.17. *Mockup* de la información personal del usuario

En la Figura 2.18 se muestra el *mockup* de la página que muestra los registros de un formulario, tiene un menú para visualizar la información de un periodo específico, botones para generar un archivo PDF o de Excel; se tiene la lista de registros ingresados por el profesor en el periodo correspondiente; además, cuenta con las opciones para agregar, modificar o eliminar los registros.

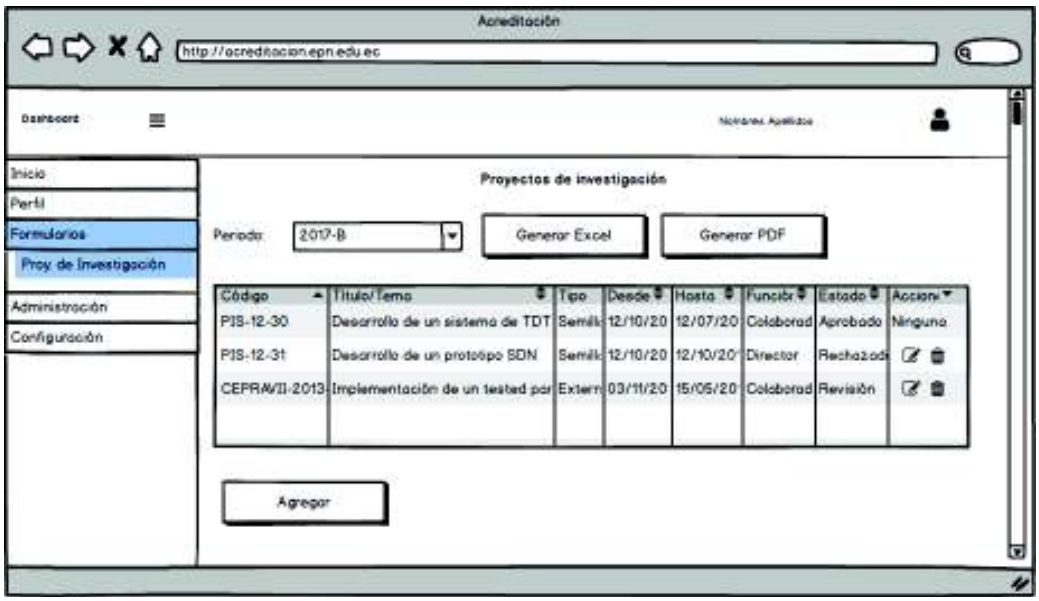


Figura 2.18. *Mockup* de la información de un formulario

En la Figura 2.19 se muestra el *mockup* de la página de un formulario donde se ingresará la información solicitada por el administrador, cada formulario está compuesto por un conjunto de campos especificados por el administrador de acuerdo a la información requerida; los campos son de diferentes tipos para permitir el ingreso de diferente información como textos, fechas, archivos, entre otros.

Figura 2.19. *Mockup* de un formulario

En la Figura 2.20 se presenta el *mockup* de la página que muestra una lista de bloques que corresponden a un portafolio académico que se utilizará en cada una de las asignaturas que imparten los profesores; los bloques tendrán diferentes tipos de campos de acuerdo a la información que se requiera; donde los campos pueden ser textos, fechas, archivos, entre otros.

Los profesores tienen la opción de ingresar, modificar y eliminar la información ingresada; además, cada bloque tiene un rango de fechas para ingresar la documentación solicitada durante el periodo especificado, caso contrario se mostrará un mensaje indicándole que está fuera de la fecha para ingresar la documentación solicitada.

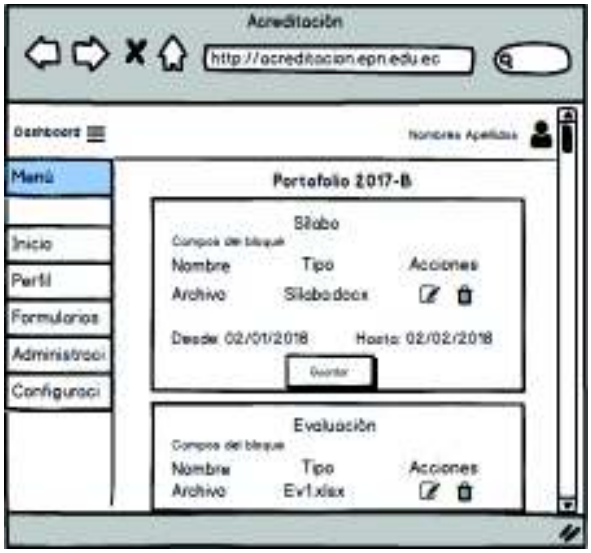


Figura 2.20. *Mockup* del portafolio de una asignatura

2.2.5.3 Mockups de las páginas del administrador

El administrador tiene el control para gestionar las diferentes categorías, formularios, portafolios, usuarios y periodos; además, tiene la capacidad de validar la documentación ingresada por los profesores. A continuación, se presentan los *mockups* de las páginas del administrador:

En la Figura 2.21 se presenta el *mockup* que muestra una lista de categorías con sus respectivas opciones, las cuales que se han creado en la aplicación web para organizar los formularios y la información que se ingresará; la página cuenta con los botones utilizados para agregar, modificar o eliminar las categorías.

Figura 2.21. *Mockup* de la lista de categorías

En la Figura 2.22 se muestra el *mockup* de la página del formulario utilizado para agregar o modificar las categorías; además, permite agregar subcategorías mediante la opción de categoría principal.

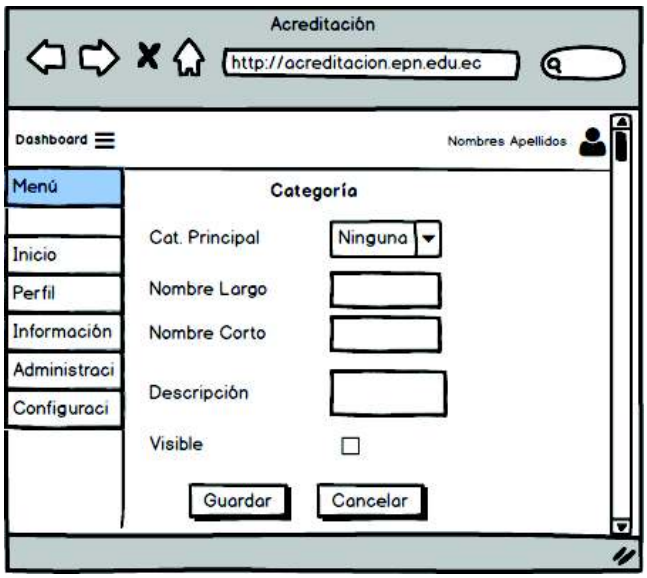


Figura 2.22. *Mockup* del formulario de las categorías

En la Figura 2.23 se presenta el *mockup* de la página que contiene una lista de los formularios de la aplicación web; la página contará con las funciones básicas para agregar, modificar y eliminar los formularios; también cuenta con un botón para ir a la página que muestra la lista de campos que tiene cada uno de los formularios.

Figura 2.23. *Mockup* de la lista de formularios

En la Figura 2.24 se muestra el *mockup* de la página que se utiliza para agregar o modificar los formularios de la aplicación web, cabe señalar que cada formulario está organizado en categorías.

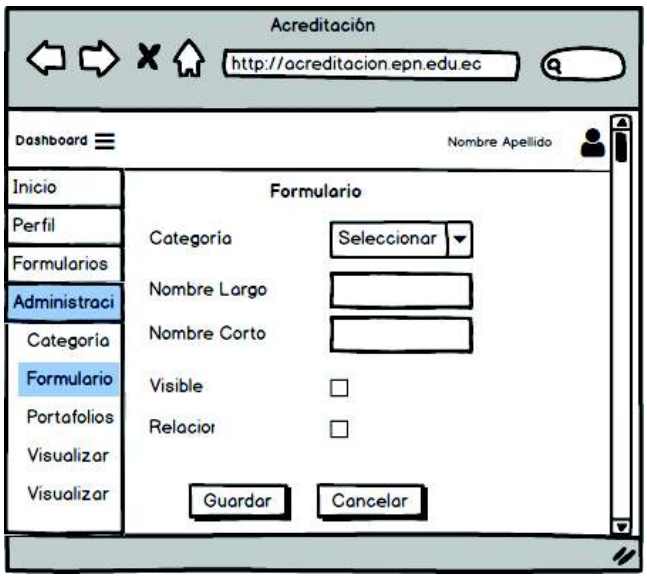


Figura 2.24. *Mockup* de la página para agregar o modificar formularios

En la Figura 2.25 se presenta el *mockup* de la lista de los campos de cada formulario de la aplicación web, estos campos son totalmente personalizables porque se pueden agregar, modificar o eliminar; cabe indicar que se tienen campos de diferentes tipos, de acuerdo a la información que se requiera, por lo que se tienen campos de textos, fechas, archivos, entre otros.

Figura 2.25. *Mockup* de la lista de campos de un formulario

En la Figura 2.26 se muestra el *mockup* de la página que presenta una lista con los registros de la documentación ingresada por los profesores en cada formulario, cuenta con las opciones para visualizar la información de otro formulario o de visualizar la información de otro periodo académico; los registros cuentan con las opciones para validar los registros o para habilitar su edición en el caso en que el registro esté aprobado; además, tiene las opciones para generar un archivo de informe con el formato de Excel o de PDF.

Figura 2.26. *Mockup* de los registros de un formulario

En la Figura 2.27 se muestra el *mockup* de la lista de portafolios con sus respectivas opciones para agregar, modificar o eliminar; además, se tiene un botón que lleva a una página en donde se puede agregar bloques y campos al portafolio.

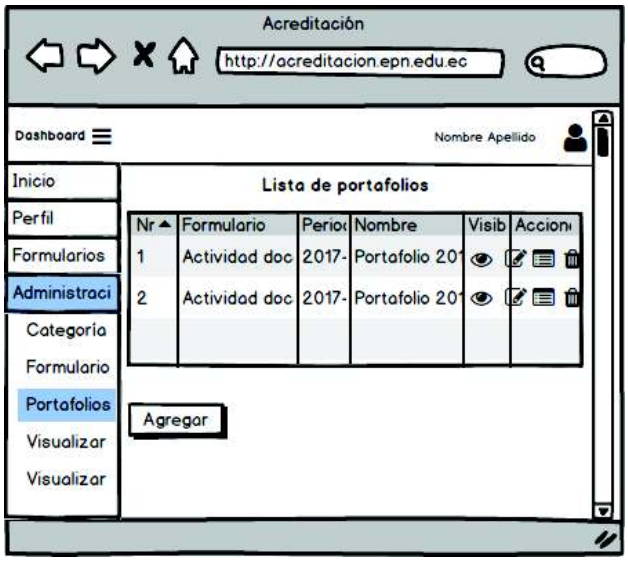


Figura 2.27. *Mockup* de la lista de portafolios

En la Figura 2.28 se muestra el *mockup* del formulario utilizado para agregar o modificar un portafolio.

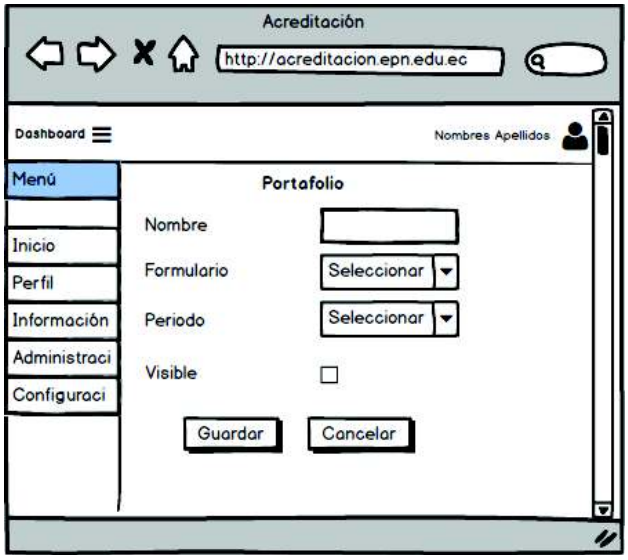


Figura 2.28. *Mockup* del formulario de un portafolio

En la Figura 2.29 se muestra el *mockup* de los bloques de un portafolio; donde el administrador podrá agregar bloques como el sílabo, evaluaciones, entre otros; a cada bloque se le podrá agregar diferentes campos como textos, archivos, entre

otros, con la finalidad en la que los profesores puedan ingresar la información solicitada.

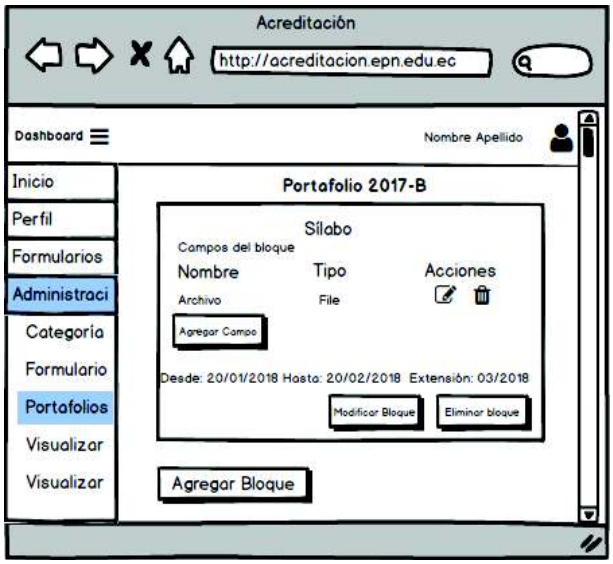


Figura 2.29. *Mockup* de los bloques de un portafolio

En la Figura 2.30 se muestra el *mockup* de una lista con todos los registros ingresados al portafolio en cada campo, donde los administradores podrán validar dicha información o volverla a habilitar para que el profesor pueda ingresar de nuevo la información requerida; además, se contará con la opción para generar un informe de la información ingresada por los profesores en formato de Excel o PDF.

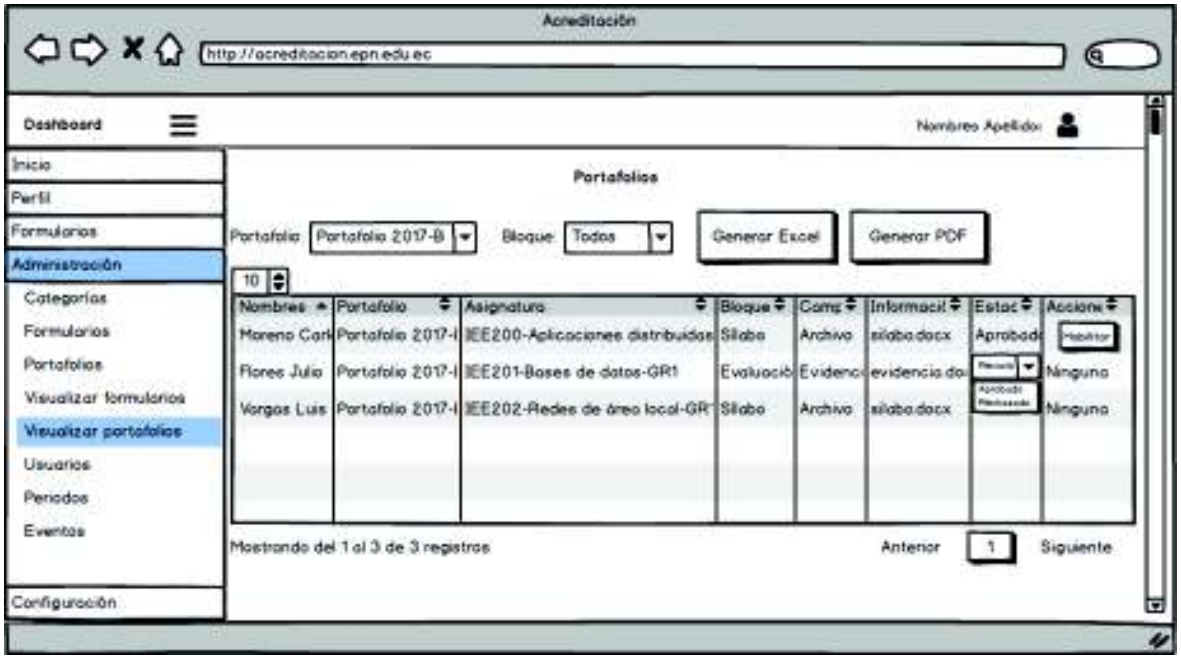


Figura 2.30. *Mockup* de los registros de los bloques de un portafolio

En la Figura 2.31 se muestra el *mockup* de la lista de periodos académicos con sus respectivas opciones para agregar, modificar o eliminar los periodos académicos.

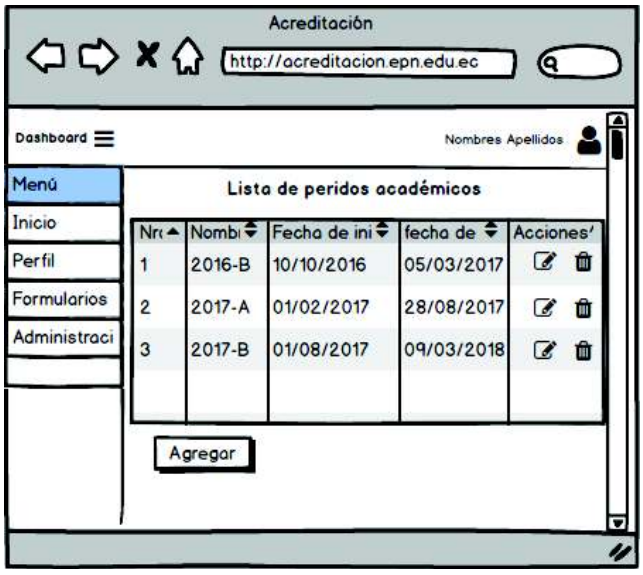


Figura 2.31. *Mockup* de la lista de periodos académicos

En la Figura 2.32 se muestra el *mockup* del formulario utilizado para agregar o modifica un periodo académico.

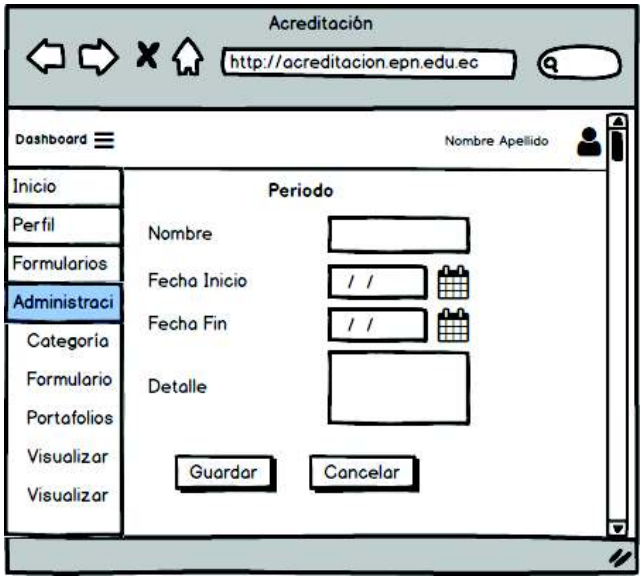


Figura 2.32. *Mockup* del formulario de un periodo académico

2.2.5.4 Mockups de las páginas del súper administrador

El súper administrador tiene el control para gestionar los formatos de los documentos, la cuenta de correo electrónico y las notificaciones; además cuenta

con los privilegios para gestionar las categorías, los formularios, los portafolios, los usuarios y los periodos académicos. A continuación, se presentan los *mockups* de las páginas del súper administrador.

En la Figura 2.33 se muestra el *mockup* del formulario que permite dar formato a los correos electrónicos que serán enviados a los profesores con las notificaciones.

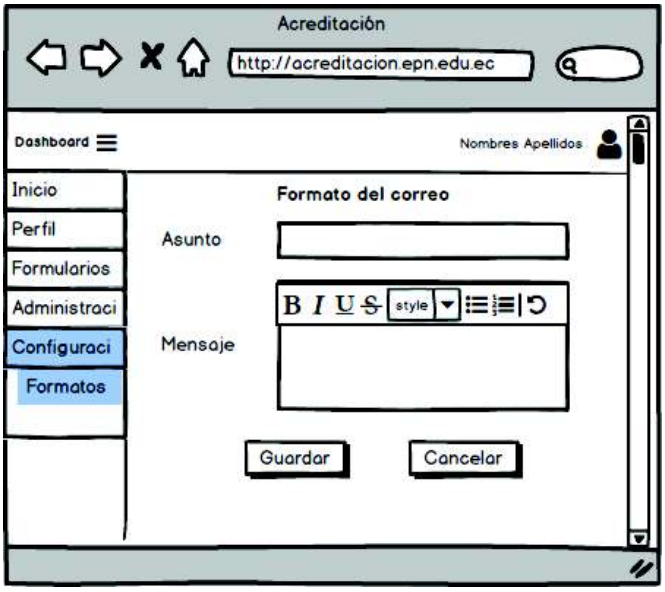


Figura 2.33. *Mockup* del formulario para dar formato a los correos electrónicos

En la Figura 2.34 se muestra el *mockup* del formulario para dar el formato al encabezado y el pie de la página de los informes que se generarán.

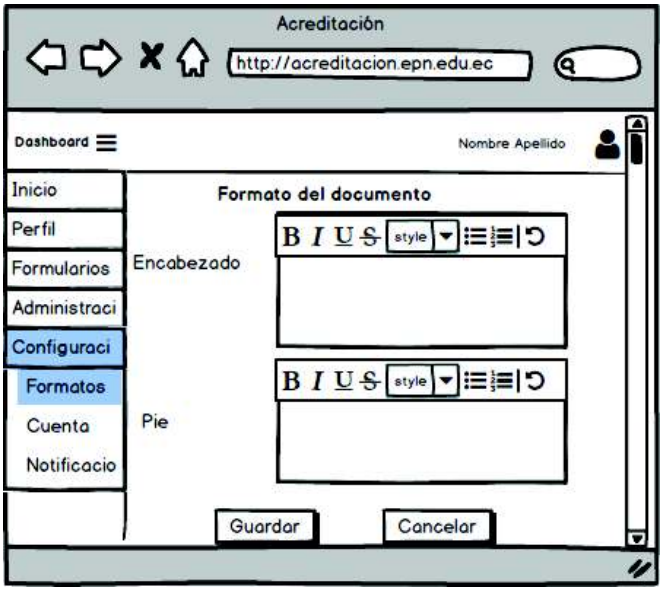


Figura 2.34. *Mockup* del formulario para dar el formato a los informes

En la Figura 2.35 se muestra el *mockup* del formulario que permite configurar la cuenta de correo electrónico, la cual se utilizará para enviar los correos electrónicos de notificaciones.

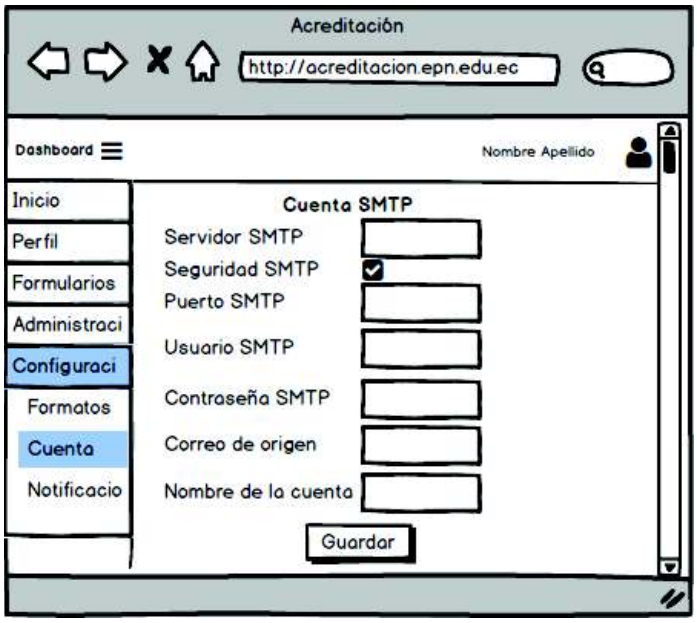


Figura 2.35. *Mockup* del formulario de la cuenta SMTP

En la Figura 2.36 se muestra el *mockup* del formulario para realizar la configuración de las notificaciones que se utilizan para enviar los mensajes de recordatorios a los profesores.

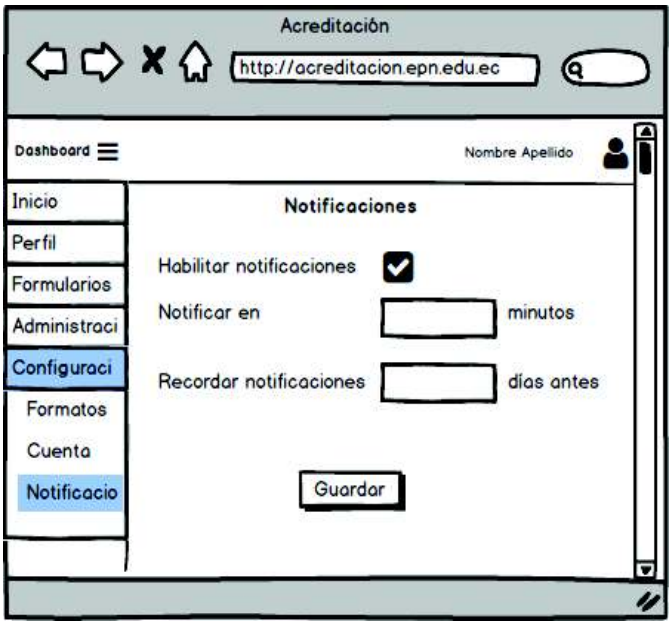


Figura 2.36. *Mockup* del formulario de la configuración de las notificaciones

CAPÍTULO 3

3. IMPLEMENTACIÓN Y PRUEBAS DE LA APLICACIÓN

3.1 IMPLEMENTACIÓN

La implementación de la aplicación web del presente Trabajo de Titulación se realizó utilizando las siguientes herramientas:

- Visual Studio 2016 con ASP.NET MVC para la implementación de la aplicación web.
- SQL Server 2017 para la gestión de la base de datos.
- SQL Server Management Studio (SSMS) para la implementación de la base de datos.
- Entity Framework v6.2.0 para generar las entidades del modelo de datos.
- ASP.NET Identity v2.2.1 para agregar un sistema de autenticación y autorización seguro.
- Html5 para organizar el contenido en cada una de las vistas.
- CSS3 para mejorar la apariencia del contenido en las vistas.
- La plantilla AdminLTE para conseguir una combinación apropiada de estética y funcionalidad durante la interacción entre el usuario y la aplicación web.
- La librería JQuery v3.3.1 para implementar y utilizar funcionalidades interactivas del lado del cliente.
- La librería de Boostrap v3.3.7 para conseguir que el contenido se ajuste a cualquier dispositivo y tamaño de pantalla; de tal forma que siempre se vea bien.
- El *plugin DataTables* v1.10.13 que es un complemento para la librería de JQuery para presentar tablas dinámicas en las que se puede ordenar el contenido de las columnas, realizar paginación, realizar búsquedas, entre otros.
- La librería iTextSharp v5.5.13 para crear archivos en formato PDF.

- La librería EPPlus v4.1.1 para leer y crear archivos en formato de Excel.
- El editor de textos TinyMCE v4.7.4 para establecer el encabezado y el pie de página de los informes en formato PDF.

3.1.1 NORMAS PARA LA IMPLEMENTACIÓN

Se utilizaron las siguientes normas para la implementación, las cuales son convenciones utilizadas para escribir el código de la aplicación web, de tal forma que el código sea fácilmente legible y entendible; por lo tanto, los nombres de los controladores, los métodos de acciones y las variables se escriben de acuerdo a la funcionalidad que realizan de una forma clara y descriptiva. Se utiliza la notación *PascalCase*[37] y *CamelCase*[38] para nombrar los principales elementos de ASP.NET MVC, como se describe en la Tabla 3.1.

Tabla 3.1. Notación para nombrar los elementos de la aplicación web

Elementos	Descripción
Controladores (clases)	Los nombres de los controladores se escribieron con la notación *PascalCase*, seguido de la palabra *Controller*, por ejemplo: `AreaPersonalController`.
Acciones (métodos)	El nombre de los métodos de acción se escribieron con la notación *PascalCase*, por ejemplo: `Index`.
Vistas	Los nombres de las vistas se escribieron con la notación *PascalCase* y tienen los mismos nombres de los métodos de acción.
Vistas Parciales	Los nombres de las vistas parciales se escribieron con un guion bajo inicial, seguido del nombre de la vista parcial con la notación *PascalCase*, por ejemplo: `_MenuFormularios`.
Entidades del modelo (objetos)	Los nombres de las entidades del modelo se escribieron en plural con la notación *PascalCase*, por ejemplo: `Formularios`.
Variables	Los nombres de las variables utilizarán la notación *CamelCase*, por ejemplo: `rutaArchivos`.

[37] *PascalCase*: Notación donde la primera letra del identificador de cada palabra comienza con mayúsculas, por ejemplo: `AreaPersonal`.
[38] *CamelCase*: Notación en la que la primera letra del identificador inicia con minúscula, mientras que las primeras letra de las siguientes palabras concatenadas están en mayúscula, por ejemplo: `msgCorreo`.

Los elementos de la base de datos también tienen su propia notación; para lo cual se utiliza *PascalCase* y *SnakeCase*[39], como se detalla en la Tabla 3.2.

Tabla 3.2. Notación para nombrar los elementos de la base de datos

Elementos	Descripción
Tablas	Los nombres de las tablas se escribieron en plural con la notación *PascalCase*, por ejemplo: `Formularios`.
Atributos (columnas)	Los nombres de los atributos se escribieron con la notación *SnakeCase* en singular, por ejemplo: `fecha_inicio`.

Cabe mencionar que las siguientes tablas se generaron automáticamente: `AspNetRoles`, `AspNetUserClaims`, `AspNetUserLogins`, `AspNetUserRoles` y `AspNetUsers`; motivo por el cual las tablas mencionadas y sus atributos utilizan la notación *PascalCase*.

3.1.2 BASE DE DATOS

La implementación de la base de datos se realizó con SQL Server Management Studio (SSMS); se creó la base de datos con el nombre `dbacreditacion`, luego se crearon las diferentes tablas de acuerdo a los diagramas presentados en el capítulo anterior.

En la Figura 3.1 se presenta el *script* utilizado para crear la tabla `Formularios` con sus atributos; se inicia creando la tabla `Formularios` con la sintaxis `CREATE TABLE` (línea 4), se crea el atributo `id` con la propiedad `IDENTY` para incrementar su valor automáticamente de uno en uno (línea 5) y se establece como clave primaria (línea 13); se crea el atributo `categoria_id` que tiene el tipo de dato entero (línea 6); se crean los atributos `nombre_largo` que es de tipo de dato `varchar` con una longitud de 200 caracteres (línea 7) y `nombre_corto` que es de tipo de dato `varchar` con una longitud de 100 caracteres (línea 8); se crea el atributo `fecha_modificacion` que es de tipo de dato fecha (línea 9); y, finalmente se crean los atributos `visible`, `portafolio`, e `informe` que son de tipo de dato booleano.

[39] *SnakeCase*: Notación donde cada una de las palabras se escriben separadas por un guion bajo (_), por ejemplo: `formulario_id`.

El *script* para crear la base de datos completa se encuentra en el Anexo C.

```
4  CREATE TABLE [Formularios](
5      [id] [int] IDENTITY(1,1) NOT NULL,
6      [categoria_id] [int] NULL,
7      [nombre_largo] [varchar](200) NULL,
8      [nombre_corto] [varchar](100) NULL,
9      [fecha_modificacion] [date] NULL,
10     [visible] [bit] NULL,
11     [portafolio] [bit] NULL,
12     [informe] [bit] NULL,
13 PRIMARY KEY ([id])
14 )
```

Figura 3.1. Código para crear la tabla `Formularios`

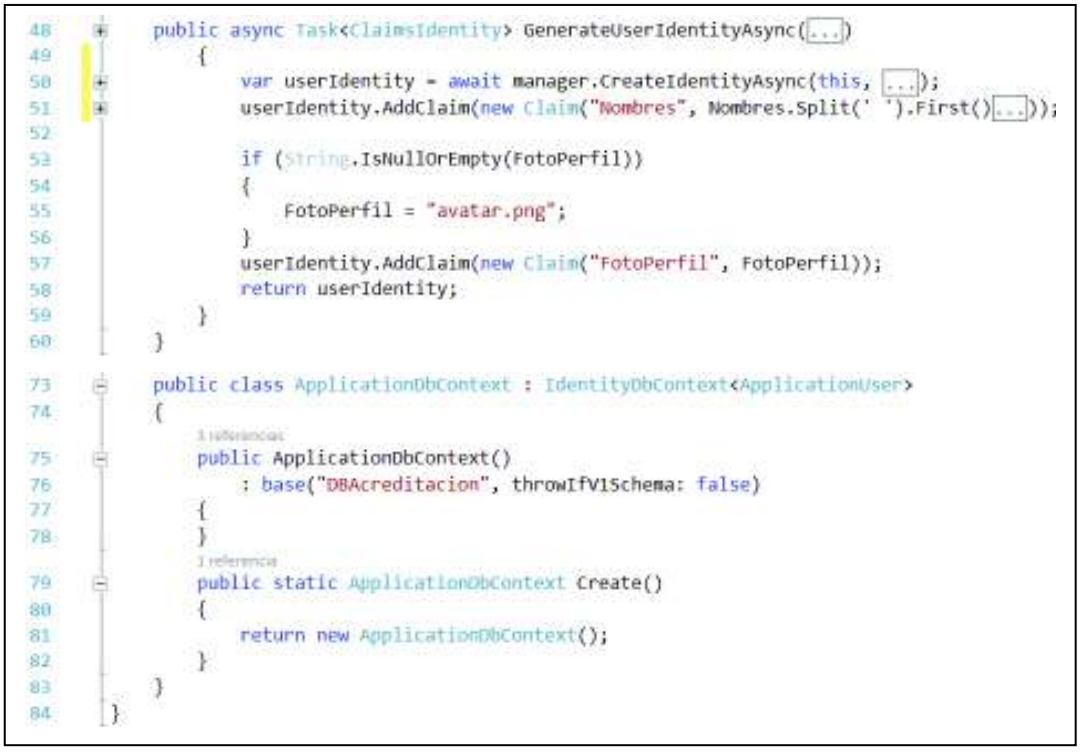
Una vez creada la base de datos, se crearon las tablas que almacenan la información de las cuentas de cada usuario, este proceso se realiza de forma automática; sin embargo, para llevar a cabo este proceso se tiene que ingresar al archivo `IdentityModels.cs` y escribir las sintaxis que se indican a continuación:

- Ubicar la clase `ApplicationUser` y dentro de esta clase agregar los atributos con sus respectivas anotaciones de datos, como se puede observar en la Figura 3.2. Se inicia escribiendo la anotación de datos `stringLength` para indicar que no se acepta más de 50 caracteres en el atributo `Nombre` (línea 17); `Display` para que se muestre una etiqueta con el atributo `Nombre` (línea 18); y finalmente, se escribe el tipo de dato y el nombre del atributo con las propiedades `GET` y `SET` (línea 19). Se procede de igual forma para los demás atributos: `Apellidos`, `CI`, entre otros.

```
15     public class ApplicationUser : IdentityUser
16     {
17         [StringLength(50)]
18         [Display(Name = "Nombres")]
           5 referencias
19         public string Nombres { get; set; }
20         [StringLength(50)]
21         [Display(Name = "Apellidos")]
           5 referencias
22         public string Apellidos { get; set; }
23         [StringLength(10)]
24         [Display(Name = "CI")]
           2 referencias
25         public string CI { get; set; }
26         ...
```

Figura 3.2. Nuevos campos en la tabla Usuarios

- Ubicar dentro de la clase `ApplicationUser` el método `GenerateUserIdentityAsync` y agregar las sintaxis para que se pueda visualizar el nombre del usuario y una imagen de perfil en cada una de las páginas de la aplicación web; las instrucciones que se agregan se pueden observar en la Figura 3.3. Se agrega un Claim que es un fragmento de información sobre el usuario con un valor `string`; para lo cual se especifica en el constructor del Claim una etiqueta `Nombres` con el valor que corresponde al primer nombre y al primer apellido del usuario (línea 51); luego se valida que el usuario tenga una foto de perfil, si no tiene se le asigna una por defecto (líneas 54-56); posteriormente se agrega otro Claim con la etiqueta `FotoPerfil` y en el valor se coloca la ruta de la foto de perfil (línea 57).
- Para crear las tablas que almacenan la información de los usuarios, es necesario escribir el nombre de la base de datos de la aplicación web dentro de la clase `ApplicationDbContext`, en la línea 76.



```
48      public async Task<ClaimsIdentity> GenerateUserIdentityAsync(...)
49          {
50              var userIdentity = await manager.CreateIdentityAsync(this, ...);
51              userIdentity.AddClaim(new Claim("Nombres", Nombres.Split(' ').First()...));
52
53              if (String.IsNullOrEmpty(FotoPerfil))
54              {
55                  FotoPerfil = "avatar.png";
56              }
57              userIdentity.AddClaim(new Claim("FotoPerfil", FotoPerfil));
58              return userIdentity;
59          }
60      }

73      public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
74      {
            3 referencias
75          public ApplicationDbContext()
76              : base("DBAcreditacion", throwIfV1Schema: false)
77          {
78          }
            1 referencia
79          public static ApplicationDbContext Create()
80          {
81              return new ApplicationDbContext();
82          }
83      }
84  }
```

Figura 3.3. Nuevos campos en la tabla de usuarios

- Finalmente, abrir la `Consola del Administrador` de paquetes y escribir: `Enable-Migrations` que permite activar las migraciones y crea

la carpeta `Migrations` en el proyecto; luego escribir `Add-Migration` seguido de un nombre, para generar un *script* de migración que contiene las instrucciones para crear o modificar las tablas en la base de datos; y, finalmente escribir `Update-Database` para actualizar el esquema de la base de datos.

3.1.3 TAREAS INICIALES DE LA APLICACIÓN WEB

Se creó la solución denominada `Acreditacion` y dentro de ella se crearon dos proyectos: el proyecto `Acreditacion` que contiene los controladores, las vistas, las librerías y los recursos necesarios de la aplicación web; mientras que en el otro proyecto denominado `Datos` se encuentran las entidades del modelo de datos. Se utiliza ASP.NET Identity con el tipo de autenticación de cuentas de usuarios individuales para autenticar a los usuarios mediante el registro de una cuenta de usuario, en la cual se ingresar únicamente el nombre de usuario y la contraseña.

3.1.3.1 Apariencia

Se integra la plantilla de un *dashboard* denominada AdminLTE en la aplicación web, la apariencia de la plantilla se puede observar en la Figura 3.4.

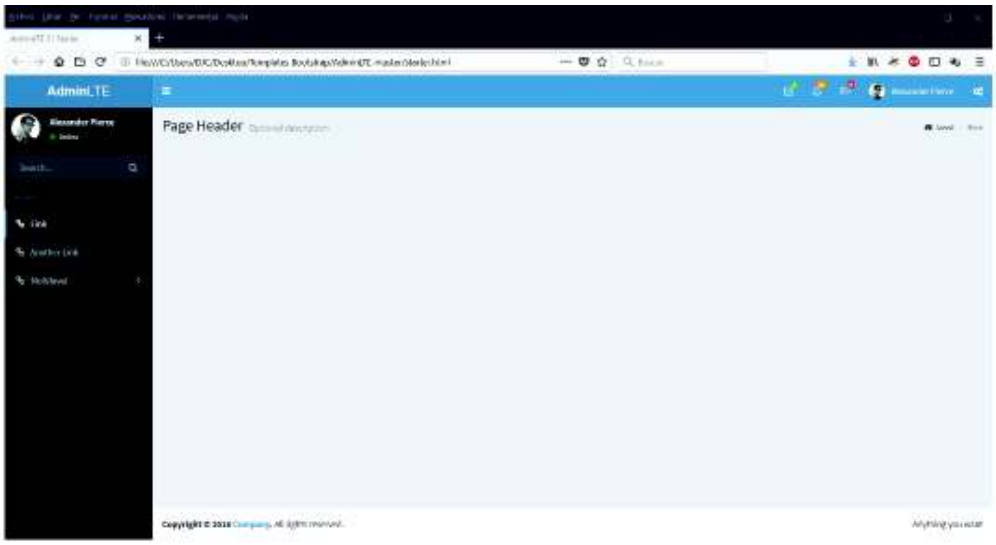


Figura 3.4. Plantilla de la aplicación web

Para agregar la plantilla en la aplicación web, se crea un *layout* dentro de la carpeta `Views/Shared`, el cual se denominó `_AreaPersonal.cshtml`; este archivo contiene la estructura HTML de la plantilla AdminLTE; también se incluyeron las

referencias a las hojas de estilos y las librerías utilizadas para agregar fuentes, íconos, responsividad, entre otras características.

En la Figura 3.5 se presentan en la parte a) las referencias de las hojas de estilos que se utiliza en la página, estas referencias se escriben dentro de la etiqueta `head` del documento HTML y se agregan con el método `Styles.Render()`, las cuales se encargan de buscar las referencias de su ubicación en el archivo `App_Start/BundleConfig.cs`; mientras que en la parte b) se presentan las referencias de las librerías utilizadas, las cuales se escriben al final de la etiqueta `body` del documento HTML.

La sentencia de la línea 35 permite agregar nuevos estilos en las vistas que requieran; la sentencia de la línea 365 permite incluir librerías JavaScript en las vistas que requieran de otras funcionalidades.

Finalmente, dentro de la etiqueta `body` del documento HTML se tiene que incluir la sentencia `@RenderBody()` la cual se utiliza para renderizar las vistas que utilizan esta plantilla.

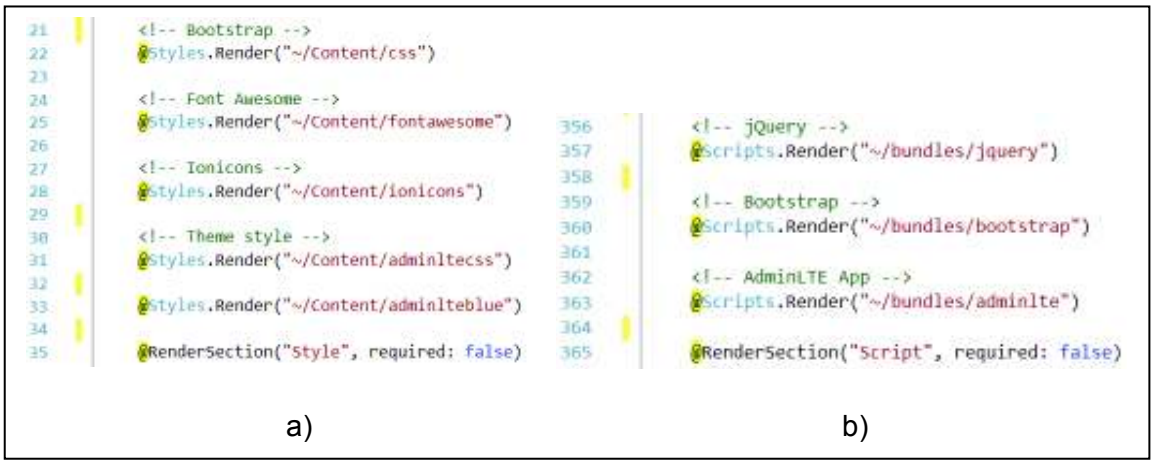


Figura 3.5. Integración de estilos y librerías en la plantilla

Las ubicaciones de los archivos de las hojas de estilos y de las librerías que se van a utilizar en la aplicación web se tienen que agregar en el archivo `BundleConfig.cs`, el cual se encuentra dentro de la carpeta `App_Start`; esto con la finalidad de optimizar la carga del código JavaScript y las hojas de estilos cuando el cliente realice las solicitudes de las vistas.

En la Figura 3.6 se presenta un ejemplo de código para agregar Bootstrap al archivo `BundleConfig.cs`, en la línea 23 se crea un nuevo paquete de JavaScript llamado `~/bundles/boostrap`, en la línea 24 se especifica la ubicación del archivo JavaScript; de forma similar se procede para agrega la hoja de estilos de Bootstrap mediante las sentencias de las líneas 26 y 27.

```
23              bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
24                        "~/Content/js/bootstrap.min.js"));
25
26              bundles.Add(new StyleBundle("~/Content/css").Include(
27                        "~/Content/css/bootstrap.min.css"));
```

Figura 3.6. Ubicación de estilos y *scripts* en el *bundle*

Se crean los *layouts*: `_LayoutInicio.cshtml` utilizada para presentar la página pública de la aplicación web; y, `_Login.cshtml` utilizado para presentar la página del *Login*.

3.1.3.2 Modelo

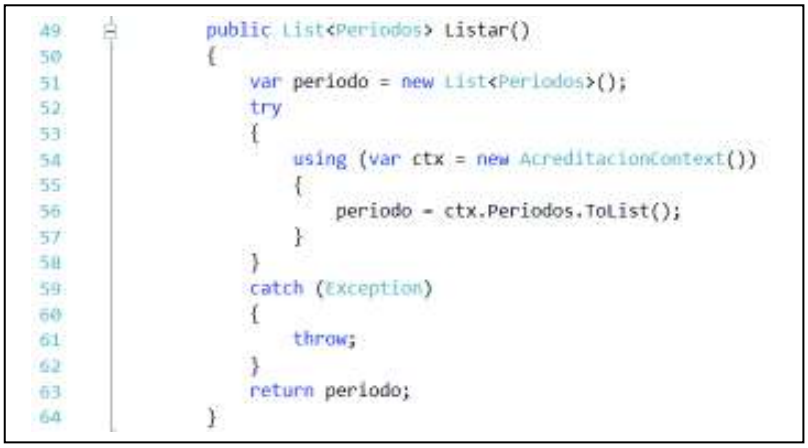
En el proyecto `Datos` se utilizó Entity Framework con *Code First* para crear las entidades del modelo a partir de la base de datos existente; también se creó la clase de contexto que se denomina `AcreditacionContext`.

Se agregaron métodos a cada una de las entidades del modelo con las funciones CRUD[40] para evitar repetir el código durante la implementación de la aplicación web, como ejemplo se presentan los métodos que se agregaron en la entidad `Periodos`.

En la Figura 3.7 se presenta el método `Listar` que devuelve una lista de objetos con la información de todos los periodos académicos ingresados en la base de datos. En la línea 51 se declara una variable `periodo`, la cual obtendrá la lista de los objetos `Periodos`; desde la línea 52 hasta la línea 62 se utiliza un control de excepciones mediante `try` y `catch` para manejar los posibles errores; desde la línea 54 a la 57 se utiliza la instrucción `using` para liberar de la memoria los objetos que ya no se utilizan, entre los paréntesis se instancia la clase de contexto que se

[40] CRUD (*Create*, *Read*, *Update and Delete*): Se utiliza para referirse a las funciones básicas en las bases de datos, para crear, leer, actualizar y borrar registros.

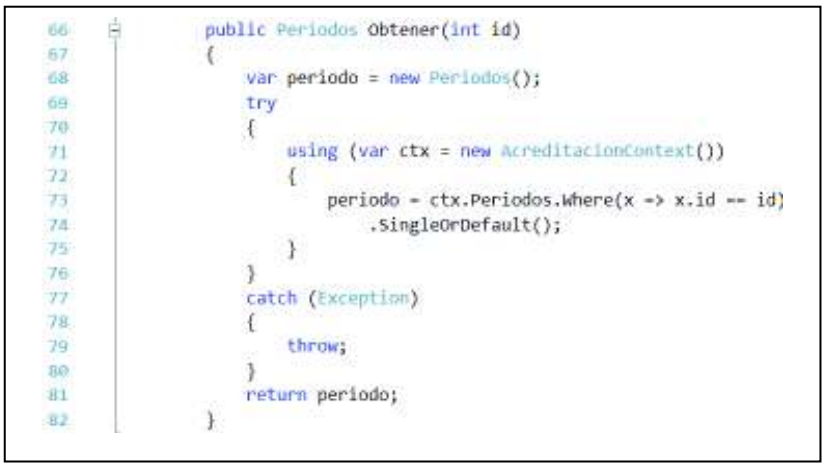
utilizará para hacer la consulta a la base de datos (línea 54), en la línea 56 se realiza la consulta a la base de datos, y finalmente se retorna la lista de periodos académicos (línea 63).



```
49      public List<Periodos> Listar()
50      {
51          var periodo = new List<Periodos>();
52          try
53          {
54              using (var ctx = new AcreditacionContext())
55              {
56                  periodo = ctx.Periodos.ToList();
57              }
58          }
59          catch (Exception)
60          {
61              throw;
62          }
63          return periodo;
64      }
```

Figura 3.7. Método para listar las entidades del modelo

En la Figura 3.8 se presenta el método `Obtener` que recibe como parámetro de entrada un `id`, que es el identificador único de un registro y como resultado se devuelve un objeto `periodo`. Las instrucciones son semejantes al método `Listar` y la diferencia se encuentra en las líneas 73 y 74, las cuales permiten obtener la información del periodo específico mediante la utilización de LINQ con operaciones Lambda para hacer la consulta a la base de datos; también se utiliza el método `SingleOrDefault()` para obtener un único elemento de una secuencia o un valor predeterminado si la secuencia está vacía; y, finalmente se retorna la información del periodo consultado (línea 81).



```
66      public Periodos Obtener(int id)
67      {
68          var periodo = new Periodos();
69          try
70          {
71              using (var ctx = new AcreditacionContext())
72              {
73                  periodo = ctx.Periodos.Where(x => x.id == id)
74                      .SingleOrDefault();
75              }
76          }
77          catch (Exception)
78          {
79              throw;
80          }
81          return periodo;
82      }
```

Figura 3.8. Método obtener una entidad del modelo

En la Figura 3.9 se presenta el método `Guardar` que se implementó en cada una de las entidades del modelo, se utiliza una estructura de código semejante a los métodos anteriores en lo referente al control de excepciones. Este método permitirá actualizar un registro o almacenar un nuevo registro para lo cual se utiliza un operador condicional que determina que: si el identificador `id` es mayor que cero (línea 90) se modifican las propiedades del objeto (línea 91), caso contrario se agrega el objeto al contexto de objetos (líneas 93-94) y finalmente se utiliza el método `SaveChanges()` (línea 94) para modificar o almacenar la información en la base de datos.

```
84      public void Guardar()
85      {
86          try
87          {
88              using (var ctx = new AcreditacionContext())
89              {
90                  if (this.id > 0)
91                      ctx.Entry(this).State = EntityState.Modified;
92                  else
93                      ctx.Entry(this).State = EntityState.Added;
94                  ctx.SaveChanges();
95              }
96          }
97          catch (Exception)
98          {
99              throw;
100         }
```

Figura 3.9. Método para guardar entidades del modelo

En la Figura 3.10 se presenta el método `Eliminar` el cual se utiliza para eliminar un registro de la base de datos; se utiliza la estructura de código para el manejo de excepciones semejante a los métodos anteriores. En la línea 111 se indica que se va a eliminar la entidad en uso, en la línea 112 se utiliza el método `SaveChanges()` para que se ejecute el procedimiento para eliminar el registro de la base de datos.

```
105     public void Eliminar()
106     {
107         try
108         {
109             using (var ctx = new AcreditacionContext())
110             {
111                 ctx.Entry(this).State = EntityState.Deleted;
112                 ctx.SaveChanges();
113             }
114         }
115         catch (Exception)
116         {
117             throw;
118         }
119     }
```

Figura 3.10. Método para eliminar las entidades del modelo

3.1.3.3 Vista

Las vistas de la aplicación web se encuentran en la carpeta `/Views/NombreDelControlador/`, el código de cada una de las vistas se organiza como se muestra en la Figura 3.11.

Se inicia indicando la entidad o las entidades del modelo que se van a utilizar en la vista (línea 1); se accede a las propiedades del modelo utilizando `@Model.NombrePropiedad`. Mediante un bloque de código de Razor (líneas 2-6) se especifica el título de la página web (línea 3), la plantilla utilizada (línea 4) y el menú expansible al cual pertenece la vista (línea 5). Se tiene una sección denominada `Title` que es utilizada para colocar un título y subtítulo en la vista, esto se visualizará en la parte superior de la vista. Se tiene un bloque para mostrar mensajes a los usuarios (línea 12). Se tiene un bloque de código HTML y de código Razor para presentar la información solicitada al usuario a través de tablas o para solicitar al usuario información mediante formularios (línea 14).

Se tienen secciones para agregar estilos o para agregar las referencias de los archivos que contienen las hojas de estilos (líneas 17-19). Finalmente, se tiene una sección para agregar el código de JavaScript o las referencias a los archivos de las correspondientes librerías utilizadas en la vista (líneas 22-24).

```
1  @model IEnumerable<Datos.Periodos>
2  @{
3      ViewBag.Title = "Lista de Periodos";
4      Layout = "~/Views/Shared/_AreaPersonal.cshtml";
5      ViewBag.Admin = "active";
6  }
7
8  @section Title {
9      //Bloque de título (...)
10 }
11
12 //Bloque de mensaje (...)
13
14 //Bloque de HTML+Código de razor (...)
15
16
17 @section Style {
18     //Bloque de estilos (...)
19 }
20
21
22 @section Script {
23     //Bloque de JavaScrip (...)
24 }
```

Figura 3.11. Estructura de la vista

3.1.3.4 Controlador

Los controladores del proyecto se encuentran en la carpeta `Controllers` y cada controlador está organizado como se muestra en la Figura 3.12: se inicia declarando el ámbito en el que se encuentra la clase (línea 8); posteriormente se tiene la clase del controlador que hereda funcionalidades de la clase `Controller`, la cual le proporcionar el control general de MVC (línea 10); luego se instancian algunas entidades del modelo que se utilizarán en los métodos del controlador (líneas 12 a 19); y, finalmente se definen los métodos de acción que contienen la lógica para realizar las tareas específicas (líneas 23 a 26).

Cabe señalar que algunos métodos de acción son utilizados para retornar vistas, para realiza una redirección a otro método de acción o para realizar funciones específicas.

```
8   namespace Acreditacion.Controllers
9   {
        0 referencias
10      public class DefaultController : Controller
11      {
12          private AcreditacionContext db = new AcreditacionContext();
13          private Periodos periodo = new Periodos();
14          private AspNetUsers usuario = new AspNetUsers();
15          private Categorias categoria = new Categorias();
16          private Formularios formulario = new Formularios();
17          private FormatoCorreos fmtCorreo = new FormatoCorreos();
18          private FormatoDocumentos fmtDocumento = new FormatoDocumentos();
19          private Logs evento = new Logs();
20
21
22          // GET: Default
            0 referencias
23          public ActionResult Index()
24          {
25              return View();
26          }
27      }
28  }
```

Figura 3.12. Estructura del controlador

3.1.4 MÓDULOS DE LA APLICACIÓN

En esta sección se presentarán las tareas del tablero Kanban que están organizadas por módulos y están descritas en el capítulo anterior.

3.1.4.1 Módulo de administración y autenticación de usuarios

3.1.4.1.1 Registro de usuarios

Los métodos para el registro de usuarios se encuentran codificados en el controlador `AreaPersonalController` y la vista utilizada se encuentra en la carpeta `/Views/AreaPersonal`.

En la Figura 3.13 se presenta el método denominado `Register` que retorna una vista, la cual permitirá registrar un nuevo usuario o modificar la información de un usuario existente.

En la línea 384 se especifica una etiqueta indicando que pueden acceder a este método solo los administradores y súper administradores; este método recibe el parámetro `usuarioId` (línea 385), que se utiliza para conocer si se trata de un usuario existente cuando el valor es diferente de nulo, pero se trata de un nuevo usuario cuando el valor es nulo (línea 388); se instancia un objeto de tipo `user` (línea 387) que es una entidad para asignar la información básica de un usuario como su `Id`, `Nombres`, `Apellidos`, `Ci`, `Email` y `Password`. Si el usuario existe se recupera la información mediante el método `Obtener` (línea 390) y se asignan algunos valores del objeto `usuario` al objeto `user` (líneas 391-395). Finalmente se retorna la información a su vista `Register` (línea 398).

```
384         [Authorize(Roles = "SuperAdmin,Admin")]
            0 referencias
385         public ActionResult Register(string usuarioId = null)
386         {
387             RegisterViewModel user = new RegisterViewModel();
388             if (usuarioId != null)
389             {
390                 usuario = usuario.Obtener(usuarioId);
391                 user.Id = usuario.Id;
392                 user.Nombres = usuario.Nombres;
393                 user.Apellidos = usuario.Apellidos;
394                 user.Ci = usuario.Ci;
395                 user.Email = usuario.Email;
396                 ...
397             }
398             return View(user);
399         }
```

Figura 3.13. Método para presentar la vista que permite registrar usuarios

En la Figura 3.14 se presenta la vista del registro o modificación de una cuenta de un usuario, se visualiza en la parte a) el formulario que se debe llenar con la

información del nuevo usuario; en la parte b) se visualiza la información de un usuario registrado que se puede modificar.

Figura 3.14. Vista para registrar/modificar la información de un usuario

En la Figura 3.15 se observa que se almacenó correctamente la información de un usuario en la base de datos; además, se puede visualizar que se almacenó el resultado de una función *hash*, la cual fue aplicada a la contraseña de la cuenta del usuario por cuestiones de seguridad.

Id	Nombres	Apellidos	CI	Email	PasswordHash	Activo
3e978f1e-eb1c-47cd-9c21-e3e02fd8921a	Juan Carlos	Flores Moreno	2222222222	juan.flores@epn.edu.ec	AM4FR2HX+fDZv6gsVGZU0o6uZn5uJUwp6g/trPZxDi0/bMK1...	0

Figura 3.15. Registro de un usuario en la base de datos

3.1.4.1.2 Cambiar de rol a los usuarios

En la Figura 3.16 se presenta el método `RolUsuario` del controlador `AdministracionController`, utilizado para cambiar el rol a los usuarios.

Se reciben los parámetros `usuarioId` que es el identificador del usuario y el parámetro `rolName` que es el nombre del nuevo rol que se asignará al usuario (línea 146); se valida que los parámetros no sean nulos ni estén en blanco (línea 148); se obtiene el rol anterior del usuario (línea 150) y se elimina el rol anterior (línea 151); finalmente se establece el nuevo rol (línea 153). Se asignan un mensaje indicando que se realizó el cambio del rol del usuario (líneas 155-156) y se retorna a la vista que muestra la lista de usuarios registrados (línea 158).

```
146         public ActionResult RolUsuario(string usuarioId, string rolName)
147         {
148             if (!string.IsNullOrEmpty(usuarioId) && !string.IsNullOrEmpty(rolName))
149             {
150                 var roles = UserManager.GetRoles(usuarioId);
151                 UserManager.RemoveFromRoles(usuarioId, roles.ToArray());
152 
153                 var resultado = UserManager.AddToRole(usuarioId, rolName);
154 
155                 TempData["mensaje"] = "Se modificó el rol del usuario!";
156                 TempData["mensajeTipo"] = "alert-success";
157             }
158             return Redirect("~/Administracion/VerUsuarios");
159         }
```

Figura 3.16. Método para cambiar el rol de un usuario

3.1.4.1.3 Activar y desactivar usuarios

En la Figura 3.17 se presenta el método `ActivarUsuario` del controlador `AdministracionController`, el cual se utiliza para activar o desactivar los usuarios registrados.

Se reciben los parámetros `usuarioId` que es el identificador del usuario y el parámetro `activo` que indica el nuevo estado del usuario que será `true` o `false` (línea 128), se valida que el parámetro `usuarioId` no sea nulo ni esté en blanco (línea 130), se obtiene la información del usuario específico con el método `Obtener` (línea 132), se asigna el nuevo estado de activo o inactivo según corresponda (línea 133) y se guarda los cambios en la base de datos mediante el método `Guardar` (línea 134). Se asigna un mensaje indicando que se realizó el cambio de estado correctamente (líneas 136-137) y se retorna a la vista que muestra la lista de usuarios (línea 139).

```
128         public ActionResult ActivarUsuario(string usuarioId, bool activo)
129         {
130             if (!string.IsNullOrEmpty(usuarioId))
131             {
132                 usuario = usuario.Obtener(usuarioId);
133                 usuario.Activo = activo;
134                 usuario.Guardar();
135 
136                 TempData["mensaje"] = "Se modificó la cuenta del usuario!"
137                 TempData["mensajeTipo"] = "alert-success";
138             }
139             return Redirect("~/Administracion/VerUsuarios");
140         }
```

Figura 3.17. Método para cambiar el estado de un usuario

3.1.4.1.4 Vista para la edición y la visualización de la información del usuario

En la Figura 3.18 se presenta la vista `EditarUsuario` que se utiliza para modificar la información personal del usuario.

La vista cuenta con un formulario que se envía mediante el método `POST` de HTTP (línea 13); se utiliza el método `AntiForgeryToken()` para validar la identidad del formulario mediante un campo oculto que se verifica durante el envío del formulario (línea 15); se agregan los campos ocultos en el documento HTML que son el `Id` de usuario y la foto de perfil (líneas 16-17); se agrega la clase `ValidationSummary()` que muestra un resumen de todos los errores de validación del formulario (línea 20); se utilizan *helpers* para presentar los campos con la información del usuario que se va a modificar (líneas 22-96); y, se agrega un botón para enviar el formulario (línea 99) y un botón para cancelar que retorna a la vista de la información del usuario (línea 100).

Finalmente se agrega las referencias a las librerías de JQuery para validar la información del formulario (línea 107).

```
13  @using (Html.BeginForm("EditarUsuario", "AreaPersonal", FormMethod.Post,...
14  {
15      @Html.AntiForgeryToken()
16      @Html.HiddenFor(x => x.Id)
17      @Html.HiddenFor(x => x.FotoPerfil)
18
19      <div class="form-horizontal">
20          @Html.ValidationSummary(true, "", new { @class = "text-danger" })
21
22          <div class="form-group">
23              <div class="control-label col-md-2 text-bold">Nombres</div>
24              <div class="col-md-6">
25                  @Html.EditorFor(model => model.Nombres, new { htmlAttributes = new { ... } })
26                  @Html.ValidationMessageFor(model => model.Nombres, "", new { ... })
27              </div>
28          </div>
29          ...
97          <div class="form-group">
98              <div class="col-md-offset-2 col-md-10">
99                  <input type="submit" value="Guardar" class="btn btn-default" />
100                 <a class="btn btn-default" href="~/AreaPersonal/DatosUsuario">Cancelar</a>
101             </div>
102         </div>
103     </div>
104 }
105
106 @section Script {
107     @Scripts.Render("~/bundles/jqueryval")
108 }
```

Figura 3.18. Vista para presentar la información de un usuario

En la Figura 3.19 se presenta la vista que muestra la información de un usuario; se realizaron cambios como la foto de perfil y la dirección de correo electrónico, lo cual se puede visualizar en la Figura 3.20.

Figura 3.19. Prueba de la vista de la información personal del usuario

Figura 3.20. Prueba de la edición de la información personal del usuario

3.1.4.1.5 Método para eliminar usuarios

En la Figura 3.21 se presenta el método `EliminarUsuario` que permite eliminar a los usuarios registrados.

Este método se codifica en el controlador `AreaPersonalController` y recibe como parámetro el `Id` del usuario (línea 714), se valida que el `Id` no sea nulo ni se encuentre vacío (línea 716), se obtienen los eventos del usuario (línea 718) y se eliminan los eventos en el caso de existir (líneas 719-723). Se asigna la cadena de caracteres `usuarioId` del usuario al atributo `Id` del objeto `usuario` (línea 724) y

se utiliza el método `Eliminar` para borrar al usuario de la base de datos (línea 725).

Posteriormente se elimina la imagen de perfil si existe (líneas 727-729), se asignan valores para presentar un mensaje (líneas 731-732) y finalmente se retorna a la vista que muestra la lista de usuarios (línea 734).

```
713         [Authorize(Roles = "SuperAdmin,Admin")]
            0 referencias
714         public ActionResult EliminarUsuario(string usuarioId)
715         {
716             if (!string.IsNullOrEmpty(usuarioId))
717             {
718                 var eventos = db.Logs.Where(x=>x.usuario_id==usuarioId).ToList();
719                 if(eventos!=null)
720                 {
721                     db.Logs.RemoveRange(eventos);
722                     db.SaveChanges();
723                 }
724                 usuario.Id = usuarioId;
725                 usuario.Eliminar();
726 
727                 string pathImage = pathRaiz + pathFotosPerfil + "img_" + usuarioId;
728                 if (System.IO.File.Exists(Server.MapPath(pathImage)))
729                     System.IO.File.Delete(Server.MapPath(pathImage));
730 
731                 TempData["mensaje"] = "Se eliminó el usuario!";
732                 TempData["mensajeTipo"] = "alert-success";
733             }
734             return Redirect("~/Administracion/VerUsuarios");
735         }
```

Figura 3.21. Método para eliminar usuarios

3.1.4.1.6 Visualización de la lista de usuarios

Los administradores y súper administradores podrán visualizar la lista de usuarios registrados, para lo cual se utiliza el método `VerUsuarios` y una vista con el mismo nombre; este método se encuentra en el controlador `AdministracionController`.

En la Figura 3.22 se presenta el código de la vista que muestra una tabla HTML con la información de los usuarios registrados (líneas 34-92); para lo cual se especifica los nombres de las columnas de la tabla HTML (líneas 37 a 50), luego se recorre la lista de usuarios obtenida de la base de datos (línea 53) y se agrega la información del usuario en cada columna de la tabla HTML (líneas 55 a 89).
Se agrega un campo en cada fila de la tabla HTML con un formulario para cambiar el rol de cada usuario (líneas 61 a 69), se muestra una lista desplegable de roles

(líneas 64 a 67) y se encuentra seleccionado el respectivo rol del usuario en la lista (línea 66). Se agrega un campo en cada fila de la tabla HTML con un formulario para activar/desactivar a un usuario (líneas 71 a 72) y se muestra en un `CheckBox` el estado del usuario (línea 74).

```
34      <div class="table-responsive">
35          <table class="table table-hover" id="usuarios">
36              <thead>
37                  <tr>
38                      <th>Nombres</th>
39                      <th>Apellidos</th>
40                      <th>CI</th>
41                      ...
50                  </tr>
51              </thead>
52              <tbody>
53                  @foreach (var item in Model)
54                  {
55                      <tr> ...
60                              @using (Html.BeginForm("RolUsuario", "Administracion", FormMethod.Post...))
61                              { ...
63                                  <select name="rolName" onchange = "this.form.submit();" >
64                                      @foreach (SelectListItem option in ViewBag.rolId)
65                                      {
66                                          <option value="@option.Text" ...@option.Text</option>
67                                      }
68                                  </select>
69                              } ...
71                              @using (Html.BeginForm("ActivarUsuario", "Administracion", FormMethod.Post...))
72                              { ...
74                                  @Html.CheckBox("activo", item.Activo, new { onchange = "this.form.submit();" }
75                              }
76                          </td>
77                          <td class="text-center">@item.Telefono</td>
78                          ...
89                      </tr>
90                  }
91              </tbody>
92          </table>
```

Figura 3.22. Código de la vista para mostrar la lista de usuarios

En la Figura 3.23 se presenta la vista que muestra una lista de usuarios registrados, cada uno de los registros de los usuarios tienen un menú desplegable para permitir cambiar el rol, un `CheckBox` para cambiar el estado y tiene botones para modificar o eliminar la cuenta del usuario. En la parte inferior se tienen botones para agregar nuevos usuarios y para generar informes en formato de Excel y PDF.

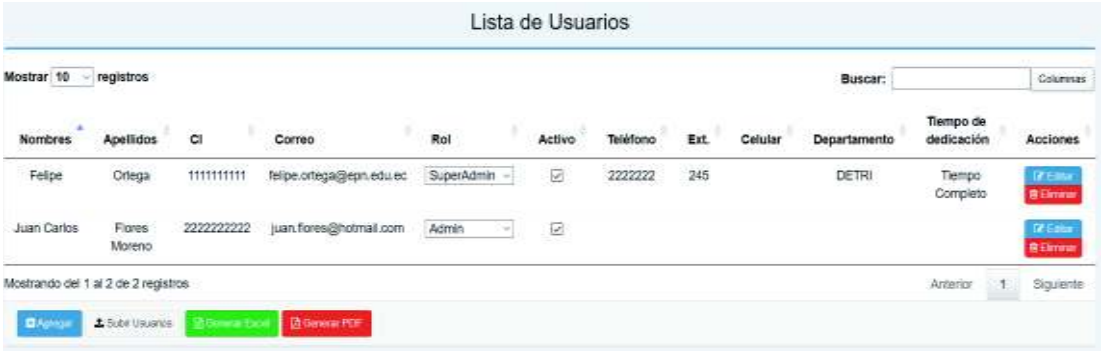


Figura 3.23. Vista que muestra una lista de los usuarios registrados

3.1.4.1.7 Métodos para generar archivos con la lista de los usuarios

En la Figura 3.24 se presenta el método `ListaUsuariosExcel` que se utiliza para generar un archivo en formato Excel con la lista de todos los usuarios registrados.

Se obtiene la lista de usuarios registrados (línea 472), se agrega los registros de cada usuario en una fila de una tabla temporal (líneas 487 - 494), se obtiene de la base de datos el formato de encabezado y de pie de página (línea 497), se agrega una hoja de Excel (línea 499); se agrega cada línea del encabezado con su formato a la hoja de Excel (líneas 508-531). Se obtienen el número de líneas del encabezado y se agrega 4 espacios para provocar una separación entre la lista de usuarios y el encabezado de la página (línea 534), se escribe la posición del inicio de la lista de usuarios (línea 534), se agrega la tabla de usuarios a la hoja de Excel (línea 535), se agrega el formato del pie de la página de igual forma que el encabezado (líneas 541-573). Se guarda temporalmente el archivo en la memoria del servidor (línea 574) y finalmente se utiliza el objeto `Response` para enviar el archivo al cliente (líneas 576 a 581).

```
470         public void ListaUsuariosExcel()
471         {
472             var usuarios = usuario.Listar();
473             ...
484             foreach (var item in usuarios)
485             {
486                 DataRow row = table.NewRow();
487                 row["Nro."] = numItem++;
488                 row["Nombres"] = item.Nombres;
489                 row["Apellidos"] = item.Apellidos;
490                 ...
494                 table.Rows.Add(row);
495             }
496             int posFila = 0;
497             var formato = db.FormatoDocumentos.....SingleOrDefault();
498             ExcelPackage excel = new ExcelPackage();
499             var ws = excel.Workbook.Worksheets.Add("Hoja1");
500             if (String.IsNullOrEmpty(formato.encabezado))
501             {...
507                 for (int i = 0; i < encLinea.Length; i++)
508                 {
509                     ws.Cells[1 + i, 1].Value = encLinea[i];
510                     ws.Cells[1 + i, 1, 1 + i, 7].Merge = true;
511                     ws.Cells[1 + i, 1, 1 + i, 7].Style.Font.Bold = true;
512                     ws.Cells[1 + i, 1, 1 + i, 7].Style.Font.Size = Convert.ToInt32(encTam[i]);
513                     ws.Cells[1 + i, 1, 1 + i, 7].Style.Font.Name = encTipo[i];
514                     ...
531                 }
532                 posFila = encLinea.Length + 4;
533             }
534             string celda = "A" + posFila;
535             ws.Cells[celda].LoadFromDataTable(table, true, OfficeOpenXml.Table.TableStyles.Light10)
536             ...
540             if (String.IsNullOrEmpty(formato.pie))
541             {
542                 ...
573             }
574             using (var memoryStream = new MemoryStream())
575             {
576                 Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml
577                 Response.AddHeader("content-disposition", "attachment; filename=" + "ListaProfesore
578                 ...
581                 Response.End();
```

Figura 3.24. Método para generar un archivo en formato de Excel

En la Figura 3.25 se presenta el método `ListaUsuariosPdf` que se utiliza para generar una lista con todos los usuarios registrados en un archivo con formato PDF.

Se obtiene la lista de usuarios registrados de la base de datos (línea 591), se crea una tabla temporal con la lista de los usuarios (líneas 607 a 614), se carga la tabla en un `grid` (línea 619), se agregan estilos al contenido del `grid` (líneas 624-625), se establecen las características del formato PDF para que tenga una orientación horizontal y deje una separación en cada borde de 50 pixeles (línea 648). Se obtiene el formato del encabezado y del pie de página de la base de datos (línea 650), se agrega el encabezado, el `grid` con la información de los usuarios (líneas 657-660) y el pie de la página. Finalmente se utiliza el objeto `Response` para enviar el archivo en formato PDF al cliente (líneas 670-673).

```
589         public void ListaUsuariosPdf(string opcion, int frmId = 0, int periodoId = 0)
590         {
591             var usuarios = usuario.Listar();
592
593             DataTable table = new DataTable();
594             ...
604             foreach (var item in usuarios)
605             {
606                 DataRow row = table.NewRow();
607                 row["Nro."] = numItem++;
608                 row["Nombres"] = item.Nombres;...
614                 table.Rows.Add(row);
615             }
616
617             GridView grid = new GridView();
618             ...
619             grid.DataSource = table;
620             ...
624             grid.HeaderRow.Style.Add("color", "#FFFFFF");
625             ...
648             Document pdfDoc = new Document(PageSize.A4.Rotate(), 50, 50, 50, 50);
649             ...
650             var formato = fmtDocumento.Obtener(0, "Lista de usuarios");
651             pdfDoc.Open();
652             ...
657             using (var srHtml = new StringReader(sw.ToString()))
658             {
659                 XMLWorkerHelper.GetInstance().ParseXHtml(writer, pdfDoc, srHtml);
660             }
661             ...
670             Response.ContentType = "application/pdf";
671             Response.AddHeader("content-disposition", "attachment;filename=reporte.pdf");
672             Response.Write(pdfDoc);
673             Response.End();
674         }
```

Figura 3.25. Método para generar un archivo en formato PDF

3.1.4.1.8 Autenticación de usuarios en la aplicación web

En la Figura 3.26 se presenta el método `Login`, que se utiliza para autenticar a los usuarios; se destaca porque es un método asincrónico, motivo por el cual se escribe `async Task<>` (línea 97) y dentro del método se especifica un `await` en la actividad que tomará tiempo en realizarse (línea 103). Además, este método recibe como parámetros las credenciales de los usuarios y la ruta de una vista a la que quiere acceder (línea 97). Si el modelo no es válido se retorna la vista del *Login* (líneas 99-102). Para autenticar las credenciales del usuario se utilizando el método `PasswordSingInAsync` (línea 103).

Finalmente, de acuerdo al resultado obtenido en la autenticación se retornará una de las siguientes opciones al usuario:

- Si la autenticación fue correcta, se direccionará a la vista que solicitó el acceso (línea 140).
- Se indicará que la cuenta se encuentra bloqueada (línea 142).
- Se indicará que se requiere la activación de la cuenta (línea 144).
- Se indicará que el intento de inicio de sesión no es válido (línea 147).

```
97      public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
98      {
99          if (!ModelState.IsValid)
100         {
101             return View(model);
102         }
103         var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password...);
104         string id = UserManager.FindByEmail(model.Email).Id;
105         ...
137         switch (result)
138         {
139             case SignInStatus.Success:
140                 return RedirectToLocal(returnUrl);
141             case SignInStatus.LockedOut:
142                 return View("Lockout");
143             case SignInStatus.RequiresVerification:
144                 return RedirectToAction("SendCode", new { ReturnUrl = returnUrl...});
145             case SignInStatus.Failure:
146             default:
147                 ModelState.AddModelError("", "Intento de inicio de sesión no válido.");
148                 return View(model);
149         }
150     }
```

Figura 3.26. Método para autenticar usuarios

En la Figura 3.27 se presenta una prueba de la autenticación de un usuario que tiene un rol de súper administrador; en la parte a) se presenta la vista del *Login* y en la parte b) se presenta una parte de la vista que muestra la foto con el nombre del usuario autenticado y un menú desplegable con los enlaces a las vistas a las que tiene acceso con el rol de súper administrador.

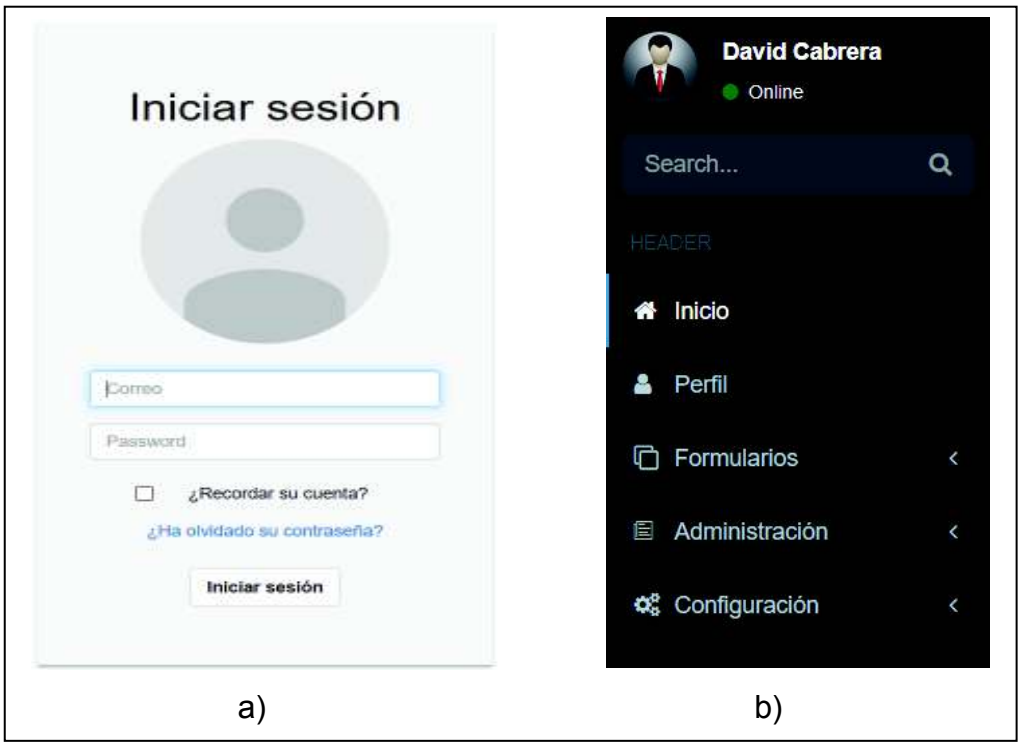


Figura 3.27. Autenticación de un usuario

3.1.4.2 Módulo para la recepción de la documentación de los profesores

Los métodos de este módulo se implementaron en el controlador `FormularioController`.

3.1.4.2.1 Vista para mostrar la lista de categorías

Para visualizar las categorías se codifica un método y una vista para mostrar todas las categorías que están registradas en la base de datos.

En la Figura 3.28 se presenta la vista con la lista de categorías que se encuentran almacenadas en la base de datos. Las categorías están organizadas en una tabla HTML en la que se pueden ver sus características y se tiene botones para agregar una nueva categoría, modificar o eliminar una categoría existente.

Figura 3.28. Vista que muestra la lista de categorías

3.1.4.2.2 Vista para crear o modificar categorías

Para crear o modificar categorías se codifica un método y una vista denominados `Categoria` que tiene un formulario en la vista para especificar la nueva categoría o modificar la información de una categoría existente; se tiene también el método denominado `GuardarCategoria` que se encarga de validar y almacenar la información de la categoría en la base de datos.

En la Figura 3.29 se presenta un *script* de JQuery que se encuentra en la vista `Categoria`, se utiliza para conocer la ubicación y el nombre actual de una categoría que se va a modificar. Se ejecuta el *script* una vez que la página está cargada (línea 93); se obtiene el valor de la categoría principal (línea 95) y se almacena en un campo oculto dentro del formulario (línea 98); también el *script* detecta el cambio del nombre corto de la categoría, el cual se corresponde con el nombre de su carpeta, se almacena su valor en un campo oculto del formulario (líneas 101-102).

```
92      <script>
93          $(document).ready(function () {
94
95              var catParentName = $('#parent_id').val();
96              if (catParentName == "")
97                  catParentName = 0;
98              $("#catParentAntId").val(catParentName);
99
100             var catNombreCorto = $('#nombreCorto').val();
101             $('#nombreCorto').change(function () {
102                 $("#nombreCortoAnt").val(catNombreCorto);
103             });
104         });
105     </script>
```

Figura 3.29. *Script* de la vista para modificar una categoría

En la Figura 3.30 se presenta una parte del método `GuardarCategoria` que se utiliza para modificar la información de las categorías y sus carpetas; por lo que se verifica si se ha cambiado el nombre o la ruta de la carpeta de la categoría (línea 109), presentándose dos casos:

- Cuando se cambia el nombre corto de la categoría (línea 111), se obtiene la ubicación y su nombre anterior (línea 113), para establecer en esa ubicación la carpeta con el nuevo nombre (línea 114).
- Cuando se cambia el nombre y su ubicación; se obtiene la ubicación anterior con su nombre para cambiar la ubicación y el nombre de la carpeta (líneas 118-119).

Luego se verifica que no exista una carpeta con el mismo nombre en la ubicación especificada, de ser así se envía un mensaje de notificación a la vista sin realizar ningún cambio (líneas 123-124); en el caso contrario se realizan los cambios en el directorio y se guardan los cambios en la base de datos con el método `Guardar` (línea 133); finalmente, se establece un mensaje (líneas 134-135) para retornar a la vista que muestra la lista de categorías (línea 137).

```
106         else //Se edita la categoria
107         {
108             string folderOrigen, folderDestino;
109             if (!string.IsNullOrEmpty(nombreCortoAnt) || catParentAntId != categoria.parent_id)
110             {
111                 if (string.IsNullOrEmpty(nombreCortoAnt))
112                 {
113                     folderOrigen = ...FolderPath(0, (int)catParentAntId) + categoria.nombre_corto;
114                     folderDestino = ...FolderPath(0, (int)categoria.parent_id) + categoria.nombre_corto
115                 }
116                 else
117                 {
118                     folderOrigen = ...FolderPath(0, (int)catParentAntId) + nombreCortoAnt;
119                     folderDestino = ...FolderPath(0, (int)categoria.parent_id) + categoria.nombre_corto
120                 }
121                 if (Directory.Exists(Server.MapPath(folderDestino)))
122                 {
123                     TempData["mensajeError"] = true;
124                     TempData["mensaje"] = "No se puede modificar la categoría, ...";
125                     return Redirect("~/Formulario/ListaCategorias/");
126                 }
127                 else
128                     Directory.Move(Server.MapPath(folderOrigen), Server.MapPath(folderDestino));
129             }
130             log.Registrar(usuarioId, "Categoría", "Modificó la categoría " + categoria.nombre_largo);
131         }
132         categoria.fecha_modificacion = fechaActual;
133         categoria.Guardar();
134         TempData["mensajeError"] = false;
135         TempData["mensaje"] = "Guardado Exitoso!";
136     }
137     return Redirect("~/Formulario/ListaCategorias/");
138 }
```

Figura 3.30. Código para modificar categorías

3.1.4.2.3 Método para eliminar categorías

En la Figura 3.31 se presenta el método para eliminar una categoría, el cual recibe como parámetro el `id` de la categoría que será eliminada (línea 144); se obtiene la ubicación y el nombre del directorio de la categoría que se procederá a eliminar (línea 148), se comprueba que el directorio exista (línea 151) y se procede a eliminar únicamente cuando se encuentre vacío (línea 152). Finalmente se guardarán los cambios en la base de datos (línea 160), se establecerá un mensaje (líneas 161-162) y se retornará a la vista en la que se muestra la lista de categorías (línea 163).

```
144         public ActionResult EliminarCategoria(int id)
145         {
146             categoria.id = id;
147             string folderPath = null;
148             folderPath = dirRuta + FolderPath(0, id);
149             if (Directory.Exists(Server.MapPath(folderPath)))
150             {
151                 if (Directory.GetFiles(Server.MapPath(folderPath)).Length == 0...)
152                     Directory.Delete(Server.MapPath(folderPath));
153                 else
154                 {
155                     TempData["mensajeError"] = true;
156                     TempData["mensaje"] = "No se puede eliminar ...";
157                     return Redirect("~/Formulario/ListaCategorias/");
158                 }
159             }
160             categoria.Eliminar();
161             TempData["mensajeError"] = false;
162             TempData["mensaje"] = "Eliminación Exitosa!";
163             return Redirect("~/Formulario/ListaCategorias/");
164         }
```

Figura 3.31. Método para eliminar una categoría

3.1.4.2.4 Vista para mostrar la lista de formularios

El método `ListarFormularios` se utiliza para presentar una vista que tiene una lista con todos los formularios creados en la aplicación web.

En la Figura 3.32 se presenta la vista que muestra la lista de formularios creados, esta vista cuenta con un botón para crear nuevos formularios, con un botón para modificar un formulario y un botón para eliminar los formularios existentes; además, se tiene un botón denominado `Campos` que conduce a una vista que muestra la lista de los campos que tiene el formulario.

Figura 3.32. Vista para visualizar la lista de formularios

3.1.4.2.5 Vista para crear o modificar un formulario

Se utiliza el método `Formulario` para presentar una vista y el método `GuardarFormulario` para almacenar la información del formulario en la base de datos.

En la Figura 3.33 se muestra el código del método `Formulario`, el cual presenta una vista en la que se puede crear o modificar un formulario. Este método recibe como parámetro el `id` del formulario para indicar que se trata de un nuevo formulario (línea 178) o que se trata de un formulario existente (líneas 181-184). Cuando se va a crear un nuevo formulario se utiliza el método `ViewBag` para enviar a la vista una lista de categorías (línea 179); sin embargo, cuando se va a modificar un formulario se obtiene su información de la base de datos (línea 182) y se envía a la vista una lista de categorías (línea 183). Finalmente se retorna a la vista donde se modificará o se agregará un formulario (línea 185).

```
176     public ActionResult Formulario(int id = 0)
177     {
178         if (id == 0)
179             ViewBag.categoria_id = new SelectList(db.Categorias, "id", "nombre_corto")
180         else
181         {
182             formulario = formulario.Obtener(id);
183             ViewBag.categoria_id = new SelectList(..., formulario.categoria_id);
184         }
185         return View(formulario);
186     }
```

Figura 3.33. Método crear o editar formularios

3.1.4.2.6 Método para eliminar formularios

En la Figura 3.34 se presenta el método para eliminar formularios; se recibe como parámetro el identificador del formulario que se va a eliminar (línea 200); se utiliza el método `Eliminar` para realizar los cambios en la base de datos (línea 203); finalmente, se retorna a la vista que presenta la lista de formularios (línea 205).

```
200         public ActionResult EliminarFormulario(int id)
201         {
202             formulario.id = id;
203             formulario.Eliminar();
204             TempData["mensaje"] = "Eliminación Exitosa!";
205             return Redirect("~/Formulario/ListaFormularios/");
206         }
```

Figura 3.34. Método para eliminar un formulario

3.1.4.2.7 Vista para mostrar la lista de los campos del formulario

Para visualizar la lista de campos de un formulario se utiliza un método y una vista denominados `ListaCampos`.

En la Figura 3.35 se muestra un ejemplo de la vista con una lista de los campos de un formulario; la cual cuenta con los botones para agregar, modificar o eliminar campos en el respectivo formulario. Se tiene un botón denominado `Agregar`, el cual despliega una lista con diferentes tipos de campos que se pueden agregar al formulario. Al editar alguno de los elementos de la lista se muestra una vista en la cual se podrá cambiar el nombre del campo o simplemente cambiar el tipo de campo que puede ser texto, fecha, archivo, entre otros.

Figura 3.35. Vista con la lista de campos de un formulario

3.1.4.2.8 Vista para crear o modificar los campos de un formulario

En la Figura 3.36 se presenta el método `CampoFormulario` utilizado para agregar o modificar los campos de un formulario. El método recibe los parámetros del tipo de campo, el identificador del formulario y el identificador de un campo en el caso que exista (línea 226). Se define una lista que contiene los tipos de campos que se pueden agregar al formulario mediante el método `ViewBag` (línea 239). Se verifica si se trata de un nuevo campo (línea 240), se agrega el identificador del formulario (línea 242) y se agrega el tipo de campo seleccionado; cuando se va a modificar un campo, se obtiene la información del campo de la base de datos (línea 247). Finalmente se envía la información del campo a la vista para agregar o modificar el campo (línea 249).

```
226         public ActionResult CampoFormulario(string tipo, int idFrm, int id = 0)
227         {
228             List<SelectListItem> tipoDatos = new List<SelectListItem>()
229             {
230           new SelectListItem {Text="Select",Value="null",Selected=true },
231           new SelectListItem {Text="Texto",Value="text" },
232           new SelectListItem {Text="Texto Largo",Value="textarea"},
233           new SelectListItem {Text="Fecha",Value="date"},
234           new SelectListItem {Text="Archivo",Value="file" },
235           new SelectListItem {Text="Número",Value="number" },
236           new SelectListItem {Text="Opciones",Value="options" },
237           new SelectListItem {Text="Estado",Value="status" },
238             };
239             ViewBag.tipo = tipoDatos;
240             if (id == 0)
241             {
242                 campoFrm.formulario_id = idFrm;
243                 campoFrm.tipo = tipo;
244             }
245             else
246             {
247                 campoFrm = db.CamposFormulario.Find(id);
248             }
249             return View(campoFrm);
250         }
```

Figura 3.36. Método de la vista para crear/modificar el campo de un formulario

En la Figura 3.37 se presenta la vista utilizada para crear o modificar un campo de un formulario; primero se pide que se ingrese el nombre del campo, luego se debe seleccionar el tipo de campo que se va a utilizar. Cabe señalar que cuando se selecciona el tipo de campo estado, automáticamente se cambia el nombre del campo a `Estado`, cuyo nombre no se podrá modificar para este tipo de campo.

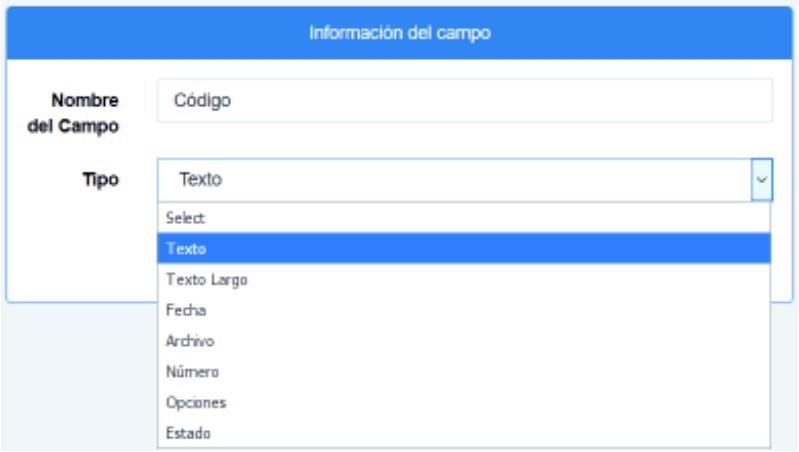


Figura 3.37. Vista para agregar o modificar campos a un formulario

3.1.4.2.9 Método para eliminar los campos de un formulario

En la Figura 3.38 se presenta el método para eliminar el campo de un formulario; se utiliza el `id` del campo para identificar el campo (línea 274) y el método `Eliminar` que ejecutará la acción (línea 275). Finalmente se utiliza el identificador `FormId` para retornar a la vista que muestra la lista de campos del formulario (línea 277).

```
272         public ActionResult EliminarCampoFormulario(int id, int formID)
273         {
274             campoFrm.id = id;
275             campoFrm.Eliminar();
276             TempData["mensaje"] = "Eliminación Exitosa!";
277             return Redirect("~/Formulario/ListaCampos/" + formID);
278         }
```

Figura 3.38. Método para eliminar un campo de un formulario

3.1.4.2.10 Vista para mostrar una lista de los registros de la documentación ingresada por los profesores

Se utiliza un método y una vista denominados `ListaRegistros` para visualizar la lista de los registros de la documentación ingresada por los profesores en cada formulario.

En la Figura 3.39 se presenta el método `ListaRegistros`, este método recibe el parámetro `id` que es el identificador de un formulario y `periodoId` que es el

identificador de un periodo, estos parámetros se utilizan para obtener únicamente los registros del formulario correspondientes al respectivo periodo académico; cabe señalar que, si no se especifica un periodo académico se asigna un valor de cero a `periodoId` (línea 291). Se utiliza la estructura de control `switch` para obtener una lista de periodos y seleccionar un periodo específico, de acuerdo a los casos que se detallan a continuación (líneas 294-306): se seleccionará el periodo actual cuando el valor del `periodoId` tenga un valor de cero (líneas 297-298); no se seleccionará ningún periodo cuando el `periodoId` tenga un valor de menos uno (línea 301); y, se seleccionará un periodo específico cuando el `periodoId` tome el valor correspondiente a dicho periodo (línea 304).

Se instancia un objeto de la clase `Tuple` (líneas 308-310) para enviar a la vista el objeto `Formulario`, una lista de campos del formulario y una lista de registros que se obtienen de la base de datos con los parámetros que recibe este método.

```
291     public ActionResult ListaRegistros(int id, int periodoId = 0)
292     {
293         ViewBag.path = dirRuta + FolderPath(id, 0);
294         switch (periodoId)
295         {
296             case 0:
297                 periodoId = periodo.PeriodoActual();
298                 ViewBag.periodoId = new SelectList(db.Periodos, "id", "nombre", periodoId);
299                 break;
300             case -1:
301                 ViewBag.periodoId = new SelectList(db.Periodos, "id", "nombre", -1);
302                 break;
303             default:
304                 ViewBag.periodoId = new SelectList(db.Periodos, "id", "nombre", periodoId);
305                 break;
306         }
307
308         var Tuple = new Tuple<Formularios, List<CamposFormulario>, List<RegistrosFormulario>>
309             (formulario.Obtener(id), campoFrm.Obtener(id), registroFrm.Listar(id, periodoId,
310             User.Identity.GetUserId())) { };
311         return View(Tuple);
312     }
```

Figura 3.39. Método para mostrar la lista con los registros de la documentación ingresada de los profesores

En la Figura 3.40 se presenta una parte de la vista para presentar la lista de los registros de la documentación ingresada por los profesores; cabe señalar que la información ingresada en cada campo del formulario es un registro en la base de datos, por tal motivo se tienen que agrupar y presentar en la vista como un solo registro. Se obtiene una lista de los identificadores de `grupo` de los registros de la

documentación de los profesores (línea 74); luego se agrupan los registros con el identificador `grupo` (línea 78) y se utiliza un `foreach` para presentar cada grupo de registros con su respectivo valor como si fuera un solo registro (líneas 78-103); también se verifica si cada grupo de registros está aprobado o rechazado para mostrar los botones que se utilizan para eliminar o modificar los registros (líneas 83-90). A su vez, se verifica si el registro es de tipo archivo (línea 80) con la finalidad de agregar un enlace para que se pueda visualizar el documento (línea 96) o para indicar al profesor que no ha subido el archivo (línea 100).

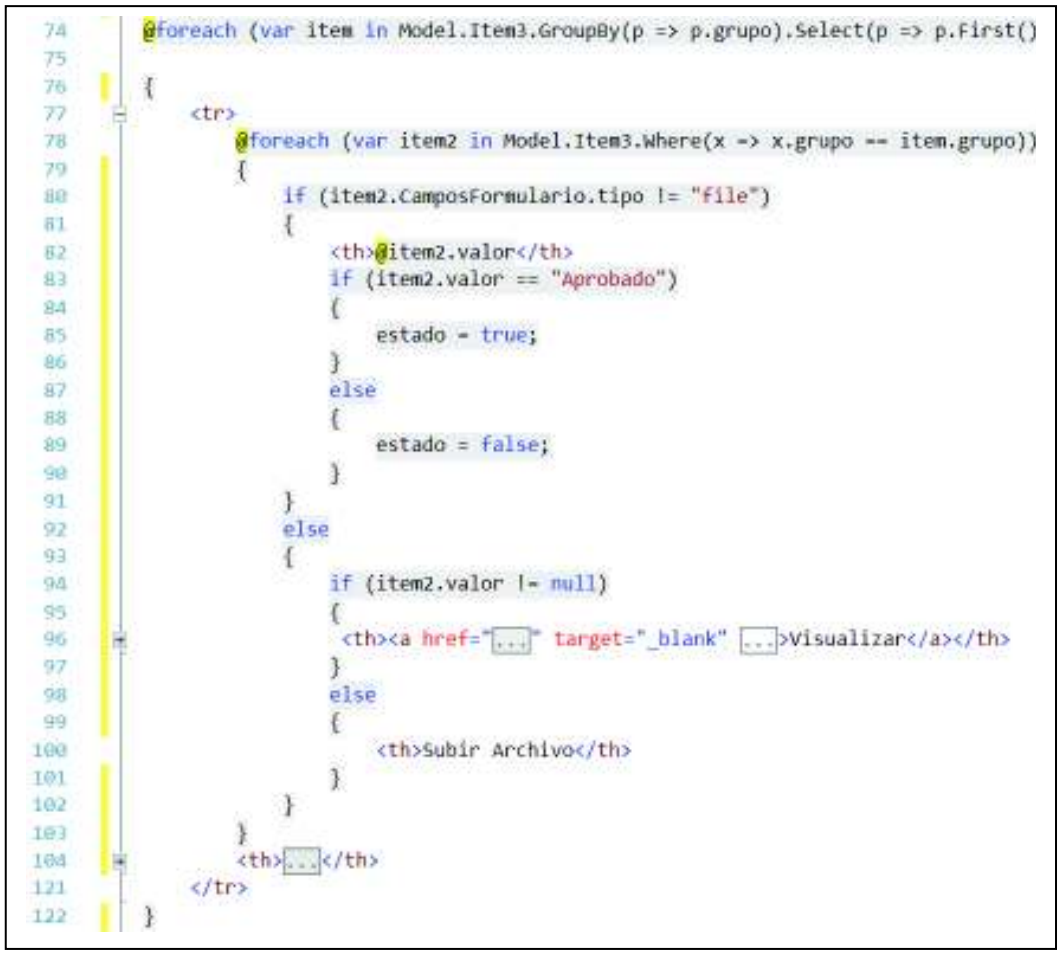


Figura 3.40. Código para mostrar la lista de la documentación ingresada por los profesores

En la Figura 3.41 se presenta la vista que muestra una lista de registros con la documentación ingresada por los profesores. Esta vista cuenta con un menú desplegable para seleccionar y obtener la información correspondiente al periodo seleccionado; se cuenta con los botones para generar archivos con la lista de

registros del formulario correspondiente en formato de Excel y PDF; y, con los botones para agregar nuevos registros, para modificar y para eliminar los registros que aún no han sido aprobados por el administrador.

Figura 3.41. Vista que muestra la lista de la documentación ingresada por los profesores

En la Figura 3.42 se presenta la vista que muestra la lista con los registros de las asignaturas asignadas a un profesor; sin embargo, se utiliza la misma vista de la Figura 3.42, pero con la diferencia que en este caso se muestra un botón que lleva al respectivo portafolio para ingresar la documentación requerida de cada asignatura.

Figura 3.42. Vista para mostrar la lista de las asignaturas de cada profesor

3.1.4.2.11 Vista para ingresar o modificar la documentación de los profesores

La vista para mostrar el formulario utilizado para ingresar o modificar la información de los profesores está compuesto por un método y una vista denominados `RegistroFormulario`; y, el método `GuardarRegistros` para almacenar la información ingresada.

En la Figura 3.43 se presenta una parte de la vista `RegistroFormulario` utilizada para presentar el formulario a los profesores, en donde tienen que ingresar la información solicitada. El formulario está formado por diferentes campos, por lo que se utiliza la estructura `foreach` para recorrer la lista de campos del presente formulario (líneas 39-128) y una estructura `switch` para presentar el respectivo campo de acurdo al tipo de dato establecido (líneas 63-126), para lo cual se utilizan *helpers* de texto (líneas 65-73), texto largo (líneas 74-82), fecha (líneas 83-91), archivo (líneas 92-104) y de un menú de opciones (líneas 105-119).

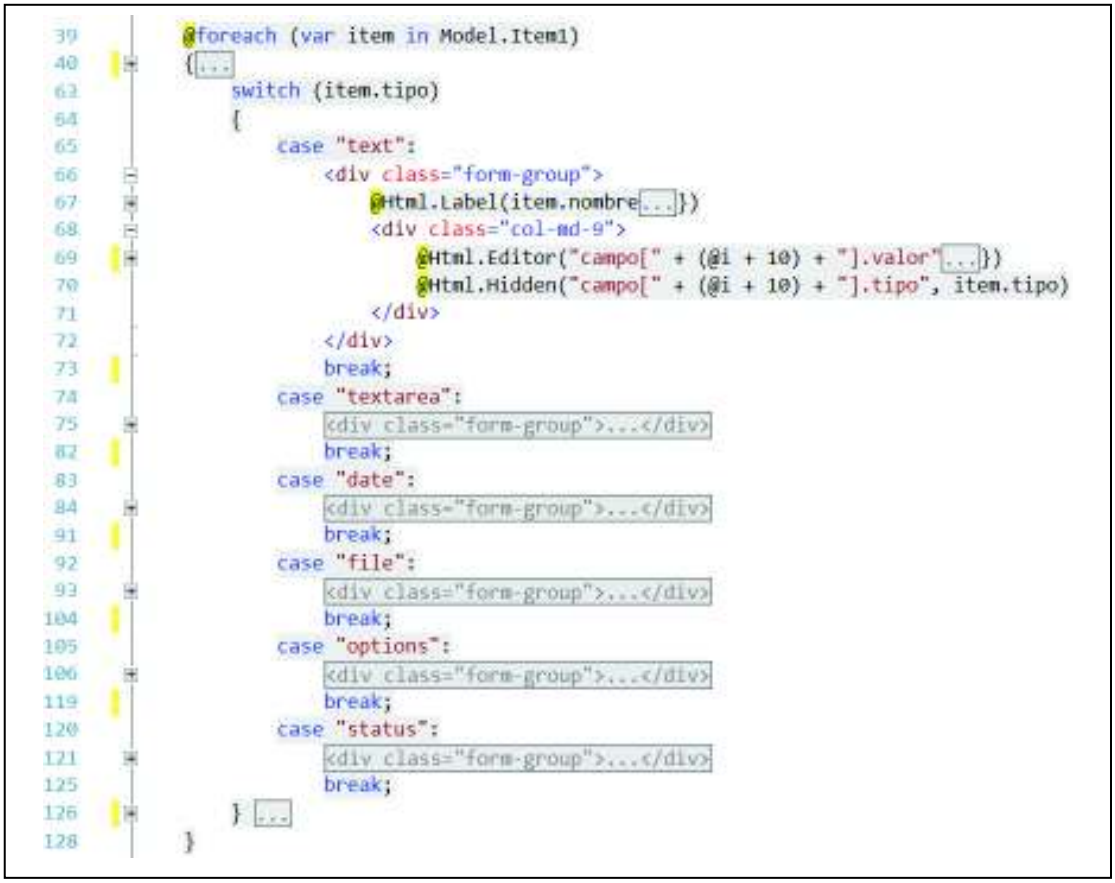


Figura 3.43. Código para presentar los campos de los formularios

En la Figura 3.44 se presenta una parte del método `GuardarRegistros` que se utiliza para guardar cada campo del formulario como un registro único en la base de datos. Se verifica si el campo del formulario es de tipo archivo (línea 518) para asignar un nombre (línea 524), para lo cual se verifica que no exista un archivo con el nombre asignado en la carpeta del formulario (línea 529), en el caso en que exista se incrementa en uno el contador del nombre del archivo (línea 531) y se vuelve a

verificar la existencia del archivo hasta que no exista un archivo con el nombre establecido (líneas 529-533). Finalmente se almacena el campo en la base de datos y se retorna a la vista que muestra la lista de registros del respectivo formulario.

```
511  foreach (var model in campo)
512  {
513      model.grupo = grupoId;
514      model.periodo_id = periodoId;
515      ...
518      if (campoFrm.tipo == "file" && file[0] != null)
519      {
520          ...
523          string fileExt = Path.GetExtension(file[fileIndex].FileName);
524          var fileName = formulario.nombre_corto + "_[" + usuario.Apellidos...;
525          int nroArchivo = 1;
526          string filePath = dirRuta + FolderPath(frmId) + fileName + nroArchivo + fileExt;
527          if (campoFrm.tipo == "file" && String.IsNullOrEmpty(model.valor))
528          {
529              while (System.IO.File.Exists(Server.MapPath(filePath)))
530              {
531                  nroArchivo++;
532                  filePath = dirRuta + FolderPath(frmId) + fileName + nroArchivo + fileExt;
533              }
534              ...
539              else
540                  model.valor = "";
541              fileIndex++;
542          }
```

Figura 3.44. Código para guardar la documentación de los profesores

En la Figura 3.45 se presenta el formulario utilizado para que los profesores ingresen la información de sus títulos académicos, el cual tiene los campos definidos por el administrador.

Figura 3.45. Vista para ingresar/modificar la documentación de los profesores

3.1.4.2.12 Método para eliminar los registros de la documentación ingresada

Para eliminar los registros se tiene el método `EliminarRegistro` presentado en la Figura 3.46, el cual que recibe como parámetros el identificador del grupo de registros y el identificador del formulario al que pertenecen (línea 582). Se obtienen el grupo de registros de la documentación ingresada utilizando el identificador `grupoId` (líneas 585-586). Se verifica si existen archivos para proceder a eliminarlos (líneas 591-596). Finalmente se elimina el grupo de registros (línea 598) y se retorna a la vista que muestra la lista de registros del formulario (línea 601).

```
582  public ActionResult EliminarRegistro(int grupoId, int frmId)
583  {
584
585      var registros = db.RegistrosFormulario.Include(x => x.CamposFormulario)
586          .Where(x => x.grupo == grupoId).ToList();
587
588      string pathArchivo = dirRuta + FolderPath(frmId);
589      foreach (var item in registros)
590      {
591          if (item.CamposFormulario.tipo == "file")
592          {
593              pathArchivo += item.valor;
594              if (System.IO.File.Exists(Server.MapPath(pathArchivo)))
595                  System.IO.File.Delete(Server.MapPath(pathArchivo));
596          }
597      }
598      registroFrm.Eliminar(grupoId);
599
600      TempData["mensaje"] = "Eliminación Exitosa!";
601      return Redirect("~/Formulario/ListaRegistros/" + frmId);
602  }
```

Figura 3.46. Método para eliminar registros de la documentación ingresada

3.1.4.2.13 Vista para validar la documentación ingresada de los profesores

En la Figura 3.47 se presenta la vista para visualizar, aprobar o rechazar la documentación ingresada por los profesores.

En la parte superior se tiene un par de menús desplegables para elegir la información de otro formulario o para elegir la información de otro periodo académico. Al frente de cada registro se tiene un menú con opciones para aprobar o rechazar un registro ingresado; únicamente en los registros que están aprobados se muestra un botón `Habilitar` para permitir que el profesor pueda modificar la información de dicho registro.

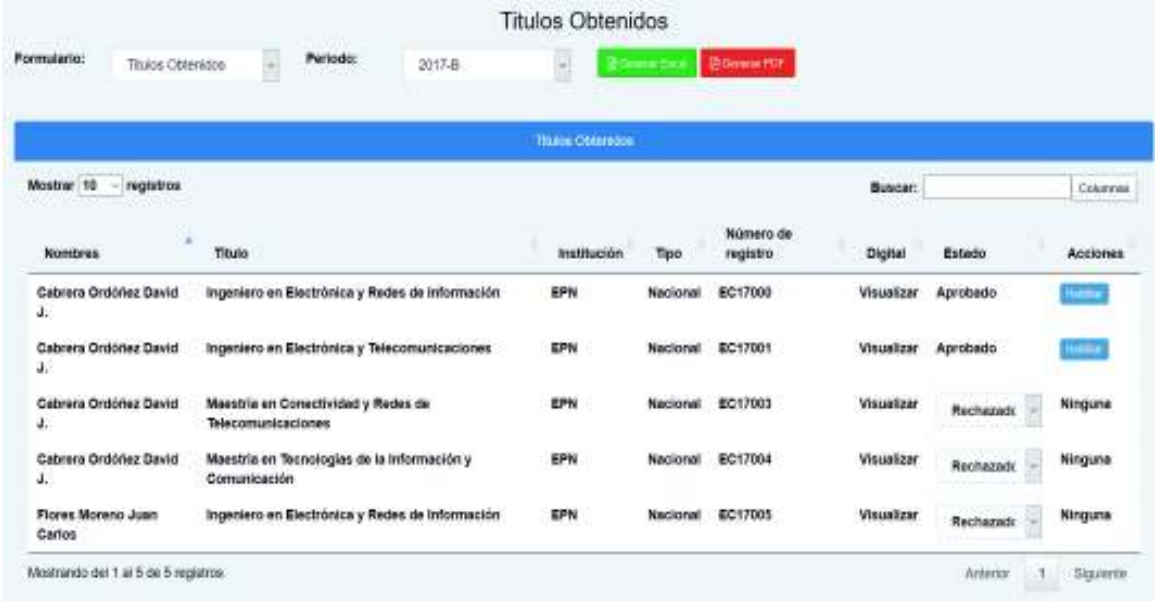


Figura 3.47. Vista para validar la documentación ingresada de los profesores

3.1.4.2.14 Vista para visualizar la lista de portafolios

En la Figura 3.48 se presenta la vista con la lista de portafolios en donde se tendrá que ingresar la documentación requerida de cada asignatura. Cuenta con botones para agregar un nuevo portafolio, editar y eliminar. Además, se tiene un botón denominado `Bloques` que se utiliza para agregar bloques y campos al portafolio académico.

Figura 3.48. Vista para visualizar la lista de portafolios

3.1.4.2.15 Vista para crear o modificar portafolios

La creación o edición de los portafolios es semejante a la creación de los formularios, con la particularidad en que los campos están organizados en bloques personalizables. En la Figura 3.49 se presenta la vista de un portafolio en la que se pueden agregar bloques que contienen grupos de campos; los bloques están

definidos por un rango de fechas para que los profesores ingresen la documentación requerida dentro de las fechas establecidas.

Dentro de cada bloque se tienen botones para agregar, modificar o eliminar un campo en el que se ingresará la documentación requerida de los profesores.

Figura 3.49. Vista de un portafolio para agregar bloques y campos

3.1.4.2.16 Eliminar los portafolios

El método utilizado para eliminar un portafolio académico es semejante al método descrito anteriormente para eliminar un formulario.

3.1.4.2.17 Ingresar la documentación de los portafolios académicos

En la Figura 3.50 se presenta el código de JQuery utilizado para evitar que se ingrese información en los campos de los bloques, fuera de las fechas establecidas por el administrador.

El código se ejecutará cada vez que se envíe un formulario de la vista (líneas 235-257) y realizará lo siguiente: se obtendrán las fechas establecidas en el bloque (líneas 236-238), se obtendrá la fecha actual (línea 244), se verifica que la fecha actual esté dentro del rango de las fechas especificadas para enviar la información del formulario (líneas 245-247), caso contrario se detiene el envío del formulario (línea 255) y se muestra un mensaje indicando que no se puede enviar el formulario por estar fuera del límite de las fechas establecidas (líneas 254).

```
235         $("form").submit(function (e) {
236             var fi = $(this).closest('form').find("#fecha_inicio").html().split('/');
237             var fechaInicio = new Date(fi[2], fi[1] - 1, fi[0], 24, 00);
238             ...
244             var fechaActual = new Date();
245             if (fechaInicio <= fechaActual && fechaActual <= fechaFin) {
246                 $("form").submit();
247             }
248             else
249                 if (fechaActual <= fechaExtension) {
250                     alert('Se encuentra fuera del plazo para subir la información. ...
251                     $("form").submit();
252                 }
253                 else {
254                     alert('Fecha fuera del límite. \n\nConsulte con el administrador.')
255                     e.preventDefault(e);
256                 }
257         });
```

Figura 3.50. Código de JQuery para validar las fechas de los portafolios

En la Figura 3.51 se presenta la vista utilizada para que los profesores ingresen la información solicitada de cada asignatura, se tendrán diferentes bloques de acuerdo a las necesidades. Cada bloque tiene un formulario web con botones para ingresar los archivos solicitados y un botón denominado `Guardar` para enviar o no el respectivo formulario para almacenar la información siempre y cuando se encuentre dentro del rango de fechas indicadas.

Figura 3.51. Vista para ingresar la documentación de los portafolios académicos

3.1.4.2.18 Vista para validar la documentación de los portafolios académicos

En la Figura 3.52 se presenta la vista para validar la documentación de los registros ingresados en cada portafolio. Se tiene un par de menús en la parte superior para

seleccionar otro portafolio o para ver los registros de un bloque específico. Además, se tiene un menú para validar cada uno de los registros ingresados o un botón denominado `Habilitar` para permitir al profesor modificar el registro específico.

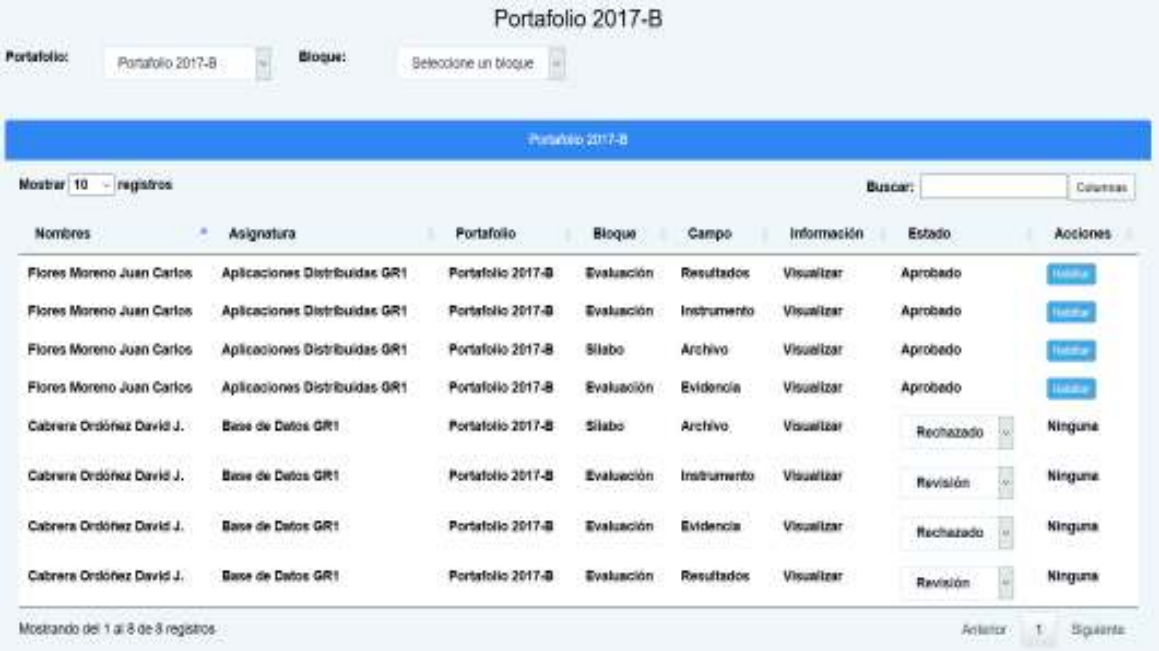


Figura 3.52. Vista para validar la documentación de los portafolios

3.1.4.3 Módulo para generar, visualizar e imprimir informes

Los informes tienen encabezados y los pies de página personalizables por el súper administrador, por tal motivo se codifican métodos y vistas que permitan realizar esta actividad.

3.1.4.3.1 Vista para configurar el encabezado y del pie de página de los informes

Los métodos para establecer el encabezado y los pies de página de los informes se encuentran codificados en el controlador `ConfiguracionController`.

En la Figura 3.53 se presenta el *script* utilizado para inicializar y agregar elementos al editor de textos TinyMCE que se utiliza para establecer el encabezado y el pie de página de los informes en formato PDF.

Se inicializa el editor de textos (línea 182); se agregan los *plugin* (líneas 197-198) y las barras de herramientas con los botones que realizan las funciones de copiar, pegar, crear tablas, agregar formatos de texto, entre otros (línea 200-201); y, se configura para que se almacenen las imágenes automáticamente (líneas 207-209).

Se agrega un botón para que se muestre un modal con la lista de palabras clave y sus reemplazos (líneas 225-237).

Finalmente se crea una función para reemplazar los signos: mayor que (>) y menor que (<) del contenido generado en el editor de textos para evitar las restricciones al guardar el documento XML en la base de datos (líneas 239-242).

```
182         tinyMCE.init({
183             selector: 'textarea',
184             branding: false,
185             resize: true,
186             statusbar: true,
187             menubar: false,
188             theme: 'modern',
189             ...
197             plugins: ["advlist autolink lists link image charmap hr anchor",
198                     ...
200             toolbar: [" styleselect  | fontselect |  ...
201                     ...
203             encoding: "xml",
204             ...
207             automatic_uploads: true,
208             images_upload_url: '/Configuracion/TinyMceUpload?name=@Model.tipo',
209             images_reuse_filename: true,
210         ...
225             setup: function (editor) {
226                 ...
232                         $('#myModal').modal('show');
233                     ...
235             }
236
237         });
238
239         function saveIt() {
240             var html = $('#text').val();
241             $('#text').val(html.replace(/</g, "&lt;").replace(/>/g, "&gt;"));
242         }
```

Figura 3.53. *Script* para agregar el editor de textos

En la Figura 3.54 se presenta una parte de la vista con un editor de textos utilizado para establecer el encabezado y el pie de página de los informes en formato PDF.

El editor de textos permite ingresar textos y establecer su respectivo formato. Se utiliza también para agregar tablas e imágenes. La particularidad más importante es que se utilizan palabras clave que serán reemplazadas para que se pueda visualizar su correspondiente información en el informe; la lista de las palabras clave con sus respectivos reemplazos se pueden visualizar dando clic en el ícono ( ).

Figura 3.54. Editor de texto para establecer el encabezado y el pie de página de los informes en PDF

En la Figura 3.55 se presenta una parte del código JQuery utilizado en la vista `FormatoDocumento` para establecer el formato de los encabezados y de los pies de página de los informes en formato de Excel. Se tiene el código que permite agregar líneas con *inputs* para ingresar texto en el encabezado (líneas 289-305), también le permite eliminar las líneas con los *inputs* que no se vayan a utilizar (líneas 307-309). El código presentado se utiliza únicamente para establecer el encabezado de los informes, mientras que para establecer el pie de página de los informes se utiliza un bloque de código similar.

```
287   var max_op = 15;
288   var x = 1;
289   $('#agregarEncabezado').click(function (e) {
290       e.preventDefault();
291       var numId = $("#opcionesEncabezado div.form-group").length;
292       x = numId;
293       if (x < max_op) {
294           $('#opcionesEncabezado').append('<div ...>' +
295       '<label ...>Línea ' + x + '</label>' +
296       '<div class="col-md-6">' +
297           '<input type="text" ... name="encabezado"/></div>' +
298           '<div class="col-md-4">' +
299           '<select ...>' + $('#tamFuenteBase').html() + '</select></div">' +
300           '<select ...>' + $('#tipoFuenteBase').html() + '</select></div">' +
301           '<select ...>' + $('#alignFuenteBase').html() + '</select></div">' +
302       '</div> <div class="col-md-1">' +
303       '<a ... id="eliminarEncabezado">Eliminar</a></div></div>');
304       }
305   });
306
307   $('#opcionesEncabezado').on("click", "#eliminarEncabezado", function (e) {
308       e.preventDefault(); $(this).parent().parent('div').remove(); x--;
309   });
```

Figura 3.55. Código JQuery para agregar y eliminar elementos HTML

En la Figura 3.56 se presenta la vista `FormatoDocumento` que se utiliza para establecer el encabezado y el pie de página de los informes en formato de Excel. Se tiene un botón `Agregar Línea` que agrega un bloque HTML que tiene un cuadro de texto donde se ingresará el texto que se visualizará en el informe, se tiene que especificar el tamaño de fuente, el tipo de fuente y su alineación; se pueden agregar los bloques que sean necesarios. Al frente de cada cuadro de texto se tiene un botón para eliminar el respectivo bloque. Además, se pueden ingresar palabras clave según sea necesario.

Figura 3.56. Establecer el formato del encabezado y del pie de página de los informes en formato Excel

3.1.4.3.2 Vista para visualizar los informes e imprimir

La vista denominada `Informe` presentará un informe en el que se mostrará la documentación ingresada por el profesor y que fue aprobada por el administrador en cada uno de los formularios. En la vista se ha incluido un *script* de JQuery, el mismo que se puede observar en la Figura 3.57 y realiza lo siguiente: permite imprimir la página actual con el método `print` (líneas 175-177), permite recargar la página con el método `reload` (líneas 175-177) y permite eliminar un bloque de información correspondiente a un formulario dentro de la vista con el método `remove`, de tal forma que se puedan imprimir únicamente los formularios que se requieran (líneas 180-182).

```
169         //Función para imprimir la página
170         function imprimir() {
171             window.print();
172         }
173
174         //Recargo la página
175         $('#recargar').click(function () {
176             location.reload();
177         });
178
179         //Elimino la información de un formulario
180         $('.eliminar_bloque').on("click", "#eliminar", function (e) {
181             e.preventDefault();
182             $(this).parent().parent().parent().parent().remove();
183         });
```

Figura 3.57. Código JQuery de la vista para visualizar los informes y su impresión

En la Figura 3.58 se presenta un parte de la vista de los informes que podrán visualizar los profesores; contará con los botones para imprimir la página, recargar la página y para eliminar la información de un formulario al dar clic en la equis (x).

Figura 3.58. Vista para visualizar los informes e imprimir

3.1.4.3.3 Generar un archivo de informe en formato de Excel

El código para generar un archivo en formato de Excel con la información de la documentación ingresada por los profesores es muy similar al código utilizado para generar una lista de usuarios en formato de Excel (ver la página 104); con la particularidad en que se realiza el reemplazo de las palabras clave y se realizan cambios a la tabla temporal que tiene la información de los registros.

Se utiliza un método denominado `RemplazarPalabras` para remplazar las palabras clave que se ingresan en el encabezado y en el pie de página. Una parte del código del método para reemplazar las palabras clave se muestra en la Figura 3.59, en donde se valida que la propiedad del encabezado no se encuentre vacía (línea 909), para proceder a realizar el reemplazo de cada una de las palabras clave (línea 910).

En la Figura 3.60 se tiene el código para crear una tabla temporal con los registros aprobados de la documentación ingresada (línea 668-669) y con los registros de la documentación que no tienen estado (línea 676); de la tabla temporal se eliminará las columnas que tengan los nombres de los archivos y el estado de los registros (líneas 675-676).

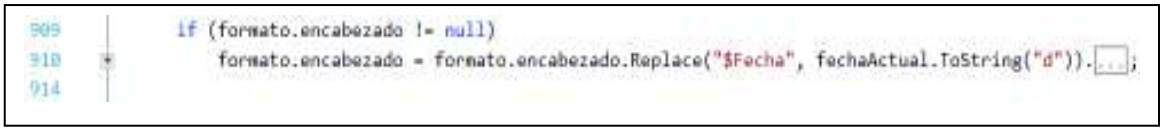


Figura 3.59. Código para reemplazar las palabras clave

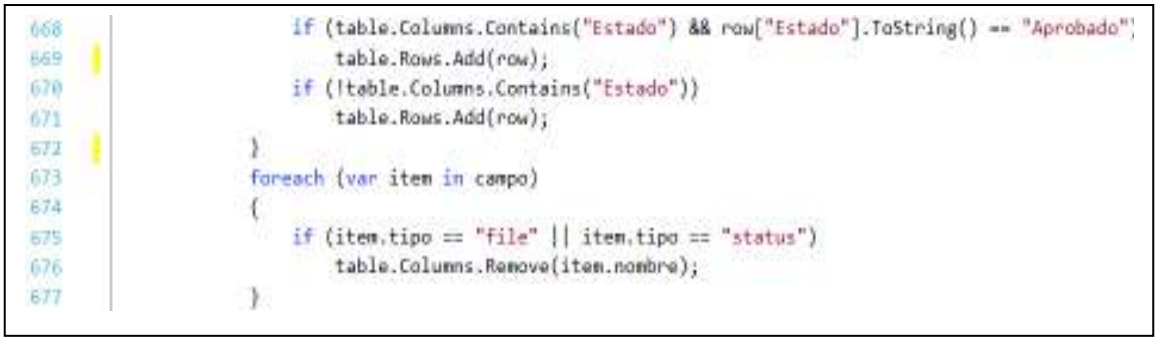


Figura 3.60. Código para agregar registros aprobados y eliminar columnas innecesarias

3.1.4.3.4 Generar un archivo de informe en formato PDF

El método para generar un archivo PDF es muy similar al método utilizado para generar la lista de usuarios en formato PDF (ver la página 105), pero con la particularidad en que se presentarán únicamente los registros aprobados; además, se reemplazarán las palabras clave y las columnas innecesarias serán eliminadas, tal es el caso de la columna que tiene los nombres de los archivos y la columna del estado de los registros.

3.1.4.4 Módulo para generar notificaciones a los profesores sobre la información que se tiene que ingresar

Para enviar correos electrónicos con notificaciones a los profesores es necesario ejecutar un hilo en segundo plano. Se agrega la línea `App_Start.BackgroundThread.StartNotificaciones()` en el archivo `Global.asax` y se indica que se creó una clase denominada `BackgroundThread` en la carpeta `App_Start`, de donde se ejecutará el método `StartNotificaciones`. Los mensajes que se envíen serán personalizados por el súper administrador; además, se contará con una vista en la cual se podrán habilitar o deshabilitar las notificaciones y establecer la hora de envío de los correos electrónicos de notificación.

3.1.5.4.1 Métodos para enviar las notificaciones a los profesores

En la Figura 3.61 se presenta el método `StartNotificaciones` en el que se instancia un hilo para que ejecute el método `StartJob` (línea 18), el hilo correrá en segundo plano (línea 19), se llamará a `BackgroundNotificaciones` (línea 20) e iniciará la ejecución junto con la aplicación web (línea 21).

En el método `StartJob` instancia la clase `Notificaciones` (línea 26) para acceder a su método `EnvioNotificacion`, el cual tiene la lógica para generar los correos electrónicos de notificaciones. El método `EnvioNotificacion` realiza un barrido de la base de datos verificando que los usuarios tengan ingresada la documentación solicitada, en el caso en que no tengan ingresada la documentación solicitada o que se encuentre rechazado algún registro, se procederá a enviar un correo de notificación.

Se obtiene la hora actual del sistema (línea 27), se obtiene del archivo de configuraciones la hora de envío de la notificación y el número de días que se tiene que esperar para volver a enviar las notificaciones (líneas 28-29). Se realiza una resta entre la hora del envío de notificaciones y la hora actual (línea 30) para esperar el tiempo calculado (línea 31) y enviar las notificaciones a la hora establecida. Se instancia un `timer` (línea 32) y se establece un intervalo de tiempo en el que se enviarán las notificaciones, de acuerdo al número de días especificado

en la variable `dia` (líneas 33-36). Con el evento `Elapsed` se indica que se ejecute el método `EnvioNotificacion` que genera las notificaciones (línea 34).

Finalmente se habilita el *timer* (línea 35), se habilita la propiedad de reinicio para que se ejecute constantemente en cada intervalo de tiempo (línea 36) y se proceda a iniciar el temporizador (línea 37).

```
16      public static void StartNotificaciones()
17      {
18          var thread = new Thread(new ThreadStart(StartJob));
19          thread.IsBackground = true;
20          thread.Name = "BackgroundNotificaciones";
21          thread.Start();
22      }
23
24      private static void StartJob()
25      {
26          Notificaciones notificaciones = new Notificaciones();
27          TimeSpan horaActual = DateTime.Now.TimeOfDay;
28          TimeSpan horaNotificacion = TimeSpan.Parse(ConfigurationManager.AppSettings["horaNot"].ToSt
29          int dia = Convert.ToInt32(ConfigurationManager.AppSettings["horaNot"].ToString());
30          int tiempoEspera = (int)horaNotificacion.Subtract(horaActual).Duration().TotalMilliseconds;
31          Thread.Sleep(tiempoEspera);
32          var timer = new System.Timers.Timer();
33          timer.Interval = dia * 24 * 3600 * 1000;
34          timer.Elapsed += new ElapsedEventHandler(notificaciones.EnvioNotificacion);
35          timer.Enabled = true;
36          timer.AutoReset = true;
37          timer.Start();
38      }
```

Figura 3.61. Métodos para generar las notificaciones de los profesores

3.1.5.4.2 Vista para configurar las notificaciones

En la Figura 3.62 se presenta la vista para realizar las configuraciones del envío de correos electrónicos con recordatorios a los profesores.

La vista cuenta con casillas de verificación para habilitar o deshabilitar el envío de notificaciones tanto de los formularios como de los portafolios; se tiene un campo para especificar la hora y otro campo para indicar cada cuantos días se tienen que enviar los correos de notificaciones; un campo que se llenará automáticamente con la dirección URL del servidor, esta dirección se utilizada para mostrar en los mensajes de correo electrónico los hipervínculos de acceso directo a los formularios o a los registros rechazados.

Las configuraciones de esta vista se guardarán en el archivo de configuración denominado `Web.config`. Cabe señalar, se han habilitado las conexiones seguras mediante SSL para acceder a la aplicación web, lo cual se puede visualizar en la URL de la Figura 3.62.

Figura 3.62. Vista para configurar las notificaciones a los profesores

En la Figura 3.63 se presenta una vista en la que el súper administrador tiene que establecer el mensaje de los correos de notificación. Puede usar las palabras clave `$Usuario` para que se escriba el nombre del profesor, `$Formularios` para que se agregue la información de los formularios en los que tiene que corregir o ingresar información solicitada y `$Portafolios` para que se muestre en el correo la información que debe corregir o ingresar de cada una de las asignaturas.

Figura 3.63. Vista para escribir el mensaje de notificaciones a los profesores

3.1.4.5 Módulo para registrar y visualizar los eventos (*logs*)

Para registrar los eventos de los usuarios se instancia el objeto `Logs` en cada uno de los controladores y se utiliza el método `Registrar`, este método se agrega a cada uno de los métodos de los controladores para almacenar el evento realizado. Se van a conservar únicamente los últimos 50 registros de eventos de cada usuario; por tal motivo, cada vez que inicia sesión un usuario, se verificará que no se exceda de 50 eventos, caso contrario se procederá a borrar todos los eventos en exceso, empezando por el más antiguo.

3.1.4.5.1 Método para almacenar los eventos de los usuarios

En la Figura 3.64 se presenta el método `Registrar` utilizado para almacenar los eventos de los usuarios. Recibe los parámetros de la identidad del usuario, la acción y el detalle del evento realizados (línea 34); se obtiene la dirección IP local del equipo utilizado por el usuario (líneas 37-46), con la finalidad de registrar el equipo de donde se produjo la acción; también se obtiene la fecha y hora a la que se produjo el evento (línea 48).

Finalmente se procede a asignar los valores en los atributos del objeto `Logs` y posteriormente se utiliza el método `Guardar` para almacenar el evento en la base de datos.

```
34          public void Registrar(string usuarioId, string accion, string detalle)
35          {
36              //Obtener la IP Local
37              IPHostEntry host = Dns.GetHostEntry(Dns.GetHostName());
38              string localIP = "";
39              foreach (IPAddress ip in host.AddressList)
40              {
41                  if (ip.AddressFamily == AddressFamily.InterNetwork)
42                  {
43                      localIP = ip.ToString();
44                      break;
45                  }
46              }
47              //Fecha
48              DateTime fechaActual = DateTime.Now;
49
50              //Almacenar el log
51              usuario_id = usuarioId;
52              fecha = fechaActual;
53              ip = localIP;
54              this.accion = accion;
55              this.detalle = detalle;
56              Guardar();
57          }
```

Figura 3.64. Método para almacenar los eventos de los usuarios

3.1.4.5.2 Vista para mostrar los eventos de los usuarios

En la Figura 3.65 se presenta la vista que muestra los eventos de los usuarios. Los eventos se presentan en una tabla que tiene diferentes columnas de tal forma que se pueda conocer con detalle la acción realizada.

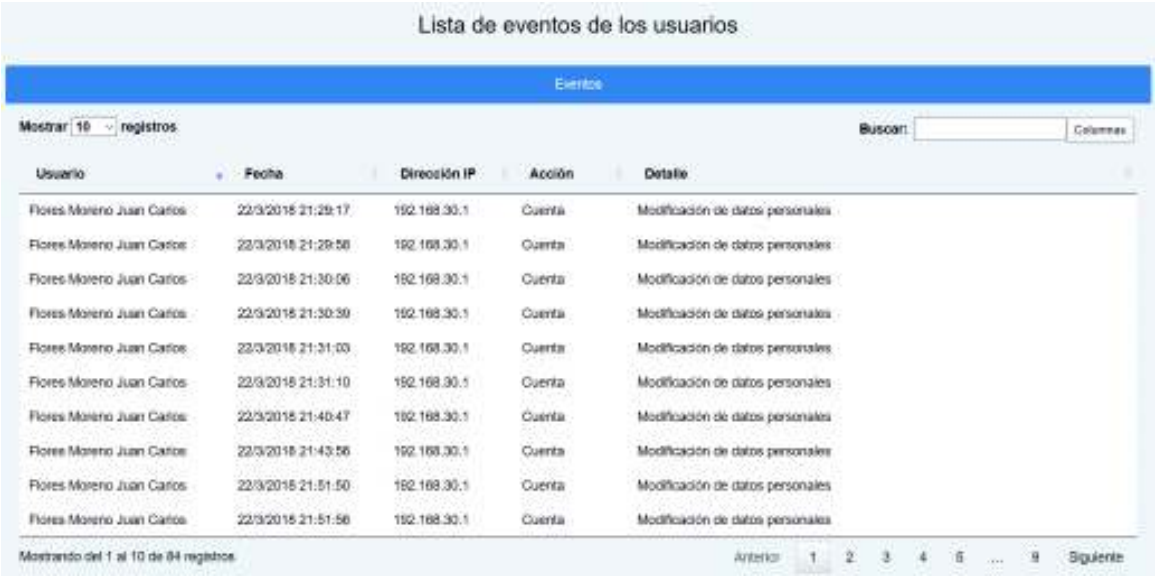


Figura 3.65. Vista para mostrar los eventos de los usuarios

3.1.4.6 Tareas adicionales de la aplicación web

Se realizan algunas tareas adicionales que son importantes para el correcto funcionamiento de la aplicación web.

3.1.4.6.1 Vista para establecer el encabezado de los mensajes de correo electrónico

En la Figura 3.66 se presenta la vista utilizada para establecer los encabezados de los correos que se enviarán cuando se registra una nueva cuenta de usuario, con las instrucciones para restablecer una contraseña y con la información de los registros de la documentación rechazada; además, se puede establecer el asunto del mensaje del correo electrónico.

Cabe señalar que se utiliza el editor de textos para establecer los encabezados de los mensajes de correo electrónico, donde se pueden agregar imágenes, tablas, hipervínculos, entre otros. Las imágenes no se adjuntan al mensaje de correo electrónico, por lo que es necesario agregar únicamente enlaces de imágenes que se encuentren en Internet.

Figura 3.66. Vista para establecer el encabezado de los mensajes de correo electrónico

3.1.4.6.2 Vista para configurar una cuenta de correo electrónico para el envío de mensajes

En la Figura 3.67 se presenta la vista utilizada para ingresar la información necesaria de una cuenta de correo electrónico, la cual se utilizará para el envío de los mensajes de correo electrónico. La información ingresada en el formulario de esta vista será almacenada en el archivo de configuración `Web.conf`.

Figura 3.67. Vista para configurar una cuenta de correo electrónico

3.1.4.6.3 Vista para mostrar la lista de los periodos académicos

La vista que presenta la lista de los periodos académicos es muy similar a las vistas utilizadas para visualizar la lista de categorías, formularios y portafolios.

En la Figura 3.68 se presenta el *script* para inicializar la librería *Data Tables*, utilizada para mostrar una tabla dinámica que permite paginar los registros, ordenar las columnas y realizar búsquedas. Se agrega en la tabla HTML el identificador `periodos`, para indicar que sobre esta tabla se tiene que agregar las propiedades de la librería en cuestión (línea 99).

Se configuran algunas propiedades de la librería como: habilitar la paginación (línea 100), permitir cambiar el número de filas que se visualizan (línea 101), permitir las búsquedas (línea 102), permitir ordenar las columnas (línea 103), permitir mostrar la información sobre el número de registros (línea 104), permitir ajustar el contenido a la columna (línea 105). La tabla mostrará información en inglés, por tal motivo se tiene que realizar la respectiva traducción de cada una de sus propiedades (líneas 106-129); como ejemplo se muestra la traducción de la propiedad procesamiento, la cual mostrará un mensaje cuando se esté cargando la información de la tabla (línea 107).

```
99          $("#periodos").DataTable({
100             "paging": true,
101             "lengthChange": true,
102             "searching": true,
103             "ordering": true,
104             "info": true,
105             "autoWidth": true,
106             language: {
107                 "sProcessing": "Procesando...",
108               ...
129             }
130         });
131     });
```

Figura 3.68. *Script* para inicializar la librería *Data Tables*

3.1.4.6.4 Crear o modificar un periodo académico

El método para crear o modificar un periodo académico es muy similar a los métodos utilizados con el mismo propósito en las categorías, formularios y portafolios.

3.1.4.6.5 Eliminar periodos académicos.

El método para eliminar los periodos académicos es muy similar a los métodos utilizados para eliminar las categorías y los formularios.

3.2 PRUEBAS DE LA APLICACIÓN

3.2.1 PRUEBAS DE INTEGRACIÓN

Con la realización de las pruebas de integración se verificó que los componentes de la aplicación funcionan correctamente actuando en conjunto.

Las pruebas se realizaron durante y después de la codificación de los diferentes métodos y vistas de la aplicación web. Cada uno de los errores detectados fueron corregidos de tal forma que la aplicación web funciona de acuerdo a los requerimientos. Cabe señalar que se escribió un resumen de los errores detectados en el *backlog*, el cual se encuentra en el Anexo B.

3.2.2 PRUEBAS DE VALIDACIÓN

Estas pruebas permitieron evaluar la aplicación web para determinar que se cumple con lo especificado en los requerimientos. Las pruebas de validación se realizaron con la participación del M.Sc. David Mejía que es el director del presente Trabajo de Titulación, el Ph.D. Diego Reinoso que es profesor del DETRI y con el M.Sc. Jorge Carvajal que es el Coordinador de Ingeniería en Electrónica y Telecomunicaciones. Los documentos generados en las pruebas de validación se encuentran en el Anexo F.

Los resultados de las pruebas de validación se registraron en diferentes tablas que tienen una columna para indicar la prueba realizada, una columna para indicar si pasó o no la prueba y una columna en la que se registran las observaciones realizadas a las tareas.

En la Tabla 3.3 se presenta la lista de las pruebas de validación que se realizaron al módulo de administración y autenticación de usuarios. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones en ninguna de las tareas del módulo.

Tabla 3.3. Pruebas de validación del módulo de autenticación y administración

MÓDULO DE ADMINISTRACIÓN Y AUTENTICACIÓN DE USUARIOS		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Registrar uno o varios usuarios.	SI	
Autenticarse en la aplicación web.	SI	
Asignar un rol a un usuario.	SI	
Activar o desactivar un usuario.	SI	
Visualizar y editar la información de un usuario.	SI	
Eliminar un usuario.	SI	
Visualizar la lista de usuarios.	SI	
Generar un archivo en formato Excel y PDF con la lista de los usuarios.	SI	

En la Tabla 3.4 y la Tabla 3.5 se presenta la lista de las pruebas de validación que se realizaron al módulo para la recepción de la documentación de los profesores. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones.

Tabla 3.4. Pruebas de validación del módulo de recepción de documentos (Parte 1 de 2)

MÓDULO PARA LA RECEPCIÓN DE LA DOCUMENTACIÓN DE LOS PROFESORES		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Visualizar la lista de categorías.	SI	
Crear y editar una categoría.	SI	
Eliminar una categoría.	SI	
Visualizar la lista de formularios.	SI	
Crear y editar un formulario.	SI	
Eliminar un formulario.	SI	
Visualizar la lista de los campos de un formulario.	SI	
Agregar y editar los campos de un formulario.	SI	
Eliminar un campo de un formulario.	SI	
Visualizar lista de los registros de la documentación ingresada por los profesores.	SI	

Tabla 3.5. Pruebas de validación del módulo de recepción de documentos (Parte 2 de 2)

MÓDULO PARA LA RECEPCIÓN DE LA DOCUMENTACIÓN DE LOS PROFESORES		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Eliminar un registro de la documentación ingresada de un profesor.	SI	
Aprobar o rechazar la documentación ingresada de los profesores.	SI	
Visualizar la lista de portafolios.	SI	
Crear y modificar un portafolio.	SI	
Eliminar un portafolio.	SI	
Ingresar la documentación y su posterior visualización en un portafolio académico.	SI	
Aprobar o rechazar los documentos ingresado en los portafolios académicos.	SI	

En la Tabla 3.5 se presenta la lista de las pruebas de validación que se realizaron al módulo para generar, visualizar e imprimir informes. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones.

Tabla 3.5. Pruebas de validación del módulo de informes

MÓDULO PARA GENERAR, VISUALIZAR E IMPRIMIR INFORMES		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Verificar que el informe tenga el encabezado y del pie de página establecidos.	SI	
Visualización de informes y su impresión.	SI	
Generar un archivo en formato Excel con los registros de la documentación ingresada.	SI	
Generar un archivo en formato PDF con la documentación ingresada.	SI	

En la Tabla 3.6 se presenta la lista de las pruebas de validación que se realizaron al módulo para generar notificaciones a los profesores sobre la información que

deben ingresar. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones.

Tabla 3.6. Pruebas de validación del módulo de notificaciones

MÓDULO PARA GENERAR NOTIFICACIONES A LOS PROFESORES SOBRE LA INFORMACIÓN QUE SE DEBE INGRESAR		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Revisar el correo electrónico recibido de la notificación al profesor.	SI	
Realizar la configuración de las notificaciones de los profesores.	SI	

En la Tabla 3.7 se presenta la lista de las pruebas de validación que se realizaron al módulo para registrar y visualizar los eventos. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones.

Tabla 3.7. Pruebas de validación del módulo de eventos

MÓDULO PARA REGISTRAR Y VISUALIZAR LOS EVENTOS		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Generar eventos.	SI	
Visualizar los eventos de los usuarios.	SI	

En la Tabla 3.8 se presenta la lista de las pruebas de validación que se realizaron a varias tareas adicionales que son importantes para la aplicación web. El resultado indica que se superaron todas las pruebas realizadas y no se registraron observaciones.

Tabla 3.8. Pruebas de validación de tareas adicionales

TAREAS ADICIONALES		
PRUEBAS	**SI/NO**	**OBSERVACIONES**
Visualizar y modificar la información de la cuenta SMTP.	SI	
Visualizar los periodos académicos.	SI	
Crear y editar un periodo académico.	SI	
Eliminar un periodo académico.	SI	

3.2.3 PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación verificaron que la aplicación web funciona de acuerdo con las especificaciones. Las pruebas se realizaron con la participación del M.Sc. David Mejía que es el director del Trabajo de Titulación y el M.Sc. Jorge Carvajal que es el Coordinador de la Carrera de Ingeniería en Electrónica y Telecomunicaciones.

A continuación, se presenta un resumen en la Tabla 3.9, donde se muestran los requerimientos de la aplicación web y el resultado de las pruebas de validación para contrastar que la aplicación web cumple con las especificaciones. Como se puede observar en las tablas, se cumple con lo propuesto en este Trabajo de Titulación.

Tabla 3.9. Pruebas de aceptación de la aplicación web (Parte 1 de 2)

REQUERIMIENTOS	SI/NO	OBSERVACIONES
Permitirá al administrador crear, modificar y eliminar usuarios.	SI	
Permitirá gestionar roles de usuario.	SI	
Permitirá la autenticación de usuarios.	SI	
Permitirá al administrador gestionar categorías para organizar la información recolectada de los profesores.	SI	
Permitirá al administrador crear, modificar o eliminar formularios con sus respectivos campos, donde los profesores ingresarán la información solicitada.	SI	
Los profesores ingresarán la información en cada uno de los formularios generados por el administrador, además la información se podrá visualizar, modificar o eliminar.	SI	
El administrador tendrá la capacidad de aprobar o no la información ingresada por el profesor, también podrá habilitar la edición de dicha información una vez aprobada	SI	
Permitirá imprimir informes de la información ingresada tanto a los profesores como a los administradores.	SI	

Tabla 3.9. Pruebas de aceptación de la aplicación web (Parte 2 de 2)

REQUERIMIENTOS	SI/NO	OBSERVACIONES
Se generará eventos (*logs*) de las actividades realizadas por los profesores en la aplicación web.		
La aplicación web generará notificaciones a los profesores sobre la información que tienen que ingresar.	SI	
La aplicación permitirá agregar, modificar o eliminar los periodos académicos con sus respectivas fechas.	SI	

Cabe señalar que los profesores que participaron en las pruebas comprobaron que la aplicación web es personalizable acoplándose a las necesidades durante la creación de formularios y portafolios, en donde los profesores tendrán que ingresar la información solicitada; también se pueden personalizar los encabezados, los pies de página y los correos de notificaciones.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las conclusiones y recomendaciones obtenidas como resultado del desarrollo del presente Trabajo de Titulación.

4.1 CONCLUSIONES

- Durante la obtención de los requerimientos para el desarrollo de la aplicación web se conoció que los profesores tienen inconvenientes con la entrega de la documentación para el proceso de acreditación de las carreras, dado que en algunas ocasiones se extravía la documentación presentada, en otros casos no se presenta la documentación a tiempo, entre otros.
- La aplicación web se ha desarrollado en dos proyectos, el primer proyecto contiene la lógica, las vistas y los recursos utilizados, mientras que el segundo proyecto tiene las entidades del modelo de datos.
- En el desarrollo de la aplicación web se creó y se administró la base de datos con SQL Server Management Studio de una forma fácil, el cual permitió modificar las tablas y sus atributos de la base de datos sin perder la información almacenada.
- Se utilizó Entity Framework con *Code First* para crear automáticamente las entidades del modelo y la clase de contexto a partir de la base de datos previamente creada, pero se tuvieron que agregar anotaciones de datos y métodos para realizar las funciones CRUD (*Create, Read, Update and Delete*) en cada una de las entidades del modelo.
- Se agregaron métodos en las entidades del modelo para realizar las funciones CRUD (*Create, Read, Update and Delete*), con la finalidad de reutilizar el código durante el desarrollo de la aplicación web y facilitar las consultas a la base de datos.
- Se desarrolló una sola vista para modificar o para agregar nueva información mediante formularios, evitando tener que desarrollar dos vistas independientes con semejantes características, las cuales demandan de mayor tiempo en el momento de realizar cambios.

- Se utilizó la sintaxis de Razor en las vistas para agregar la información a las tablas y los formularios HTML, con el objetivo que se pueda visualizar la información de una forma apropiada y ordenada.
- Las vistas cuentan con dos bloques en los cuales se agregaron referencias a las hojas de estilos y las referencias a librerías JavaScript para poder incorporar funcionalidades adicionales únicamente en las vistas que requieren, de esta forma se puede utilizar las tablas dinámicas o un editor de textos.
- Se utilizaron métodos asincrónicos para el registro de usuarios, con la finalidad de evitar que la aplicación web se congele mientras se realiza el registro y se envía el correo de notificación.
- El uso de la metodología de desarrollo de software Kanban permitió planear, estructurar y controlar el proceso del desarrollo de la aplicación web en cada una de las etapas para garantizar el cumplimiento del producto.
- Las pruebas de integración permitieron verificar que los componentes de la aplicación funcionan correctamente actuando en conjunto.
- Las pruebas de aceptación de la aplicación web verificaron que se cumplió con los requerimientos levantados.
- Se utilizó el gestor de paquetes Nuget para agregar automáticamente Entity Framework al proyecto utilizado para generar el modelo de la base de datos, ItextSharp para generar los informes en formato PDF y EPPlus para generar los informes en formato Excel.
- Se utilizó Bootstrap en las Vistas de la aplicación web para generar un diseño adaptable, de tal forma que el contenido de la aplicación web se ajuste a cualquier dispositivo con cualquier tamaño de pantalla para que se visualice correctamente.
- ASP.NET MVC permite habilitar fácilmente las conexiones seguras mediante SSL para el acceso a la aplicación web.
- Se utilizó ASP.NET Identity para contar con un sistema de autenticación y autorización seguro en la aplicación web, el cual se integra fácilmente al proyecto y cuenta con funcionalidades para registrar usuarios, cambiar de roles, acceder a la información del usuario, entre otros.

- LinQ y las operaciones Lambda simplificaron las consultas de la información en la base de datos a través de las entidades del modelo.
- Se agregó el editor de texto TinyMCE en algunas vistas para establecer el formato de los encabezados y de los pies de página de los informes en formato PDF.
- Se crearon palabras clave para especificar los formatos de los informes; estas palabras serán reemplazadas por su correspondiente información al momento de generar el informe.
- Se implementó un hilo en *background* para el envío de correos electrónicos con recordatorios a los profesores que están registrados en la aplicación web.
- Las vistas parciales se utilizan para presentar menús con accesos directos a las categorías y sus formularios en los que se tienen que ingresar documentación.
- Los formularios utilizados para el ingreso y la modificación de información en la aplicación web cuentan con el método denominado `AntiForgeryToken`, que se utiliza para evitar falsificaciones generando un campo oculto en el formulario que se valida cuando se envía el formulario.
- La aplicación web permite crear categorías que a su vez crean carpetas para organizar la documentación ingresada de los profesores.
- Los archivos de la documentación ingresada de los profesores se guardan con un nombre único de tal forma que se pueda identificar fácilmente.
- La aplicación web permite crear formularios con campos de diferentes tipos de forma dinámica, de tal forma que se puedan agregar o eliminar los campos a conveniencia.
- La aplicación web permite crear portafolios de las asignaturas que imparten los profesores de forma personalizable, permitiendo agregar grupos de campos para que se ingrese la documentación requerida.
- Se utilizó una plantilla de un *dashboard* para conseguir una combinación apropiada de estética y funcionalidad al momento en que se produce la interacción entre el usuario y la aplicación web.

- Se corrigió el código de la librería *Data Tables* para que se pueda ordenar apropiadamente la columna que muestra las direcciones IP del lugar de donde se produjeron los eventos (*logs*).
- Se utilizó un *Bundle* para reducir el número de solicitudes al servidor y reducir el tamaño de los archivos correspondientes a las hojas de estilos y librerías JavaScript, con la finalidad de mejorar el tiempo de carga de las vistas de la aplicación web.
- La aplicación web puede utilizarse con otras finalidades en donde se tenga que recolectar documentación o información.

4.2 RECOMENDACIONES

- Utilizar la aplicación web del presente Trabajo de Titulación para automatizar el proceso de recolección de los documentos de los profesores del DETRI para el proceso de acreditación de carreras.
- Mejorar la apariencia de las vistas de la aplicación web, incorporando mayor interactividad en cada una de ellas.
- Agregar la propiedad *PopOver* al HTML de los campos de los formularios y de los portafolios, para que se muestre la información de ayuda referente al respectivo campo.
- Desarrollar un módulo para obtener reportes estadísticos de la información ingresada en lo referente a los proyectos de investigación que realizan los profesores u otras actividades.
- Desarrollar un módulo en la aplicación web que facilite y automatice el proceso para generar y restaurar respaldos de la documentación de los profesores que se almacenan en la aplicación web.
- Utilizar una cuenta de correo electrónico asignada por la Escuela Politécnica Nacional que cuente una mayor capacidad para el envío de correos electrónicos.
- Publicar la aplicación web en Internet, de tal forma que los profesores puedan acceder e ingresar la documentación solicitada desde cualquier lugar y en cualquier momento.

- Utilizar la metodología Kanban en el desarrollo de nuevos proyectos de software, porque es muy sencilla utilizar y consiste en ir etiquetando con tarjetas cada una de las tareas que se deben llevar a cabo.
- Utilizar ASP.NET Identity para contar con un sistema de autenticación y autorización seguro en las aplicaciones web, el cual se integra fácilmente al proyecto.
- Utilizar una plantilla HTML gratuita cuando no se cuenta con un diseñador gráfico que se encargue de la apariencia de una aplicación web.
- Utilizar el editor de textos TinyMCE por tratarse de un editor muy completo que cuenta con una gran cantidad de funcionalidades y por ser muy personalizable durante su integración al proyecto.
- Utilizar JQuery porque simplifica la forma de interactuar con los documentos HTML y la manipulación del DOM, permite desarrollar contenido dinámico de una forma más sencilla que utilizando JavaScript.
- Utilizar Bootstrap en los proyectos web porque permite organizar y presentar de una forma fácil el contenido que se visualizará de forma responsiva.
- Utilizar métodos asincrónicos en las tareas que requieran mayor tiempo de procesamiento, de tal forma que no se congele la aplicación web mientras se realiza alguna acción.
- Utilizar hilos en segundo plano que se encarguen de realizar tareas específicas como es el caso del envío automático de mensajes de correo electrónico de recordatorios, de tal forma que no se congele la aplicación web, ni interfiera con el trabajo de los usuarios.
- Desarrollar aplicaciones web en capas para facilitar el mantenimiento y el desarrollo de versiones posteriores.
- Utilizar una sola vista en proyectos de ASP.NET MVC para agregar o actualizar la información porque facilita y ahorra tiempo en el momento de realizar cambios en la vista.
- Crear en ASP.NET MVC un proyecto para la capa de datos en donde se encontrarán las entidades del modelo, de tal forma que se puedan desarrollar fácilmente otras aplicaciones tanto móviles como de escritorio utilizando la información de la misma base de datos.

- Agregar en ASP.NET MVC métodos con las funciones CRUD (*Create, Read, Update and Delete*) en las entidades del modelo porque simplifica las consultas a la base de datos y permite reutilizar el código.
- Acceder a la aplicación web mediante los navegadores Firefox y Chrome porque son los navegadores con los que se llevaron a cabo las pruebas de integración y de validación sin presentar inconvenientes.
- Utilizar un *Bundle* para reducir el número de solicitudes al servidor y reducir el tamaño de los archivos correspondientes a las hojas de estilos y librerías JavaScript, con la finalidad de mejorar el tiempo de carga de las vistas de la aplicación web.

REFERENCIAS BIBLIOGRÁFICAS

[1] CEAACES, *Reglamento de Evaluación Acreditación y Categorización de Carreras de las Instituciones de Educación Superior,* Quito, 2014, pp. 11-16.

[2] G. Coulouris, J. Dollimore, T. Kindberg y G. Blair, Distributed Systems. Concepts and Design, Quinta ed., Boston, Massachusetts: Pearson Education, 2012, p. 1.

[3] C. De la Torre, U. Zorrilla, M. Á. Ramos y J. Calvarro, Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0, España: Krasis Consulting, S. L., 2010, pp. 33-34.

[4] R. Vargas, “Di Mare,” 05 02 2007. [En línea]. Available: http://www.di-mare.com/adolfo/cursos/2007-2/pp-3capas.pdf. [Último acceso: 02 04 2017].

[5] J. García, “Parvulos,” 01 09 2014. [En línea]. Available: http://joseluisgarciab.blogspot.com/2014/09/programacion-en-3-capas.html. [Último acceso: 05 04 2017].

[6] D. Nations, “Lifewire,” Lifewire, 17 10 2016. [En línea]. Available: https://www.lifewire.com/what-is-a-web-application-3486637. [Último acceso: 01 03 2017].

[7] N. Pizarro, “Ideas Digitales Aplicadas SpA.,” Estrategia Digital, 26 9 2016. [En línea]. Available: https://www.ida.cl/blog/estrategia-digital/diferencias-aplicacion-web-sitio-web/. [Último acceso: 01 03 2017].

[8] E. Tholome, “1&1 Digital Guide,” Desarrollo web, 28 6 2016. [En línea]. Available: https://www.1and1.es/digitalguide/paginas-web/desarrollo-web/que-es-una-web-app-y-que-clases-hay/. [Último acceso: 1 3 2017].

[9] S. Smith, “Microsoft Docs,” Microsoft , 14 10 2016. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview. [Último acceso: 01 02 2017].

[10] S. Chauhan, “DotNetTricks,” 26 10 2016. [En línea]. Available: http://www.dotnettricks.com/learn/mvc/aspnet-mvc-request-life-cycle. [Último acceso: 10 3 2017].

[11] F. Giardina, “Maestros del web,” 14 12 2010. [En línea]. Available: http://www.maestrosdelweb.com/tutoria-desarrolloweb-asp-net/. [Último acceso: 1 3 2017].

[12] MSDN, “Developer Network,” MSDN Microsoft, 20 10 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/ms171868(v=vs.110).aspx. [Último acceso: 25 3 2017].

[13] S. Shirhatti, “Learn Microsoft,” Microsoft, 22 11 2016. [En línea]. Available: https://www.iis.net/learn/get-started/whats-new-in-iis-10. [Último acceso: 25 3 2017].

[14] J. Galloway, B. Wilson, K. Scott Allen y D. Matson, Professional ASP.NET MVC 5, Canada: John Wiley & Sons, Inc., 2014.

[15] Microsoft, “Developer Network,” Microsoft MSDN, 12 2 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb399567(v=vs.110).aspx. [Último acceso: 10 4 2017].

[16] Microsoft, “Developer Network,” Microsoft MSDN, 5 10 2012. [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb896269(v=vs.100).aspx. [Último acceso: 5 4 2017].

[17] Microsoft, “Developer Network,” 5 10 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.componentmodel.dataannotations(v=vs.110).aspx. [Último acceso: 5 4 2017].

[18] W. Montes, ASP.NET MVC 6 - Una guía introductoria, Costa Rica: Walter Montes Delgado, 2015, pp. 27-30.

[19] W. W. W. C. (W3C), “W3C,” Draft, 6 3 2017. [En línea]. Available: http://w3c.github.io/html/#html-vs-xhtml. [Último acceso: 20 3 2017].

[20] Mozilla Developer Network y colaboradores individuales, “Mozilla Fundation,” Mozilla Developer Fundation, 5 1 2017. [En línea]. Available: https://developer.mozilla.org/es/docs/HTML/HTML5. [Último acceso: 20 3 2017].

[21] M. Bowman, “QA En dispositivos moviles,” 23 5 2013. [En línea]. Available: https://qaendispositivosmoviles.wordpress.com/2013/05/04/que-es-y-para-que-sirve-jquery-y-html5/. [Último acceso: 10 3 2017].

[22] Mozilla, “Mozilla Developer Network,” Mozilla Fundation, 6 3 2017. [En línea]. Available: https://developer.mozilla.org/es/docs/HTML/HTML5/HTML5_lista_elementos. [Último acceso: 3 4 2017].

[23] J. Z., “HTML5 New World,” 07 04 2014. [En línea]. Available: https://html5newworld.wordpress.com/2014/04/07/que-es-css3/. [Último acceso: 01 4 2017].

[24] W. W. W. C. (W3C), “W3C,” W3C Working Group Note, 31 1 2017. [En línea]. Available: https://www.w3.org/TR/CSS/#css. [Último acceso: 1 4 2017].

[25] T. FitzMacken, A. Pasic y T. Dykstra, “Microsoft Docs,” Microsoft , 2 7 2014. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c. [Último acceso: 20 3 2017].

[26] U. d. Alicante, “Universidad de Alicante,” 20 8 2012. [En línea]. Available: https://si.ua.es/es/documentacion/asp-net-mvc-3/3-dia/helpers-basico.html. [Último acceso: 10 4 2017].

[27] R. Murphey, “The jQuery Foundation,” 26 1 2015. [En línea]. Available: http://learn.jquery.com/. [Último acceso: 5 3 2017].

[28] P. Le Hégaret, L. Wood y J. Robie, “W3C Recomendation,” 04 03 2004. [En línea]. Available: https://www.w3.org/2005/03/DOM3Core-es/introduccion.html. [Último acceso: 10 06 2017].

[29] C. Krall, “Aprender a programar,” 21 04 2014. [En línea]. Available: http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=801:dom-o-document-object-model-javascript-ique-es-para-que-sirve-w3c-nodos-child-ejemplos-cu01123e&catid=78&Itemid=206. [Último acceso: 10 06 2017].

[30] María, “Punto Abierto,” 08 03 2016. [En línea]. Available: http://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas. [Último acceso: 01 05 2017].

[31] R. Mamani Ninaja, “ManINformatic,” 10 1 2014. [En línea]. Available: http://maninformatic.blogspot.com/2014/01/tut-aprender-asp-net-mvc-paso-paso-en-7.html. [Último acceso: 10 4 2017].

[32] D. García, “Let's code something up!,” 15 11 2013. [En línea]. Available: https://danielggarcia.wordpress.com/2013/11/15/el-controlador-en-asp-net-mvc-4-iii-action-filters/. [Último acceso: 8 4 2017].

[33] D. García, “Let's code something up!,” 13 11 2013. [En línea]. Available: https://danielggarcia.wordpress.com/2013/11/13/el-controlador-en-asp-net-mvc-4-ii-action-results/. [Último acceso: 8 4 2017].

[34] E. Tomas, “Desarrollo Web,” 18 5 2011. [En línea]. Available: https://desarrolloweb.com/articulos/tabla-rutas-l-dotnet.html. [Último acceso: 5 4 2017].

[35] R. Anderson, “Microsoft Docs,” 23 8 2012. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification. [Último acceso: 10 4 2017].

[36] M. Rouse, “Techtarget,” Search SQL Server, 1 2 2017. [En línea]. Available: http://searchsqlserver.techtarget.com/definition/SQL-Server. [Último acceso: 6 4 2017].

[37] I. Sommerville, Ingeniería de Software, Novena ed., Mexico: Pearson Education, 2011, pp. 28-50.

[38] R. Pressman, Ingeniería del software. Un enfoque práctico, Séptima ed., New York: McGraw-Hill, 2010.

[39] R. L. Granados La Paz, Desarrollo de aplicaciones web en el entorno servidor, Málaga: IC Editorial, 2014.

[40] L. Gilibets, “Comunidad IEBS,” Innovation & Entrepreneurship Business School, 31 07 2013. [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/. [Último acceso: 10 02 2018].

[41] S. Robson, Agile SAP: Introducing Flexibility, Transparency and Speed to SAP Implementations, United Kingdom: IT Governance Publishing, 2013.

[42] L. Moon, “Trello,” 01 01 2017. [En línea]. Available: https://trello.com. [Último acceso: 01 05 2017].

[43] S. Labs, “Kanban Tool,” Shore Labs, 01 01 2017. [En línea]. Available: http://kanbantool.com/es/tablero-kanban. [Último acceso: 01 05 2017].

[44] M. Cannon-Brookes y S. Farquhar, “Atlassian,” 01 01 2017. [En línea]. Available: https://es.atlassian.com/software/jira. [Último acceso: 01 05 2017].

[45] C. Hefley, “Virtual Kanban,” 18 09 2012. [En línea]. Available: http://virtualkanban.net/. [Último acceso: 01 05 2017].

[46] R. Burger, “Target Process,” 05 01 2017. [En línea]. Available: https://www.targetprocess.com/. [Último acceso: 01 05 2017].

[47] CEAACES, *Modelo genérico de evaluación del entorno de aprendizaje de carreras presenciales y semipresenciales de las universidades y escuelas politécnicas del Ecuador,* Quito, 2015.

ANEXOS

Los anexos se encuentran en un CD adjunto a este documento.

ANEXO A. DIAGRAMAS DE CASOS DE USO

ANEXO B. BACKLOG

ANEXO C. SCRIPT PARA GENERAR LA BASE DE DATOS

ANEXO D. CÓDIGO DE LA APLICACIÓN WEB

ANEXO E. MANUAL DEL USUARIO

ANEXO F. PRUEBAS DE VALIDACIÓN