ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN WEB BASADA EN MVC PARA LA GESTIÓN DE SOLICITUDES EN LA CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

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

VANESSA MARISOL CHENAZ PASPUEZÁN

DIRECTOR: FRANKLIN LEONEL SÁNCHEZ CATOTA, MSc.

Quito, febrero 2020

AVAL

Certifico que el presente trabajo fue desarrollado por Vanessa Marisol Chenaz Paspuezán, bajo mi supervisión.

FRANKLIN LEONEL SANCHEZ CATOTA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Vanessa Marisol Chenaz Paspuezán, 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 dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Vanessa Marisol Chenaz Paspuezán

DEDICATORIA

A mi esposo Geovanny por todo su apoyo durante todo este proceso de la carrera y a mi Emilia por ser mi mayor fuerza para seguir adelante.

A mis padres Eduardo y Mercedes por siempre motivarme para ser mejor cada día, por cada palabra de aliento y por siempre estar para mí.

AGRADECIMIENTO

A Dios por la fuerza y sabiduría que me brindó en la elaboración de esta tesis y durante toda mi carrera.

"… No puede el hombre recibir nada, si no le fuere dado del cielo. Juan 3:27".

A mis padres por ser mi motor, mis ganas de seguir adelante por ser siempre ese motivo para levantarme y seguir superándome

A mis hermanas Lorena, Milena, Diana y Paola por estar siempre pendientes de mí, gracias por cada visita, por cada llamada, por cada palabra de aliento, son muy importantes en mi vida.

A mi esposo por su paciencia y apoyo en estos años de carrera universitaria, gracias por secar cada una de mis lágrimas en mis malos momentos y por celebrar conmigo en mis buenos momentos, TE AMO.

Al MSc. Franklin Sánchez por su paciencia y apoyo durante la realización de este trabajo de titulación.

Al MSc. Pablo Hidalgo por todo su apoyo y sus consejos en momentos difíciles, gracias porque aparte de ser un excelente docente es una gran persona, gracias por todo.

A mis amigos por cada momento compartido.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

Los estudiantes de la carrera de Ingeniería en Electrónica y Redes de Información usualmente realizan solicitudes que son entregadas personalmente en la secretaría de la coordinación, su trámite es manual y la respuesta al estudiante es realizada a través de frecuentes visitas del estudiante a la secretaría para conocer el estado de la solicitud.

El presente Trabajo de Titulación se enfoca en la creación de una aplicación web para automatizar la gestión del proceso de solicitudes, que le permite al estudiante realizar las solicitudes en línea y mediante un correo electrónico informarle al estudiante el estado de su solicitud. Además, el estudiante puede revisar si la respuesta a su solicitud incluye documentación adjunta.

La aplicación web se la realizó con Microsoft MVC, SQL Server y utilizando la metodología Scrum y cuando la aplicación estuvo terminada se realizó la publicación en la plataforma en la nube Amazon Web Services.

Los resultados de la aplicación fueron validados con pruebas unitarias, pruebas de integración y pruebas de validación con la ayuda de un *framework* de pruebas y los actores de la aplicación.

El presente documento está estructurado de la siguiente forma. En el capítulo 1 se listan las tecnologías utilizadas para el desarrollo de la aplicación web. En el capítulo 2 se especifican los requerimientos, el diseño de la base de datos y la implementación de la aplicación utilizando la metodología Scrum. El capítulo 3 muestra los resultados de las pruebas realizadas con la aplicación. El capítulo 4 contiene las conclusiones y recomendaciones sobre la aplicación desarrollada.

PALABRAS CLAVE: MVC, Aplicación Web, Gestión de Solicitudes.

ABSTRACT

Students of the Electronics and Information Networks Engineering degree usually make requests that are personally delivered to the secretariat, their processing is manual and the student's response is made through frequent visits by the student to the secretariat to know the status of application.

This Degree Work focuses on the creation of a web application to automate the management of the application process, which allows the student to make applications online and by email to inform the student of the status of their application. In addition, the student can check if the response to his request includes attached documentation.

The web application was made with Microsoft MVC, SQL Server and using the Scrum methodology and when the application was finished the publication on the Amazon Web Services cloud platform was made.

The results of the application were validated with unit tests, integration tests and validation tests with the help of a test framework and the application actors.

This document is structured as follows. Chapter 1 lists the technologies used for the development of the web application. Chapter 2 specifies the requirements, the design of the database and the implementation of the application using the Scrum methodology. Chapter 3 shows the results of the tests performed with the application. Chapter 4 contains the conclusions and recommendations on the application developed.

KEYWORDS: MVC, Web Application, Request Management.

1. INTRODUCCIÓN

Durante el proceso de solicitudes como por ejemplo extensión de créditos, anulación de matrícula, etc., los estudiantes deben acercarse a la Secretaría de la carrera de Ingeniería en Electrónica y Redes de Información con el fin de entregar la solicitud de manera física y a la espera de una respuesta que debe ser consultada constantemente, causando molestia en la Secretaría y a los mismos estudiantes al tener que trasladarse a la Universidad en especial cuando éstos están fuera de la ciudad de Quito. El proceso de creación, procesamiento y respuesta a las solicitudes no está automatizado, por este motivo se plantea desarrollar una aplicación web basada en MVC que permitirá generar solicitudes con base en plantillas predefinidas, así como también permitirá al estudiante escribir el contenido de su solicitud con la ayuda de la aplicación web autocompletando información predefinida; también permitirá que el personal de Secretaría pueda notificar cuándo pueden acercarse a retirar la solicitud de ser ese el caso; y, además permitirá conocer el estado de las solicitudes por parte de la Coordinación.

1.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es desarrollar un prototipo de aplicación web basada en MVC para la gestión de solicitudes en la carrera de Ingeniería en Electrónica y Redes de Información.

Los objetivos específicos del Proyecto Técnico son:

- Diseñar cada uno de los módulos que permitirán disponer de una aplicación web basada en MVC que servirá para la gestión de solicitudes.
- Implementar los distintos módulos que forman parte de la solución.
- Analizar los resultados de las pruebas realizadas a la aplicación web.

1.2 ALCANCE

Este trabajo de titulación plantea desarrollar una aplicación web basada en MVC haciendo uso de la metodología Scrum. La aplicación permitirá generar solicitudes con base en plantillas prediseñadas, o que el estudiante redacte sus propias solicitudes si este fuere el caso, autocompletando información predefinida, y, una vez que la solicitud haya sido tramitada, la respuesta será enviada al estudiante a través de un correo electrónico mediante la aplicación web, igualmente se le permitirá a la Coordinación conocer el estado actual de las solicitudes.

En caso de que el estudiante requiera que el documento tenga una firma, tendrá que imprimir la respuesta y acercarse con la respuesta a la Secretaría para proceder a firmar el documento; los documentos que van a ser generados dispondrán de códigos de barra que permitirán validar la autenticidad del documento generado.

El estudiante accederá a la aplicación web desarrollada en ASP.Net MVC a través de un explorador web para generar sus solicitudes, la aplicación web contará con una base de datos donde se almacenará la información. La arquitectura de la aplicación web se muestra en la Figura 1.1.

Figura 1.1 Arquitectura de la aplicación web

Las plantillas prediseñadas que se plantean para la aplicación web serán: Extensión de Créditos, Culminación de Plan de Estudios, Depuración de Currículum, Certificado de Programas de Estudios y otras que se determinarán con base al análisis de requerimientos.

La aplicación web contará con los siguientes módulos:

Módulo Gestor: este módulo permitirá generar solicitudes considerando dos aspectos, el uno con plantillas prediseñadas las cuales se establecerán haciendo la parte del análisis correspondiente en las Secretarías y el otro permitirá al estudiante redactar la información deseada pero la cual tendrá ciertos campos que serán precargados por ejemplo el nombre de la autoridad a la que la solicitud debe ir dirigida, el cargo de esta, etc. De esta manera el estudiante solo llenará los campos necesarios que se pide como requisito, o podrá crear su propia solicitud de ser el caso.

Módulo de Control: en este módulo se podrán establecer los parámetros que contendrá la solicitud, como por ejemplo el nombre de la autoridad a la que va dirigida la solicitud, cargo etc.

Módulo Canal de Comunicación: permitirá darle trámite a la solicitud y la aplicación web permitirá enviarle la respuesta al estudiante por medio de un correo electrónico.

Módulo de Estado: permitirá conocer estadísticas sobre las solicitudes que han sido recibidas durante un periodo académico.

1.3 MARCO TEÓRICO

A continuación, se revisan los fundamentos teóricos y las tecnologías que son considerados para el desarrollo de la aplicación web. Se realizan descripciones breves sobre la metodología de desarrollo, tecnologías web, bases de datos y tipos de pruebas para verificar y validar el sistema.

1.3.1 APLICACIONES WEB

Una aplicación web es cualquier programa de computadora que realiza una función específica mediante el uso de un navegador web como su cliente [1].

Para desarrollar una aplicación se utilizan *scripts* del lado del servidor (ASP, PHP, etc.) que se encargan de almacenar y recuperar información y *scripts* del lado del cliente (HTML, JavaScript, etc.) que se encargan de la presentación de la información que se indica al usuario.

Una de las características importantes de las aplicaciones web, es que éstas permiten que múltiples usuarios puedan hacer uso de la misma versión, la mayoría son compatibles con cualquier navegador. Existen varios tipos de aplicaciones web, pero las más importantes son las aplicaciones: dinámicas y estáticas que se muestran en la Figura 1.2.

1.3.1.1 Aplicaciones Web Estáticas

Las aplicaciones web estáticas tienen sus páginas generadas por un servidor y ofrecen poca o ninguna interactividad [2].

Pueden ser utilizadas cuando no sea necesaria la interacción del usuario con la aplicación, sino que solo pueda observar información desplegada en la aplicación, la cual será la misma para todo aquel que ingrese al sitio web. La información en estas aplicaciones solo puede cambiar por parte de un administrador ya que los cambios se harán directamente al HTML.

1.3.1.2 Aplicaciones Web Dinámicas

Muchos sitios web dinámicos basados en CMS[1] facilitan la actualización de la información a través de una interfaz fácil de usar [3]

En una aplicación web dinámica el servidor es el encargado de procesar los archivos antes de enviarlos al cliente, generalmente este tipo de aplicaciones se conectan a una base de datos para la gestión de información.

El contenido de estas páginas es específico para un usuario y se irá actualizando de manera dinámica de acuerdo con las preferencias del usuario.

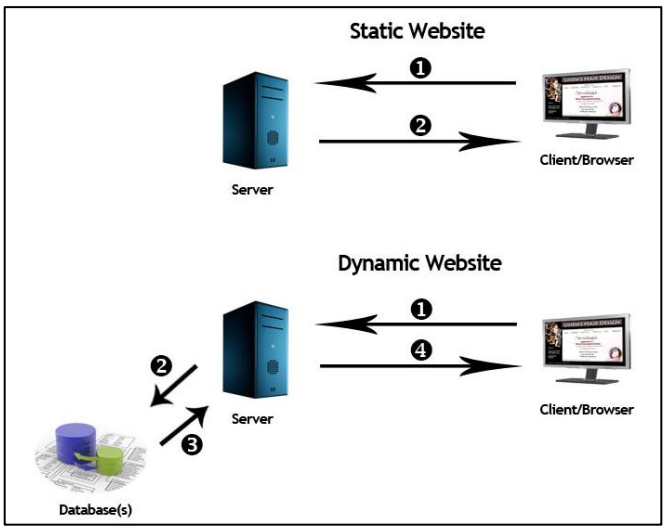


Figura 1.2 Aplicación web estática vs Aplicación web dinámica [4]

1.3.2 ESTÁNDARES WEB

Los estándares web son un conjunto de reglas/normas que marcan los requisitos que se deben cumplir en el diseño, desarrollo y puesta en producción de un sitio web para que sea compatible por los diferentes elementos que lo utilicen [5].

Usar estándares web ayuda a una mejor compatibilidad entre el código de la página web y el navegador de visualización.

Los estándares web son los siguientes:

[1] CMS (*Content Management Systems*): son programas que hacen uso de lenguajes del lado del servidor para que el trabajo de diseño web sea más fácil.

1.3.2.1 Markup Language

Un lenguaje de marcado es un lenguaje de computadora que usa etiquetas para definir elementos dentro de un documento [6]. Los dos lenguajes más populares son:

- **HTML (HyperText Markup Language).** - es utilizado para la creación de páginas web. Estas páginas pueden ser vistas por cualquier persona que se ha conectado a Internet. HTML no es un lenguaje de programación sino un lenguaje de marcado ya que el contenido de cada página web está determinado por etiquetas HTML.

  Las etiquetas de páginas básicas, como <head>, <body> y definen secciones de la página, mientras que las etiquetas como <table>, <form>, <image> y <a> definen elementos dentro de la página [6].

  La mayoría de los elementos requieren una etiqueta de inicio que consiste en el nombre del elemento en este caso *head* envuelto en paréntesis angulares de apertura y cierre así <head> y también una etiqueta de fin, que se caracteriza por ser la misma que la de inicio, pero con una barra diagonal antes del nombre del elemento así </head>; existen algunas etiquetas que no necesitan de un cierre, entre las más comunes está el salto de línea
.

  La última versión de HTML la cual salió en 2014 es HTML5, que incluye el soporte para audio y video, se incrustan haciendo uso de las etiquetas <audio></audio> y <video></video>, también incluye soporte para fórmulas matemáticas y científicas.

- **XML (Extensible Markup Language). –** XML sirve para representar información estructurada en la web, de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por diversos tipos de aplicaciones y dispositivos [7].

  En XML las etiquetas no poseen un significado definido ya que hace uso de etiquetas personalizadas, pero su delimitación si debe ser por etiquetas de inicio y de cierre como en HTML. XML separa la parte del contenido de la de presentación, como se muestra en el Código 1.1.

```
<h1>Valeria<h1>                  →HTML

<Nombre>Valeria<Nombre>     → XML
```

Código 1.1 Etiquetas en formato HTML y XML

Como se puede observar la etiqueta <h1> de HTML muestra el texto “Valeria” pero no especifica nada del significado, XML indica que se trata de un Nombre puesto que XML se preocupa de lo que la etiqueta puede significar mas no del aspecto con el que se va a mostrar en la página web.

XML es sensible a mayúsculas y minúsculas utilizadas en las etiquetas, por lo que no va a ser los mismo la etiqueta <Nombre> que <nombre>.

1.3.2.2 CSS (cascading style sheets)

Es un lenguaje de hojas de estilo utilizado para describir la presentación de un documento escrito en HTML o XML [8]. CSS se usa para el diseño de páginas web de un excelente aspecto, es decir, para definir el tamaño de letra, color, alineación, división en varias columnas, animaciones y muchas más características a los elementos HTML que ayuden a mostrar de una mejor manera la página web al usuario.

CSS puede controlar cómo se verán los elementos HTML, en el navegador presentando el diseño que desee, basado en reglas que determinan el o los grupos de estilos que deben aplicarse a elementos específicos o grupos de elementos en su página web, también se utiliza en lenguajes de marcado como SGV[2] o XML.

Para la sintaxis, se escoge un elemento HTML como se ve en el Código 1.2, en este ejemplo el encabezado de nivel 1 y se le asignará un color y un tamaño de acuerdo con lo que se desee mostrar:

```
13  h1 {
14      color: black;
15      font-size: 10em;
16  }
```

Código 1.2 Ejemplo del uso de CSS en un encabezado

Se le puede cambiar tantas propiedades y sus respectivos valores a los elementos HTML como se desee.

2 SVG (*Scalable Vector Graphics*): lenguaje de etiquetas dirigido a la representación de gráficos vectoriales.

1.3.2.3 Bootstrap

Un problema que presentan las páginas realizadas con el diseño básico de HTML es que la página puede tener aspectos diferentes en cada navegador o dispositivo en el que se quiera ingresar a la página, esto conlleva a que la modificación se deba hacer para cada navegador o dispositivo, por ello nace Bootstrap.

Bootstrap es un *framework* que hace uso de HTML, CSS y JavaScript para el diseño web [9].

Bootstrap agiliza el desarrollo de las interfaces de usuario en las aplicaciones web, entregando una navegabilidad adaptable que será compatible con cualquier tipo de dispositivo ya sea este una computadora, una *Tablet* o un *smartphone*, la representación de la página siempre será óptima.

1.3.2.4 Javascript

JavaScript es un lenguaje de secuencias de comandos que le permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes entre otras cosas [10].

En una página web: HTML define el contenido, CSS especifica el diseño y JavaScript programa su modo de comportamiento. JavaScript construye su funcionalidad de lado del cliente y no del lado del servidor, por lo que será interpretado por el navegador.

Se puede definir una secuencia de comandos en línea, donde el contenido de la escritura es parte del documento HTML como se muestra en Código 1.3 . También puede definir un script externo, donde el JavaScript esté contenido en un archivo separado y se hace referencia a través de una URL [11] cómo se indica en Código 1.4.

```
8.  <body>
9.  <script>
10. let d = new Date();
11. document.body.innerHTML = "<h1>Time right now is:  " + d.getHours() +
    ":" + d.getMinutes() + ":" + d.getSeconds()
12. "</h1>"
13. </script>
14. </body>
```

Código 1.3 Ejemplo de JavaScript directamente en un archivo HTML [12]

```
8.  <body>
9.  </body>
10. <script src="js/myscript.js"></script>
```

Código 1.4 Ejemplo de JavaScript desde un archivo separado [12]

JavaScript no necesita declarar cuál es el tipo de datos de los parámetros cuando se define la función. En JavaScript no está disponible el polimorfismo, así que, si tiene dos funciones con el mismo nombre, pero con diferentes parámetros, la segunda definición va a sustituir a la primera.

- **JQuery**

Es una librería de JavaScript, creada con el objetivo de facilitar el uso de JavaScript en los sitios web a los desarrolladores [13].

jQuery brinda muchas funciones como calendarios, formularios dinámicos, *clicks* a botones, mostrar u ocultar ciertos elementos HTML, peticiones AJAX[3] entre otras para que una página web pueda ser interactiva en varios navegadores.

Para implementar jQuery en la solución se debe agregar las librerías que son gratuitas y sus respectivas referencias en todas las páginas donde se quiera implementar estas sentencias jQuery.

1.3.3 ASP.NET MVC

Es un *framework*[4] de Microsoft creado para el desarrollo de aplicaciones web que ejecuta el patrón MVC (Modelo- Vista- Controlador).

El principal propósito de MVC es aislar la lógica de negocio de la interfaz de usuario para mejorar la capacidad de mantenimiento, pruebas y una estructura de la aplicación más limpia.

MVC separa los datos de una aplicación, la interfaz de usuario, y la lógica en tres componentes distintos.

[3] Ajax (*Asynchronous JavaScript and XML*) es una tecnología que carga contenido adicional en la ventana actual comunicándose con el servidor web sin actualizar la página HTML.

[4] *Framework*: Proporciona una base sobre la cual los desarrolladores de software pueden crear programas para una plataforma específica

En la Figura 1.3, el usuario interactúa con la interfaz realizando una petición denomina solicitud HTTP, el Controlador recibe la acción solicitada por el usuario y se encarga de consultar al Modelo dicha acción, una vez que el procesamiento de datos entre la base de datos y el Modelo se ha completado, el Controlador crea una respuesta al usuario mediante el envío de una Vista que produce código HTML para ser desplegada en el navegador.

Las tareas realizadas por el Controlador y el Modelo se realizan del lado del servidor, y la representación de la Vista que se le presenta al usuario se realiza del lado del cliente, porque es el navegador el encargado de interpretar la información que se va a mostrar en la pantalla del usuario.

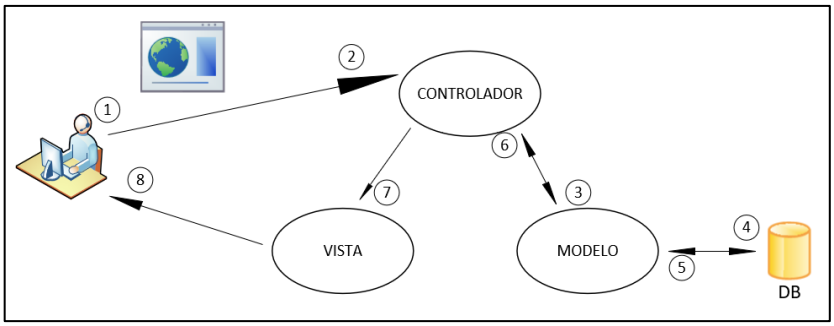


Figura 1.3 Forma en la que trabaja MVC

Los componentes de la arquitectura MVC son: el Modelo, la Vista y el Controlador, los cuales se detallan a continuación.

1.3.3.1 Modelo

El Modelo es la representación de los objetos, procesos y reglas del mundo real que definen el tema, conocido como el dominio, de la aplicación [14].

Existen validaciones de datos en el lado del cliente, éstas se colocan en el Modelo para validar la entrada del usuario y en caso de no ser cumplidas como cuando se ingresan valores erróneos o nulos, por ejemplo, desplegar mensajes al usuario para brindar una retroalimentación que ayude a que su interacción con la aplicación sea la correcta.

Se puede agregar fácilmente validación a la aplicación agregando *Data Annotations* a las clases de Modelos. *Data Annotations* permiten describir las reglas que se quiere aplicar a las propiedades del Modelo, y ASP.NET MVC se encargará de aplicarlas y mostrar los mensajes apropiados a los usuarios [15].

Los atributos de *Data Annotations* incluidos en el espacio de nombres *System.ComponentModel.DataAnnotations* más utilizados están descritos en la Tabla 1.1.

Tabla 1.1 Data Annotations más usados.

Data Annotation	Función
Required	Indica que la propiedad es un campo obligatorio, debe llenarse y no se puede omitir.
DisplayName	Define el texto que se quiere usar en la Vista.
StringLength	Define una longitud máxima para un campo de cadena (String).
MinLength	Indica la longitud mínima que deberá tener una propiedad.
MaxLength	Indica la longitud máxima que deberá tener una propiedad
Range	Define un valor máximo y mínimo para valores numéricos.
RegularExpression	Se puede establecer un patrón de *expresiones* regulares para la propiedad.

Un ejemplo de la utilización de *Data Annotations* se puede observar en el Código 1.5, éstos se colocan encima de los atributos de la clase, para la validación de los datos del Modelo.

DisplayName indica el texto que se va a mostrar para el campo indicado (líneas 30, 36, 40), *Required* especifica que es un campo requerido es decir no se puede dejar en blanco (líneas 31, 37, 41), con *StringLength* se limita el número de caracteres máximo 10 caracteres para cédula y teléfono (líneas 35, 42) y se usa *DateType.EmailAddress* para comprobar el correcto formato del correo electrónico ingresado (línea 32).

```
30          [DisplayName("Correo Electronico")]
31          [Required(ErrorMessage = "El Correo Electronico es requerido")]
32          [DataType(DataType.EmailAddress, ErrorMessage = "El correo no es valido")]
33          public string CorreoElectronico { get; set; }
34
35          [StringLength(10)]
36          [DisplayName("Telefono")]
37          [Required(ErrorMessage = "El telefono es requerido")]
38          public string Telefono { get; set; }
39
40          [DisplayName("Cedula")]
41          [Required(ErrorMessage = "La cedula es requerida")]
42          [StringLength(10)]
43          public string Cedula { get; set; }
44
```

Código 1.5 Uso de *Data Annotations* en atributos de una clase

1.3.3.2 Vista

Una Vista es una representación (visual) de su Modelo. Normalmente destacaría ciertos atributos del Modelo y suprimiría otros, por lo tanto, actúa como un filtro de presentación [16].

La Vista es la encargada de recibir los datos que vienen del Modelo y mostrarlos al usuario, es decir todo lo que respecta a la interfaz de usuario (UI), donde mostrará datos que se encuentren en el Modelo y la manera cómo podrá interactuar el usuario con la aplicación.

Como la Vista tiene una relación directa con el Modelo, ésta debe conocer la semántica de los atributos que el Modelo representa y no necesariamente todos los atributos del Modelo van a ser mostrados en la Vista, ya que esto depende de lo que el programador desee que el cliente pueda realizar, si en el Modelo existen mensajes para validación estos serán desplegados en la Vista.

Se puede utilizar Razor[5] para incrustar código en el marco HTML, cualquier lógica que exista en las Vistas por más mínima que sea, debe estar relacionada con la presentación del contenido.

En la Figura 1.4 se observa que por convención existe una carpeta llamada Vistas en la cual existen carpetas separadas para cada Controlador, éstas se crean con el mismo nombre del Controlador, pero se omite la palabra Controller y cada una de las acciones que tiene la clase Controlador pueden ser representadas por distintas Vistas.

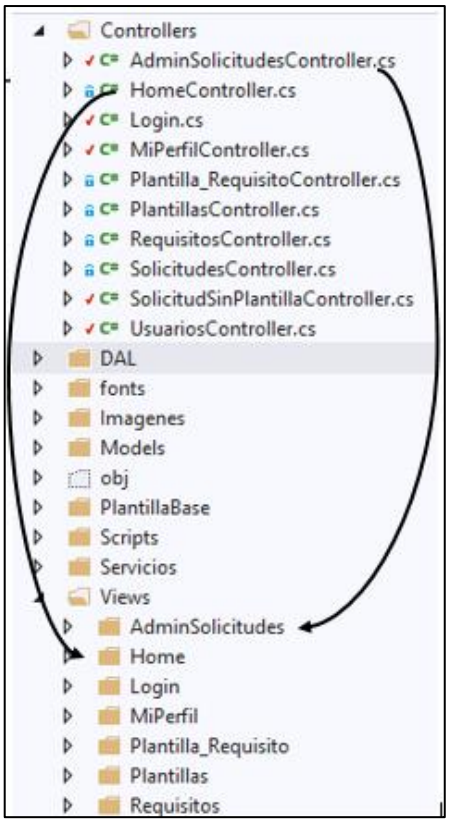


Figura 1.4 Vistas y Controladores en la solución

Una de las herramientas que pueden ser utilizadas en las Vistas es la siguiente:

[5] Razor es una sintaxis de marcado la cual permite introducir código C# o Visual Basic basado en servidor en páginas web.

1.3.3.2.1 *Razor*

No es un lenguaje de programación, es un lenguaje de marcado del lado del servidor, permite incorporar código basado en el servidor (Visual Basic y C#) en páginas web [17].

Se basa en conocimiento de HTML y C#, e incluye la sintaxis simple para añadir variables y bloques de código a su página [18].

Permite escribir una combinación de HTML y código C# o VB (Visual Basic) del lado del servidor, en este trabajo se utiliza la sintaxis C# que tiene la extensión de archivo .cshtml, entre las características más importantes de Razor se tiene:

- Permite minimizar el número de caracteres para escribir un código.
- Es fácil de aprender por que se usa el lenguaje familiar C#.
- Es *Intellisense*, que significa que admite la finalización de sentencias en Visual Studio es decir autocompletar el código para un desarrollo más ágil.

Elementos más utilizados en Razor:

- **Variables**: Para incluir una variable en una Vista se debe añadir un prefijo a la variable usando el símbolo @, que se utiliza para acceder al objeto del lado del servidor en sintaxis Razor como se muestra en el Código 1.6.

```
<span> @DateTime.Now </span>
```

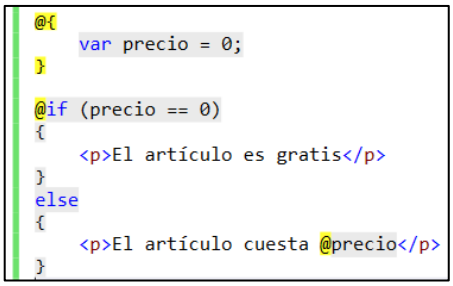
Código 1.6 Sintaxis Razor para una variable

- **Bloque de código:** se puede escribir muchas líneas de código del lado del servidor entre llaves @{ … } y cada línea debe terminar con ";" igual que en C#, como se muestra en el Código 1.7.

```
@{
    var mensaje = "Hola Mundo";
    var fecha = DateTime.Now.ToShortDateString();

}

<h2>@mensaje </h2>
<h3>Hoy es: @fecha</h3>
```

Código 1.7 Sintaxis Razor para un bloque de código

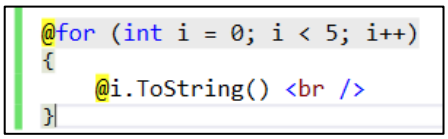
- **Condición if-else:** Para las condiciones if-else se debe iniciar con el símbolo @, el bloque de código debe estar entre llaves {} incluso si solo tiene una única declaración, como se muestra en el Código 1.8.



```
@{
    var precio = 0;
}

@if (precio == 0)
{
    <p>El artículo es gratis</p>
}
else
{
    <p>El artículo cuesta @precio</p>
}
```

Código 1.8 Sintaxis Razor para condición *if-else*

- **Bucle**: su código se escribe igual que con las condiciones if-else, como se muestra en el Código 1.9.



```
@for (int i = 0; i < 5; i++)
{
    @i.ToString() <br />
}
```

Código 1.9 Sintaxis Razor para bucle *for*

- **Modelo**: se debe usar @model para usar cualquier Modelo de objeto en la Vista y se llama a los atributos con la sintaxis @Model seguido del atributo que se quiere desplegar.

  En la Figura 1.5, se puede observar un ejemplo de cómo usar @model y así llamar al Modelo Usuario de la solución (línea 1).

- **HTML HELPERS:** El MVC Framework viene con un número de HTML Helpers que trabajan con las Data Annotations definidos en el Modelo para reducir significativamente la cantidad de código que la solución requiere [18].

  HtmlHelper asocia un objeto del Modelo a elementos HTML, para presentar el valor de las propiedades del Modelo en los elementos HTML que se han generado es decir traer información del Modelo, y entrega el valor de los elementos HTML a las propiedades del Modelo cuando envía un formulario web, es decir cuando se envía información al Modelo.

  Existe una gran variedad de métodos de extensión en la clase HtmlHelper, para crear diferentes controles HTML. En la Tabla 1.2, se detallan las más usados.

Tabla 1.2 Tipos de HTML Helpers [19]

HtmlHelper	Control HTML
Html.ActionLink	Genera un hipervínculo que se asocia a una acción en el Controlador.
Html.TextBox	Crea un cuadro de texto.
Html.TextArea	Cuadro de texto más grande que el textbox usado para una gran cantidad de información.
Html.DropDownList	Genera una lista de elementos que pueden ser seleccionados.
Html.Hidden	Sirve para ocultar propiedades del Modelo que no se quieren mostrar al usuario.
Html.Password	Cuadro de texto de contraseña, generalmente usa el símbolo (*) para no mostrar el texto que se ingresa.
Html.Display	Despliega texto HTML.
Html.Label	Crea una etiqueta estática.
Html.Editor	Similar a HTML.TextBox pero este se crea dependiendo del tipo de datos que se maneje.

En la Figura 1.5, se muestra un ejemplo de cómo se utilizan los HTML Helpers. En esta Vista el usuario podrá ver la información con respecto al Modelo de Usuario, el HTML Helper retorna el nombre del atributo que se encuentra en el Modelo en este caso Nombre (línea 14), se despliega el nombre del Usuario que se encuentra almacenado en la base de datos según haya sido seleccionado (línea 18).

Figura 1.5 Ejemplo de uso de HTML Helpers

1.3.3.3 Controlador

Los Controladores son la parte central de una aplicación ASPNET MVC, el Controlador es el primero en interactuar con la solicitud HTTP que recibe por parte del usuario y decide qué Modelo se debe usar, toma los datos del Modelo y los pasa a la respectiva Vista.

Los Controladores son los encargados de controlar el flujo de la aplicación ya que se encargan de tomar una entrada y generar una salida apropiada.

Una clase de Controlador se hereda de la clase *System.Web.Mvc.Controller*, por convención, existe una capeta llamada Controladores, en la cual se van a colocar todos los Controladores requeridos para el proyecto.

El Controlador cumple las siguientes funciones:

- Es el encargado de elegir qué va a mostrar al usuario
- Es el intermediario entre la Vista y el Modelo
- Procesa los datos antes de enviarlos a presentar en la Vista.

Los Controladores tienen métodos para procesar las solicitudes realizadas por el usuario, a estos métodos se los denomina métodos de acción ya que se pueden llamar desde la web usando alguna URL para realizar una acción.

El procesamiento de una solicitud comienza con un elemento llamado el motor de enrutamiento, que mapea la petición a un Controlador y un método. Después de eso, se ejecuta el método y produce un objeto resultado de la acción que se devuelve al usuario. El tipo más común de objeto resultante de la acción es una Vista que representa HTML en el navegador [20].

La ruta *"{controller} / {action} / {id}"* identifica como primer parámetro, el Controlador que se va a llamar, la solicitud HTTP no incluye la palabra “Controller”, el segundo parámetro define el método de acción que se ejecutará en la clase del Controlador y el tercer parámetro indica información que será utilizada por el método como se muestra en la Tabla 1.3, aunque no todas las peticiones cuentan con este tercer parámetro ya que todo depende del método al que se quiere acceder.

Tabla 1.3 Ejemplo de URLs para llamar a los métodos del Controlador

Solicitud	Valores de los parámetros.	Ejecución
http://domain/Estudiante/Index/	Controlador = Estudiante Acción = Index	El método Index en la clase EstudianteController
http://domain/Estudiante/Edit/1	Controlador = Estudiante Acción = Edit Id = 1	El método de acción Edit en la clase EstudianteController pasa el valor 1 como el parámetro Id del método de acción.

Un ActionResult es un tipo de retorno de un método de acción del Controlador, este puede retornar un Modelo a Vista, archivo, redireccionar hacia otra ruta etc.

Tipos de retorno del método de acción:

- ViewResult: retorna una Vista para representar HTML en el navegador.
- PartialViewResult: retorna una Vista parcial.
- FileResult: retorna contenido binario (por ejemplo, para descargar un archivo).
- JsonResult: retorna un objeto en formato JSON (JavaScript Object Notation).
- RedirectResult: realiza una redirección HTTP a otra URL.
- RedirectToRouteResult: realiza una redirección HTTP, pero a una ruta específica en lugar de una URL

Para el envío de información desde el Controlador hacia la Vista se utilizan los siguientes contenedores:

- **ViewBag:** es útil cuando se desea transferir datos temporales que no están incluidos en el Modelo desde el Controlador a la Vista [21].

  ViewBag es una propiedad de tipo dinámico es decir no requiere de una conversión para tipos de datos complejos.

  En el Código 1.10 se puede ver cómo se asigna la cadena “Vanessa Marisol” a la propiedad Nombre en el Controlador y se puede acceder a esta propiedad en la Vista con la notación @ViewBag.Nombre.

```
//Action Method
public ActionResult Index()
{
    ViewBag.Nombre = "Vanessa Marisol";
    return View();
}


@*View*@
<h3>@ViewBag.Nombre</h3>
```

Código 1.10 Ejemplo del uso de ViewBag

ViewBag solamente transfiere datos del Controlador a la Vista no viceversa. Si se produce la redirección estos valores serán nulos.

- **ViewData** es simplemente un diccionario que utiliza el patrón clave / valor, la información en el diccionario *ViewData* se añade simplemente dándole un nombre y estableciendo su valor [20].

  Tiene la misma función que *ViewBag*, transferir datos desde el Controlador a la Vista, pero no viceversa, requiere conversión de texto cuando se esté utilizando tipos de datos complejos y de una verificación de valores nulos para evitar errores.

  En el Código 1.11 se utiliza la clave “Fecha” y se le asigna el valor de la fecha actual en formato corto en el Controlador y es llamada igualmente con notación Razor desde la Vista.

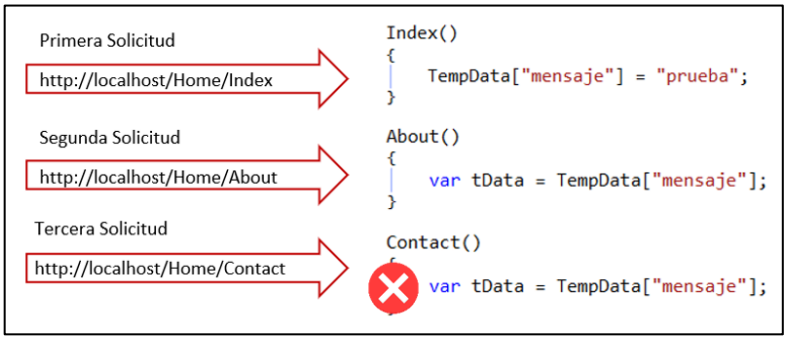
```
//Action Method
public ActionResult Index()
{
    ViewData["Fecha"] = DateTime.Now.ToShortDateString();
    return View();
}


@*View*@
<h2>@ViewData["Fecha"]</h2>
```

Código 1.11 Ejemplo del uso de ViewData

ViewBag utiliza *ViewData* como un mecanismo de almacenamiento, por lo que de cualquier forma terminan usando un diccionario en consecuencia, la clave de *ViewData* y la propiedad de *ViewBag* no deben ser las mismas.

- **TempData** se almacena en la sesión actual en lugar de un diccionario, la información se descarta rápidamente una vez que la redirección se ha completado [20].

TempData almacena datos entre dos solicitudes consecutivas, si se quiere mantener los valores para una tercera solicitud se deberá usar *TempData.Keep().* Si actualiza la página, la información de *TempData* no estará disponible como muestra el Código 1.12.



Código 1.12 Ejemplo del uso de TempData

1.3.3.4 Arquitectura en Capas

La arquitectura en capas es la Vista conceptual de la estructura de la arquitectura de una aplicación, toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos [22].

La arquitectura de una aplicación depende de cómo esté elaborado su código y de esta manera dividirlo en componentes lógicos. Cada capa tiene una respectiva funcionalidad. Cuando se diseña una aplicación se suele dividir en 3 o más capas para tener bien claro los elementos que van a ser parte de la aplicación.

Una arquitectura en capas generalmente consta de 3 capas como muestra en Figura 1.6. Estas capas que son: Capas de presentación, Capa de negocio y Capa de acceso a datos.

- **Capa de Presentación:** En esta capa está incluida la lógica de presentación y la interfaz de usuario, esta capa es la encargada de la interacción con el usuario.
- **Capa de Negocio:** BLL (*Bussiness Logic Layer*) está formada por las entidades, que representan objetos que van a ser manejados o utilizados por toda la aplicación [22].

  En esta capa se agrega lógica sin que el código de acceso a datos sea afectado.
- **Capa de Acceso a Datos:** DAL (*Data Access Layer*) se tienen clases que se comunican con la base de datos, para realizar operaciones como leer, editar, crear,

eliminar registros, todo esto se ejecuta de manera transparente para la capa de negocio.

Como están involucradas las tres capas en la aplicación: En la capa DAL se crea una clase *repository* que se encargará de ejecutar las consultas a la base de datos. En la capa de negocio o BLL se crea una *service* clase que hace referencia a la DAL y llama a los métodos para obtener los objetos necesarios. En la interfaz de usuario el Controlador llama al servicio y envía el resultado a la Vista, y ésta se encarga de representar el resultado que ha obtenido.

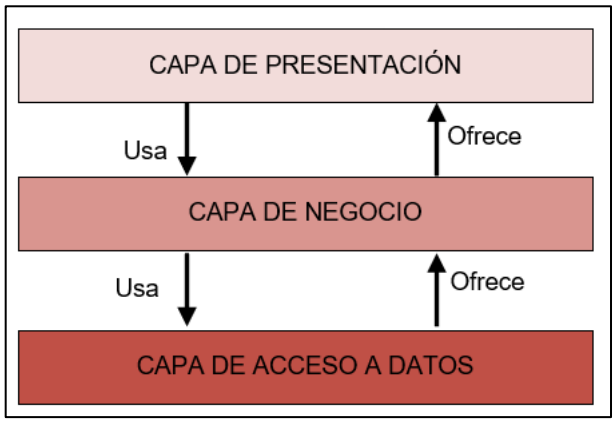


Figura 1.6 Arquitectura de 3 capas [23]

1.3.4 PRUEBAS

1.3.4.1 Pruebas Unitarias

Una prueba unitaria casi siempre se escribe utilizando un *framework* de prueba unitaria. Se puede escribir fácilmente y se ejecuta rápidamente. Es totalmente automatizado, confiable, legible y mantenible [20].

Las pruebas unitarias deben probar componentes de software individuales, es decir si se tiene un método que devuelve dos funcionalidades, se debería hacer una prueba para probar una funcionalidad que encuentre un objeto y otra para cuando no encuentre el objeto y devuelva un valor nulo.

Se pueden escribir tantas pruebas unitarias como se desee, ya que se ejecutan de forma automática y son muy rápidas. Es importante que después de algún cambio en el código, se vuelvan a ejecutar las pruebas unitarias para comprobar que el código sigue funcionando de manera correcta.

Las pruebas unitarias bien hechas, permiten identificar problemas de una manera muy rápida y fácil y de esa manera darnos cuenta donde está el error y solucionarlo de manera eficiente.

Las pruebas unitarias proporcionan simulacros o versiones falsas de dependencias (como una base de datos) de modo que la prueba unitaria no se base en ningún código externo y cualquier falla se pueda identificar con exactitud [24], de esta manera no afecta a la base de datos real, por ejemplo.

1.3.4.2 Pruebas de Integración

Son las encargadas de probar cómo trabaja un conjunto de software, al contrario que las pruebas unitarias, estas pruebas si utilizan una base de datos real y conexiones de red reales probando de esa manera si los componentes funcionan de manera correcta juntos como se requiere [25].

Es importante dedicar mayor esfuerzo en realizar las pruebas unitarias y luego realizar las pruebas de integración escogiendo algunas de ellas, y así detectar posibles errores.

En vista que las pruebas de integración son más grandes que las pruebas unitarias, son más lentas de realizar, por eso es recomendable limitar cuantas pruebas de integración tendrá la aplicación.

1.3.5 BASE DE DATOS RELACIONAL

Las bases de datos están estructuradas para facilitar el almacenamiento, recuperación, modificación y eliminación de datos junto con varias operaciones de procesamiento de datos [26].

La entidad es la representación de un objeto del mundo real, su descripción está dada por un grupo de columnas que representan los atributos, los cuales están almacenados como un registro de la tabla llamada tupla o fila, que representa un objeto único de datos.

Una tabla tiene: un nombre que representa la entidad que es un objeto real y único, las columnas representan los atributos que son características de la entidad y las filas representan los registros que contienen la información acerca de una entidad.

Cada tabla contiene una clave primaria única, la relación entre tablas se realiza a través de las claves foráneas que son campos que se asocian a las claves primarias de otras tablas, para ello existen 3 formas de relacionar tablas:

Uno a uno: el registro de la tabla está relacionado a un solo registro de otra tabla.

Uno a muchos: el registro de la tabla puede estar relacionado a uno o a muchos registros de otra tabla.

Muchos a muchos: varios registros de la tabla se relación con varios registros de otra tabla.

En el ejemplo que se muestra en la Figura 1.7, la tabla Carrera tiene en su primera columna un identificador único llamado IDCarrera conocido como clave primaria, en la segunda columna se encuentra el IDFacultad conocido como clave foránea, es la que hace referencia a la clave primaria de la tabla Facultad, esta clave es la encargada de relacionar las dos tablas, así la tabla Carrera puede acceder a los atributos que tenga la tabla Facultad.

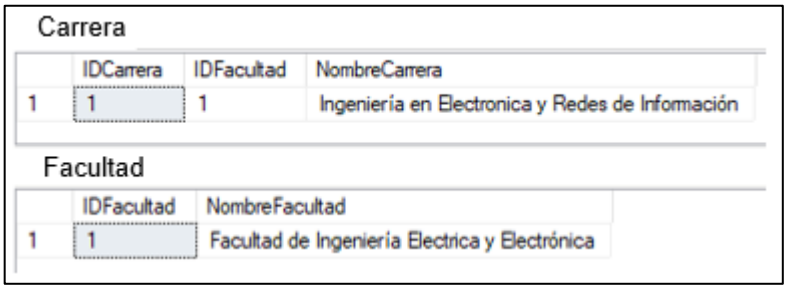


Figura 1.7 Ejemplo de modelo relacional

Los diagramas relacionales muestran de una manera esquematizada el diseño de una base de datos relacional. En la Figura 1.8, se muestra el diagrama relacional para las tablas Carrera y Facultad en las que se observan los atributos y la relación entre ellas, para este ejemplo la relación es de muchos a uno.

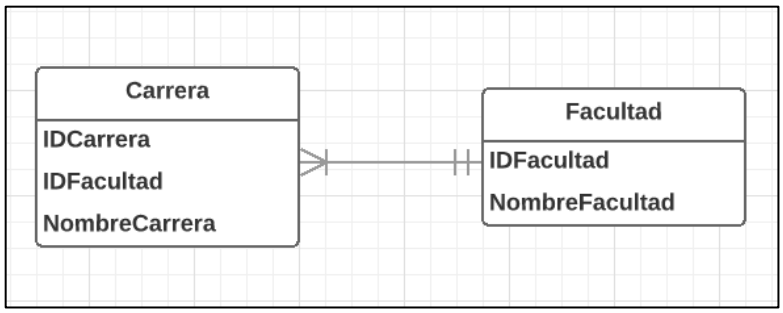


Figura 1.8 Ejemplo de diagrama relacional

Una base de datos relacional representa datos en forma de tablas, la API[6] de una base de datos relacional es SQL[7].

SQL es un lenguaje de programación estándar e interactivo para la obtención de información desde una base de datos y para actualizarla [27].

Las consultas SQL utilizan comandos que permiten seleccionar, insertar, eliminar, actualizar, alterar, truncar, conocer la ubicación de datos, es decir todo lo que se necesite para la gestión de datos.

1.3.6 ENTITY FRAMEWORK

Entity Framework es un mapeador de objetos relacionales (O/RM) que permite a los desarrolladores de .NET trabajar con una base de datos utilizando objetos .NET [28].

Entity Framework proporciona código generado automáticamente con lo que se reduce la codificación que realiza el desarrollador y en consecuencia se reduce el tiempo de desarrollo.

Cuando se crea el Modelo de la base de datos, se crea de manera automática parte del código que se requiere para la conexión entre la base de datos real y la aplicación web todo esto gracias al IDE[8].

Para crear un Modelo de entidad, *Entity Framework* proporciona entre sus métodos *Code First* y *DataBase First* [29].

- *Code first*. - El desarrollador especifica su Modelo usando clases normales POCO9, estas son relacionadas haciendo referencia entre ellas en propiedades y si se desea por ejemplo especificar el nombre de los campos en la base de datos se debe decorar las propiedades de forma especial con las llamadas *"Data Annotations".*

  Se definen las clases a partir de código, *Entity Framework* se ocupará de generar la base de datos y lo que requiera para acoplar las clases en ellas. Se puede actualizar la base de datos si el Modelo cambia usando la herramienta "Migraciones".

[6] API (*Application Programming Interface*): conjunto de comandos, funciones y protocolos informáticos que permiten integrar y desarrollar el software de las aplicaciones.
[7] SQL (*Structured Query Language*): lenguaje de programación encargado de la gestion de datos almacenados en una base de datos.
[8] IDE (*Integrated Development Environment*): Consolida las herramientas básicas necesarias para escribir y probar software.
[9] POCO (*Plain Old C# Object*): clases normales de C#

- *Database first*. - Se debe tener una base de datos ya creada con la que se quiere empezar la aplicación, *Entity Framework* generará las clases de forma automática y las actualizará si es necesario en caso de alguna variación en la base de datos.

  Cuando se usa *Entity Framework* las tablas que se tiene en la base de datos se las vincula con las clases que se tiene en la aplicación, y las columnas de las tablas se las vincula con los atributos que tienen las clases.

Entre los beneficios que tiene el uso de *Entity Framework* está el no hacer uso de código SQL sino utilizar consultas LINQ para la gestión de información de la base de datos.

1.3.6.1 Language Integrated Query (Linq)

LINQ es un conjunto de herramientas de Microsoft que permite realizar todo tipo de consultas a distintas fuentes de datos [30].

Con una consulta lo que se busca es agregar o recuperar datos de una base de datos. Linq ofrece un modelo sólido para realizar el trabajo en varias fuentes y formatos que los datos posean.

Linq es usado por la simplicidad para comunicarse con la base de datos para que el desarrollador se centre en los datos y no en el lenguaje que se está utilizando para llegar a ellos.

1.3.7 SCRUM

Scrum es un *framework* para el manejo de proyectos que tienen como fin el desarrollo de productos complejos [31].

Este marco de trabajo es utilizado para el desarrollo o creación de productos en donde es necesario entregar un producto de forma iterativa e incremental. Los cambios de requerimientos por parte del cliente usualmente causan problemas en un proyecto, pero con Scrum los cambios en los requerimientos y las prioridades son aceptados y suceden al inicio de cada *Sprint*. Esta metodología es utilizada en proyectos complejos, que necesiten resultados de forma rápida y donde prevalecen los siguientes aspectos: la flexibilidad, la innovación, la productividad y la competitividad.

Scrum también define un equipo de personas de cinco a nueve miembros que trabaja de manera efectiva, tiene los objetivos claros, se organizan en función al trabajo que van a realizar, entregan con puntualidad las funcionalidades más importantes, reciben

retroalimentación de personas externas al proyecto. El equipo y la gerencia tiene una comunicación más honesta, transparentando el progreso y los riesgos. El proceso se muestra en la Figura 1.9 [31].

1.3.7.1 Equipo Scrum

En Scrum no existe un rol de gerente ya que estas responsabilidades se dividen en 3 roles imprescindibles *Product Owner*, *Scrum Master* y el equipo de desarrollo, para llevar a cabo el proyecto de manera satisfactoria.

Los *Scrum Masters*, equipos de desarrollo y *Product Owners* trabajan juntos alrededor de requisitos y tecnologías para entregar productos funcionando de manera incremental aplicando su experiencia [32].

- **StakeHolder:** es el cliente, es el encargado de definir los requerimientos, recibe el producto realizado en cada *sprint* y brinda un *feedback* respectivo.
- **Product Owner:** gestiona el producto, es el intermediario entre el *skateholder* y el equipo de desarrollo.

  El *Product Owner* es el encargado de la recolección de requerimientos, de brindarles la prioridad respectiva y crear el *Product Backlog*, también es el encargado de aceptar o rechazar el software al final de cada sprint.
- **Scrum Master**: gestiona el proceso, actúa como moderador entre el *Product Owner* y el equipo.

  El *Scrum Master* elimina los impedimentos que podrían surgir durante el proceso para que su equipo cumpla sus objetivos y también es el encargado de ofrecer un buen entorno de trabajo para el equipo.
- **Equipo de desarrollo:** se gestiona a si mismo comprometiéndose a entregar software con calidad de producción, es un equipo autogestionado, es decir todos los integrantes del equipo se encargan de realizar las estimaciones de los ítems y así irán creando el *Sprint Backlog*.

  Se recomienda que el equipo de desarrollo esté conformado por no más de 10 personas sin contar con el *Product Owner* y el *Scrum Master*.

1.3.7.2 Sprint

Sprint es el nombre que va a recibir cada uno de los ciclos o iteraciones que se va a tener dentro de dentro de un proyecto Scrum [33].

Un Sprint puede durar un mes o menos, durante este tiempo se debe completar el trabajo comprometido por el equipo durante la planificación del *sprint*. [34]

Un sprint puede tener las siguientes etapas:

- **Sprint Planning** (planificación de *sprint*), reunión donde se decidirá qué tareas hacer y cómo se las va a realizar, y se debe tener en claro los objetivos del *Sprint*.
- **Daily Meeting Scrum** (reuniones de *scrum* diarias) buscan actualizar el estado del proyecto, analizar soluciones y desafíos, y de esa manera indicar el progreso a los dueños de los productos.
- **Sprint Review** (revisión del Sprint) es aquella donde se va a aceptar o a denegar el sprint, también se revisa el proceso que se ha llevado a cabo para identificar las cosas que se pueden mejorar en el siguiente *sprint*.
- **Sprint Retrospective** (retrospectiva de *sprint*) sirve para analizar los problemas que ha tenido el equipo y cómo se podrían corregir; permite una discusión del *sprint* y a buscar alternativas para hacer un trabajo más eficiente.

1.3.7.3 Artefactos

Los artefactos de *Scrum* representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación [35].

- **Product Backlog**

Es una lista de ítems los cuales van a representar trabajo en proceso. Los ítems deben ser valorados para entender la relación costo beneficio de estos y así saber en qué posición de la lista van a ser ubicados. El *Product Owner* es el encargado de determinar el orden en el que serán desarrollados los ítems de este listado desde el más prioritario al menos prioritario [35] .

En vista que los requerimientos son emergentes, es decir, no se conocen todas las características que se quiere que contenga el producto, el *product backlog* puede ser editado cuando se requiera agreguen nuevos ítems, eliminar otros, o se tenga que dividir un ítem en ítems más pequeños.

- **Sprint Backlog**

El *Sprint Backlog* es un listado de tareas que proviene del desglose de las Historias de Usuario que conforman la Pila de Producto (*Product Backlog*) [36].

Conocido como tablero de tareas que se van a realizar a lo largo del *Sprint*. Este tablero indica las tareas que el equipo planificó y en qué estado se encuentran actualmente. El equipo de desarrollo selecciona los ítems que formarán parte del *sprint backlog* al momento del *sprint planning*, una vez finalizados estos ítems el equipo alcanzará la meta del *sprint*.

El equipo de desarrollo es el único que tiene acceso a modificar su *sprint backlog* a lo largo del *sprint*, durante el *Daily scrum*.

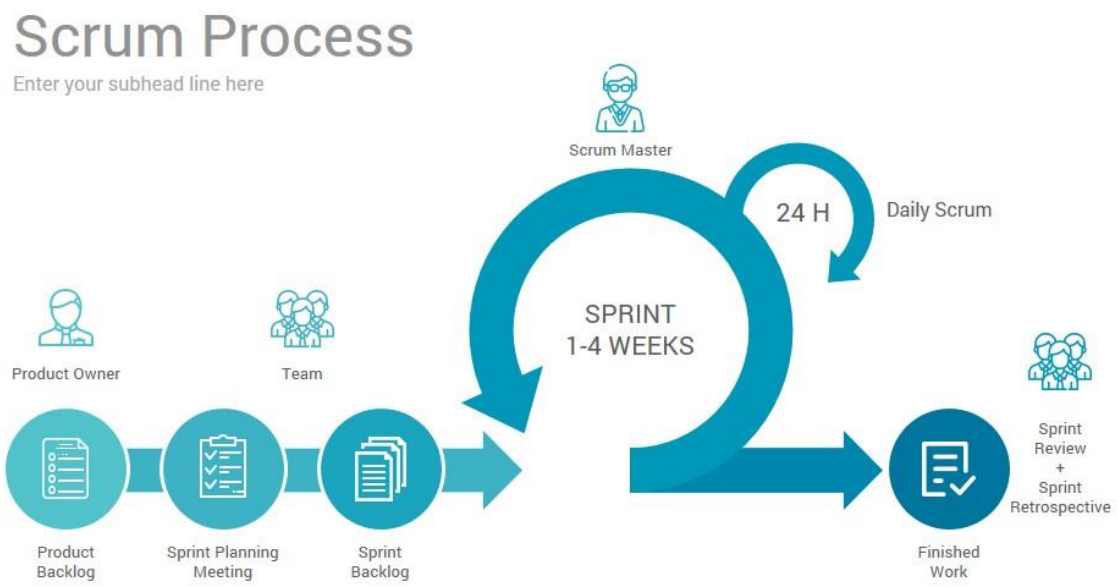


Figura 1.9 Metodología Scrum [37]

1.3.8 CORREO ELECTRÓNICO

El *framework* .NET incluye una biblioteca para enviar correos electrónicos utilizando el Simple Mail Transfer Protocol (SMTP). Esta librería se encuentra en el espacio de nombres *System.Net.Mail* e incluye las clases que permiten la creación de un mensaje de correo electrónico y el envío a un servidor SMTP. Para configurar esta funcionalidad se necesita acceder a un servicio SMTP, es posible acceder a cualquier servicio gratuito de correo como Gmail de Google, Outlook de Microsoft, etc. [38].

En la entrevista realizada al personal de secretaría se pudo determinar que el mejor medio de comunicación para dar respuesta de las solicitudes a los estudiantes podría ser el correo

electrónico, debido a que permite conocer el estado de la solicitud sin necesidad de acercarse personalmente a la secretaría, lo que permite dar un seguimiento del trámite a los interesados.

La Escuela Politécnica Nacional cuenta con el servicio de Outlook lo que facilita la implementación de la opción de correo electrónico para la automatización de la respuesta a las solicitudes generadas en la aplicación.

1.3.9 SERVIDOR WEB

Internet Information Services (IIS) es un servidor web de Microsoft que se ejecuta en sistemas Windows para publicar sitios web desarrollados en herramientas web como HTML, CSS, MVC y ASPX [39].

Este tipo de servidores aceptan solicitudes que vienen desde los clientes remotos y devuelven la respuesta apropiada, de esta manera comparten y entregan información a través de redes de área local (LAN) como intranets corporativas y redes de área amplia (WAN) como internet.

Microsoft proporciona una versión autónoma de IIS, llamada IIS Express, para que los desarrolladores prueben los sitios web. IIS Express ofrece todas las capacidades principales del servidor web IIS completo, pero permite realizar muchas tareas sin privilegios administrativos.

IIS es un complemento a Windows que se instala y habilita en el lado del computador que va a tener la responsabilidad de ser el servidor web, de la siguiente manera:

1.- Ir a Panel de control

2.- Seleccionar la opción Programas

3.- Seleccionar la opción Activar o desactivar las caracteristicas de Windows

4.- Seleccionar Internet Information Services

5.- Click en Aceptar

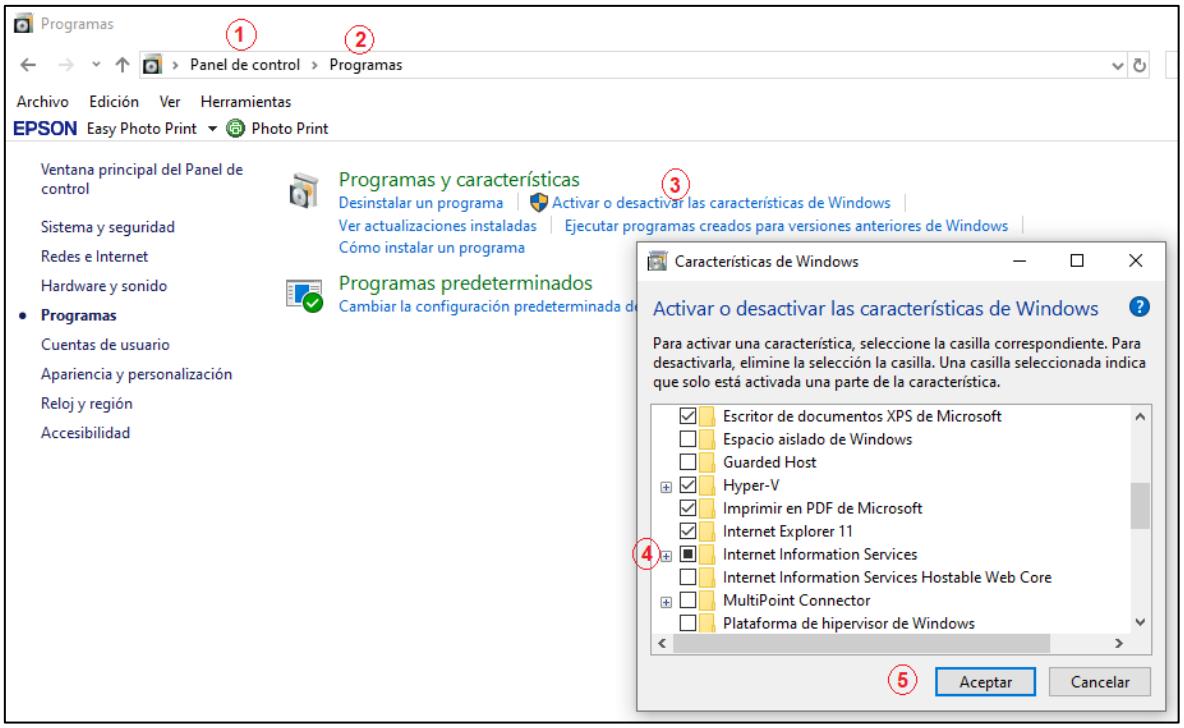


Figura 1.10 Habilitación del servidor

6.- Ingrese localhost en el navegador para comprobar el correcto funcionamiento

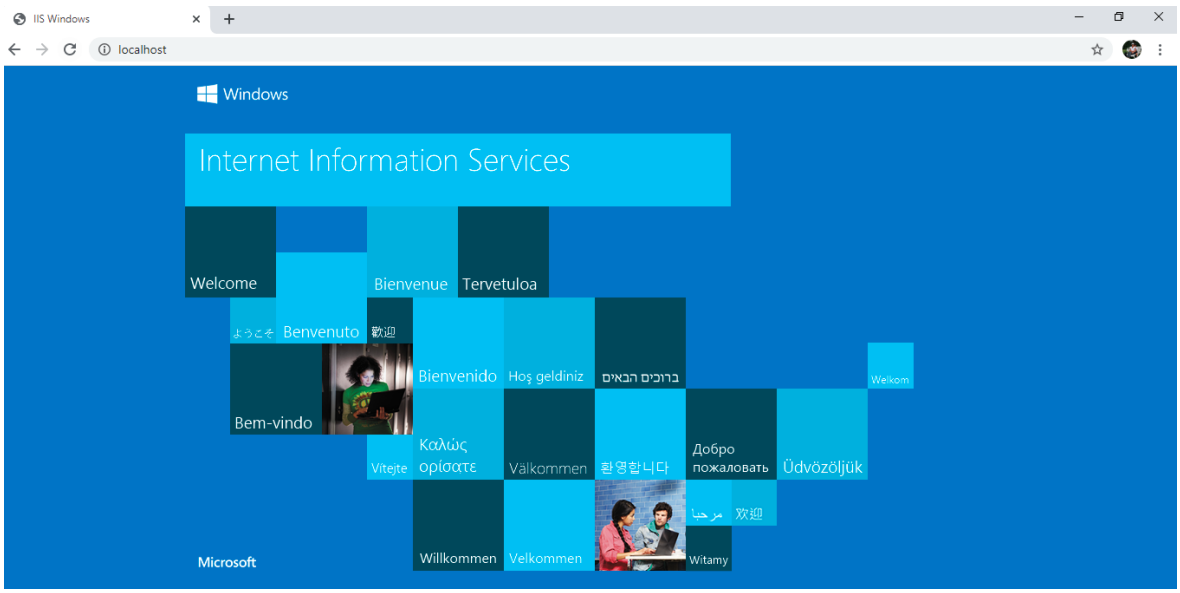


Figura 1.11 Comprobación del funcionamiento del IIS

1.3.10 REGLAMENTACIÓN DE LA ESCUELA POLITÉCNICA NACIONAL

El uso de la reglamentación de la Universidad sirve para limitar el tiempo de respuesta de la solicitud que haga el estudiante, en los artículos establecidos se especifica el tiempo máximo que se tendrá para dar una respuesta.

A continuación, se listan secciones del reglamento con respecto al tiempo de las solicitudes:

- La supresión del registro de matrículas al igual que las de retiro de una asignatura, curso o su equivalente tendrán un plazo máximo de 30 días calendario contabilizados desde el inicio de clases.
- Recalificación de notas, en el plazo de 2 días laborables de recibidas las copias del instrumento de evaluación, remitirán por separado a la autoridad de la unidad académica correspondiente, los resultados de la recalificación.
- Aprobación del Plan de Trabajo de Titulación, la aprobación o no del plan se le informará al estudiante en un tiempo máximo de 20 días calendario, en caso de que haya modificaciones se otorgará al estudiante un plazo de 10 días para presentar el documento definitivo.

Muchos de las solicitudes que los estudiantes requieren son Certificados que, con base a las respuestas del personal de secretaría en las entrevistas realizadas, tardan entre 1 y 3 días en ser entregados.

El reglamento completo de la Universidad se encuentra en el Anexo A.

2. METODOLOGÍA

En este capítulo se detallan los procesos para el levantamiento de la información, el diseño y la implementación de la aplicación web para la gestión de solicitudes de la Carrera de Ingeniería en Electrónica y Redes de Información.

La metodología utilizada para el desarrollo de la aplicación web fue *Scrum* en la cual se utilizaron las siguientes herramientas para la creación del *Product Backlog*: entrevistas, historias de usuario, diagramas de casos de uso, diagrama entidad-relación, diagrama relacional, diagramas de clases y *sketches*.

2.1 LEVANTAMIENTO DE INFORMACIÓN

2.1.1 ENTREVISTAS

Para la recopilación de requerimientos se realizaron entrevistas a los miembros de la coordinación de Redes, Telecomunicaciones, Subdecanato y Decanato y al personal de secretaría de las mismas carreras. Sin embargo, como la aplicación web está dirigida a la carrera de Ingeniería en Electrónica y Redes de Información, se consideraron más importantes las respuestas obtenidas de parte del Coordinador y personal de secretaría de esta carrera.

La entrevista tenía 8 preguntas en las que se buscó obtener información sobre:

1. Las solicitudes más frecuentes que realizan los estudiantes
2. El lugar en donde se encuentran las plantillas para dichas solicitudes si es que existieran.
3. Los datos más importantes que no deben faltar en una solicitud
4. Los requisitos que una solicitud necesita
5. El tiempo máximo para darle trámite a una solicitud
6. El procedimiento que se sigue para el respectivo trámite de la solicitud.

El detalle de las entrevistas se puede encontrar en el ANEXO B.

2.1.2 HISTORIAS DE USUARIO

Con la información obtenida de las entrevistas, se obtuvo la información para las historias de usuario para esta aplicación. Las historias de usuario detallan en resumen los requerimientos o necesidades del cliente sobre la aplicación a desarrollarse. Las historias de usuario se indican en la **¡Error! No se encuentra el origen de la referencia.**.

Tabla 2.1 Historias de usuario (Parte 1 de 2)

ID	Titulo	Descripción
HU01	Registro	El usuario necesita registrarse en la aplicación, ingresando datos como: nombres completos, apellidos completos, cédula, teléfono, correo electrónico, contraseña y seleccionando una carrera.
HU02	Autenticación	Para iniciar sesión el usuario necesita ingresar su número de cédula y su contraseña.
HU03	Datos personales	El usuario podrá gestionar su información personal como: nombres completos, apellidos completos, teléfono, correo electrónico, contraseña, carrera, pero la cédula no será modificable ya que es utilizada en la autenticación.
HU04	Gestión de usuarios	El usuario con el rol Administrador podrá crear, leer, actualizar usuarios.
HU05	Gestión de plantillas	El usuario con el rol de Secretario podrá realizar la gestión del CRUD[10] de plantillas que se mostrarán en la aplicación web.
HU06	Creación de solicitudes	El usuario con el rol Estudiante podrá crear solicitudes de dos tipos: • Con una plantilla definida sin necesidad de llenar ningún campo • Sin plantilla definida completando la información que desee, las solicitudes sin plantilla ya tendrán algunos datos precargados.
HU07	Historial de solicitudes	Los usuarios con roles de Administrador y Secretario podrán ver una lista de todas las solicitudes realizadas.
HU08	Verificación de información en solicitudes de Usuario	El usuario con el rol de Secretario podrá comprobar que la solicitud cumpla con los requisitos correspondientes como: Autoridad a la que va dirigida, Datos del estudiante como: Nombres y Apellidos, Cédula, Carrera, Correo electrónico, Teléfono, el pedido de la solicitud y verificar el/los requisitos que se deben adjuntar a la solicitud.
HU09	Solicitud – Administrador	El usuario con el rol Administrador podrá tramitar solicitudes para ser aprobadas o rechazadas.

[10] CRUD: Crear, Leer, Actualizar y Eliminar.

Tabla 2.1 Historias de usuario (Parte 2 de 2)

ID	Titulo	Descripción
HU10	Estado del trámite de las solicitudes	Cada vez que una solicitud sea creada se guardará la solicitud con un estado de *Enviado,* una vez que pase por el personal de secretaría su estado será *Aceptado* o *Rechazado* y finalmente cuando sea tramitada por la autoridad el estado será *Aprobado* o *Rechazado.* En cada cambio de estado se enviará un correo electrónico al estudiante.
HU11	Estadísticas de solicitudes	El Coordinador podrá observar todas las solicitudes que han sido gestionadas en forma de estadísticas para analizar la gestión que se les ha dado.

2.1.3 REQUERIMIENTOS

REQUERIMIENTOS FUNCIONALES

Los requerimientos se definieron en base a las historias de usuario, y son los siguientes:

1. Permitir creación, lectura y modificación de Usuarios.
2. Permitir el ingreso del usuario a la aplicación en base al rol establecido.
3. Permitir creación, lectura, modificación y eliminación de Plantillas.
4. Permitir creación, lectura, modificación y eliminación de Requisitos.
5. Permitir la asignación de Requisitos a las respectivas Solicitudes.
6. Permitir creación y lectura de solicitudes con una plantilla definida.
7. Permitir creación y lectura de solicitudes sin plantilla definida.
8. Permitir dar trámite a una solicitud.
9. Permitir notificaciones mediante correo electrónico en cada cambio de estado de la solicitud.
10. Permitir agregar archivos a la aplicación.
11. Permitir la visualización de estadísticas de las solicitudes tramitadas.

REQUERIMIENTOS NO FUNCIONALES

Se refieren a características generales para la implementación de la aplicación, estos son:

1. Desarrollo: La aplicación será desarrollada en Visual Studio 2017 como entorno de desarrollo y el Framework ASP.NET MVC con C# como lenguaje de programación para el servidor.
2. Rendimiento: La base de datos será de tipo transaccional por lo que se utilizará Microsoft SQL server.
3. Seguridad: La información de las contraseñas será cifrada.

2.1.4 MÓDULOS

La Tabla 2.2 indica las relaciones que tienen las historias de usuario con respecto a los módulos propuestos y también se añade el módulo usuario para la gestión necesaria de la aplicación.

Tabla 2.2 Historias de usuario correspondientes a los módulos

Módulo	Descripción	ID de la Historia de Usuario
Módulo Usuario	Gestión de la información de los usuarios y su acceso a la aplicación.	HU01 HU02 HU03 HU04
Módulo de Control	Establecimiento de parámetros de la solicitud	HU05
Módulo Gestor	Generación de Solicitudes	HU06
Módulo canal de comunicación	Trámite a las solicitudes y su respectiva notificación a través de correo electrónico	HU08 HU09 HU10
Módulo de Estado	Estadísticas sobre las solicitudes	HU11

2.2 DISEÑO DEL SISTEMA

2.2.1 DIAGRAMAS DE CASOS DE USO

El diagrama de casos de uso correspondiente al módulo gestor se muestra en la Figura 2.1, en este diagrama se observan las acciones que el actor *Estudiante* podrá realizar, las principales son la creación de solicitudes con plantilla y las solicitudes sin plantilla.

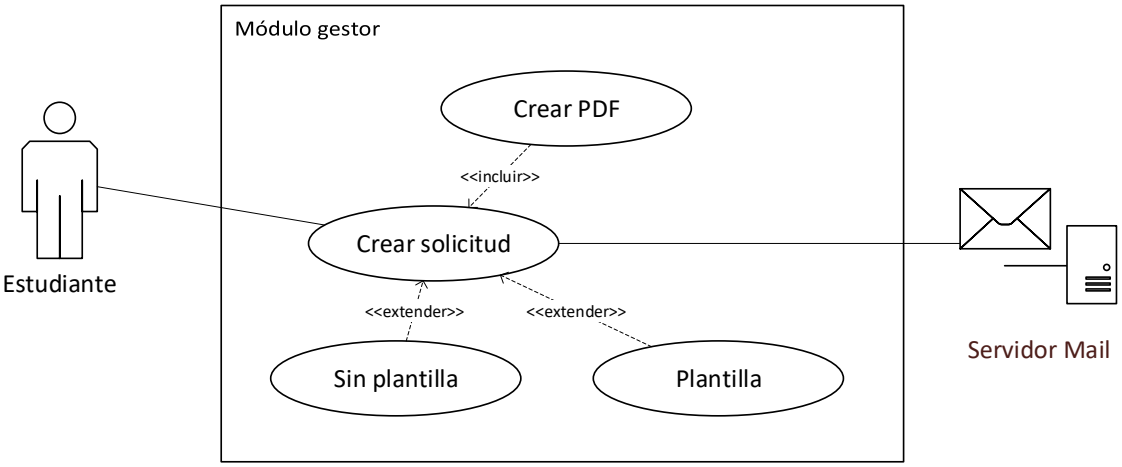


Figura 2.1 Casos de uso para el módulo gestor

El diagrama de casos de uso correspondiente al módulo de control se muestra en la Figura 2.2, tiene un solo actor debido a que es el único que tiene el acceso a las acciones correspondientes para gestionar las plantillas y los requisitos que van a tener los diferentes tipos de solicitudes.

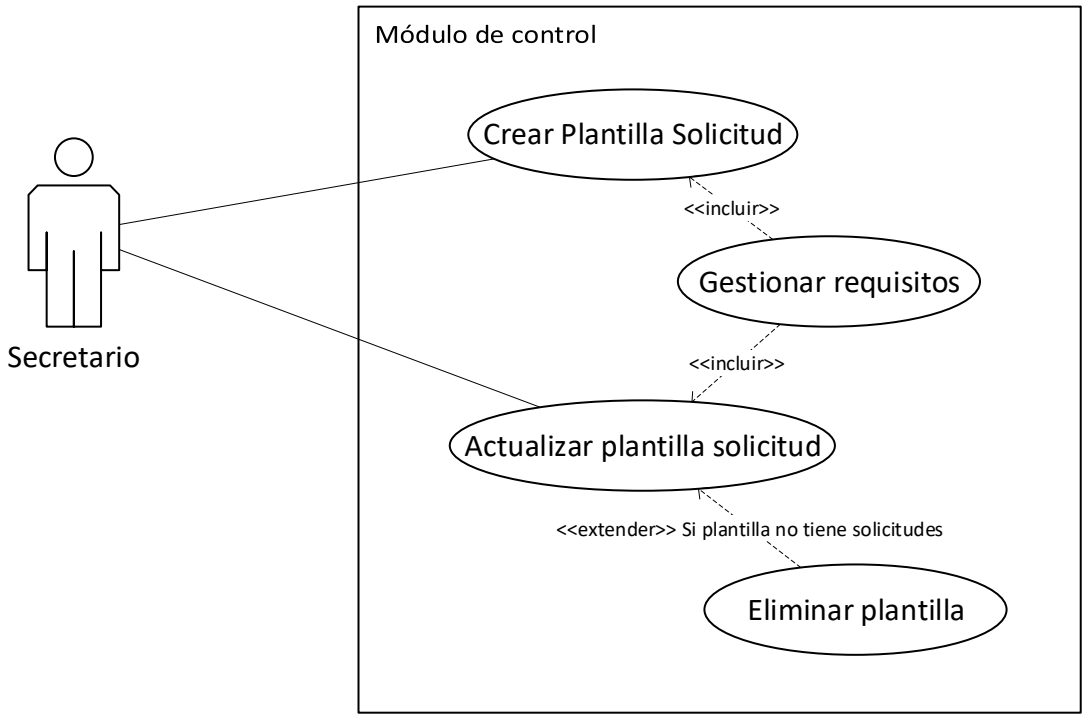


Figura 2.2 Casos de uso para el módulo de control

El diagrama de casos de uso correspondiente al módulo canal de comunicación se muestra en las siguientes 3 figuras:

La Figura 2.3 en donde el actor *Secretario* realiza la acción de verificación de la solicitud y sus requisitos, dependiendo de si estos se cumplen la solicitud es aceptada y enviada al actor *Autoridad* o si esta solicitud es rechazada se envía la respuesta directa al actor Estudiante, todas estas respuestas se las envía a través de correo electrónico (Servidor Mail).

La Figura 2.4, en donde el actor *Autoridad* realiza la acción de generar una respuesta a la solicitud para ser enviada al actor *Estudiante* y si es el caso en el que el Estudiante pidió algún documento, el actor *Autoridad* realiza la acción de agregar archivos de respuesta, en cada cambio de estado de estas acciones de respuesta se enviará un correo electrónico al actor *Estudiante* (Servidor Mail)

La Figura 2.5, en donde el actor *Estudiante* realiza la acción de revisar la respuesta enviada por el actor *Secretario* o *Autoridad*, y si ha sido generado algún documento de descarga, el actor *Estudiante* puede acceder a él.

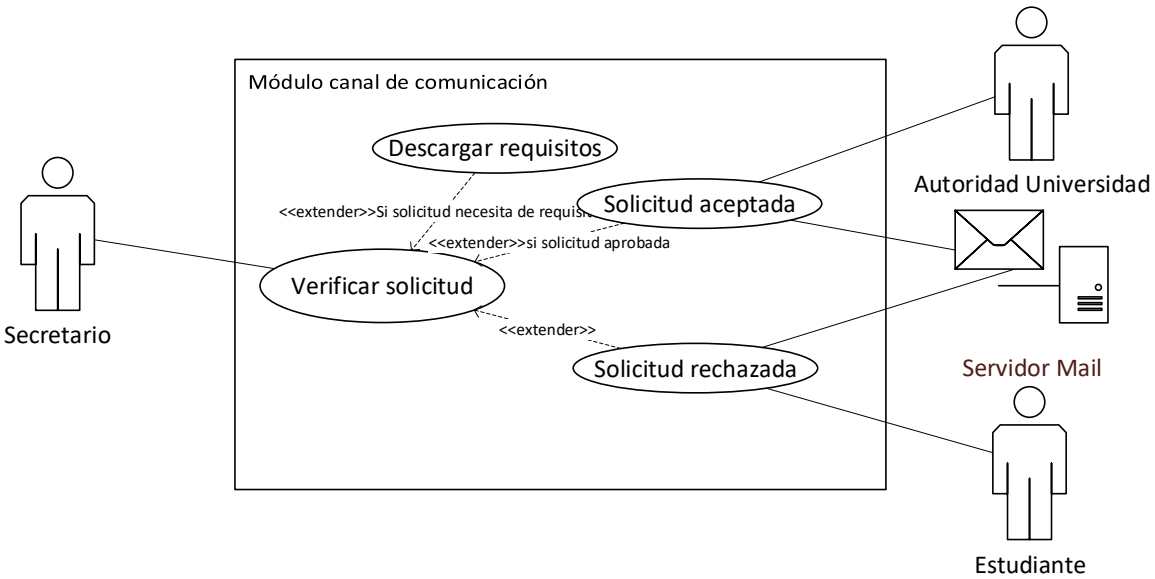


Figura 2.3 Casos de uso para el módulo canal de comunicación - Secretario

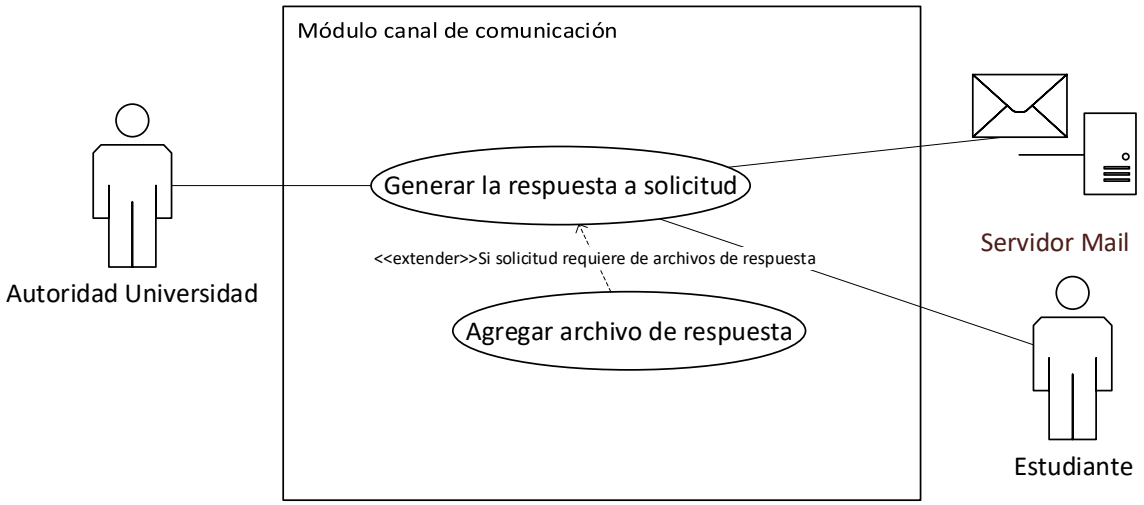


Figura 2.4 Casos de uso para el módulo canal de comunicación - Autoridad

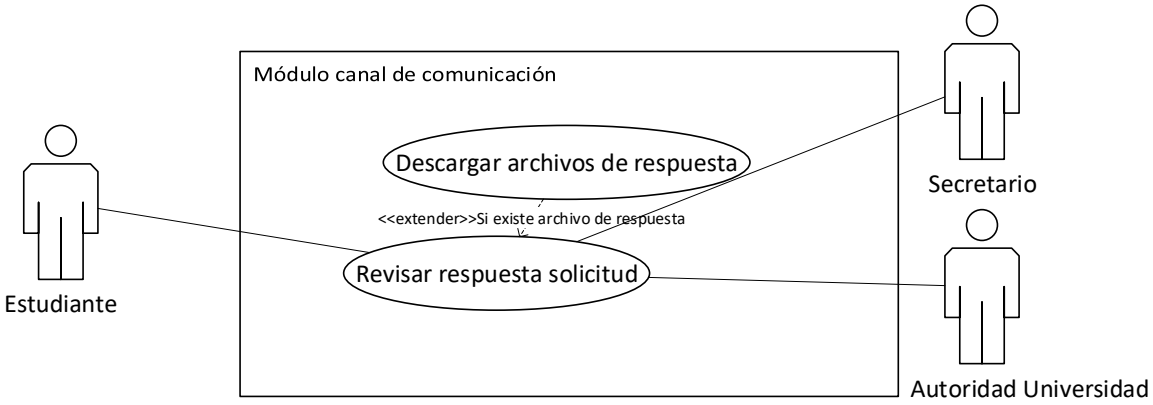


Figura 2.5 Casos de uso para el módulo canal de comunicación - Estudiante

El diagrama de casos de uso correspondiente al módulo de estado se muestra en la Figura 2.6, se tienen dos actores los cuales podrán realizar la acción de revisar el historial de solicitudes, pero solo el actor *Autoridad* podrá realizar la acción de revisar estadísticas.

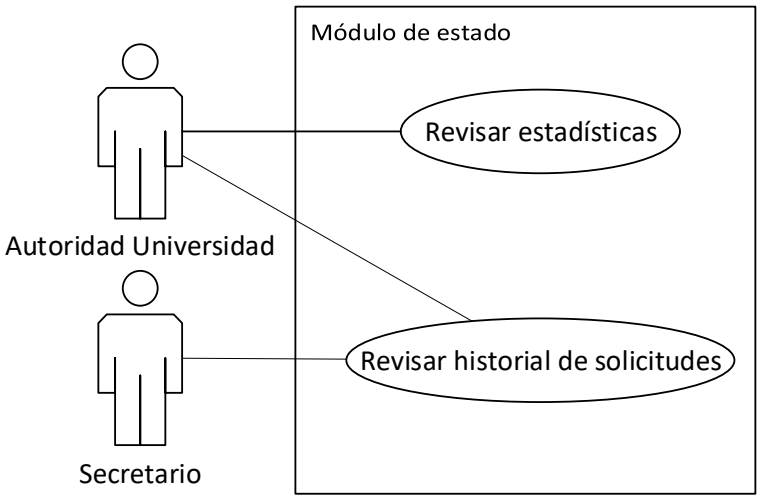


Figura 2.6 Casos de uso para el módulo de estado

En base a los requerimientos, se vio la necesidad de crear un módulo más, que es el módulo de Usuario, como se muestra en la Figura 2.7, donde se tiene al actor *Estudiante* el cual puede realizar tres acciones que son: Registrarse como usuario, actualizar su información y cambiar la contraseña. En la Figura 2.8 se observa al actor *Administrador* que podrá realizar las acciones de Crear, Actualizar, Ver y Eliminar usuarios.

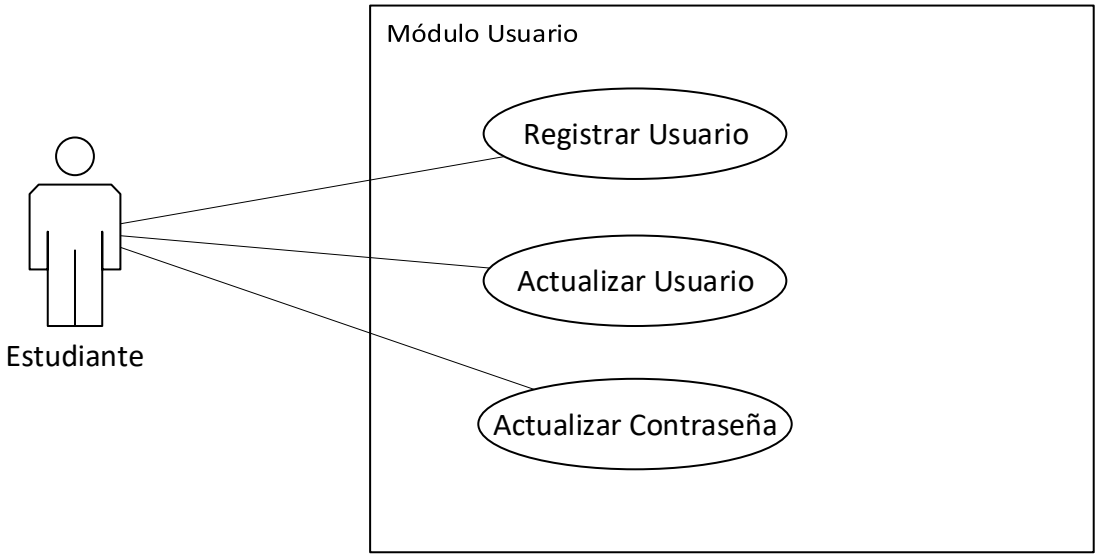


Figura 2.7 Casos de uso para el módulo usuario - Estudiante

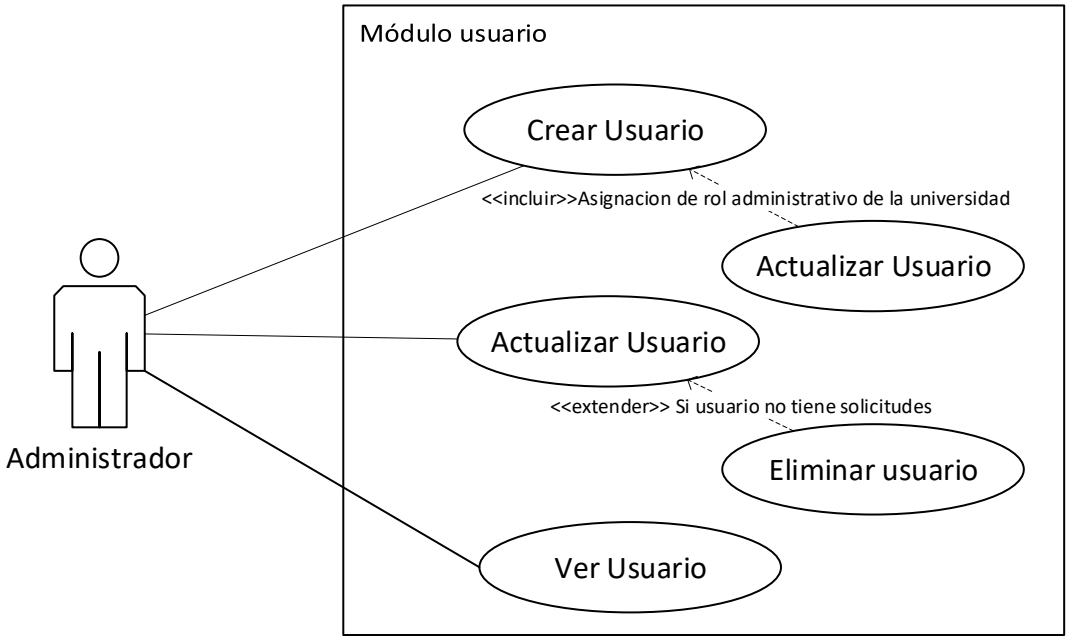


Figura 2.8 Casos de uso para el módulo usuario - Administrador

2.2.2 DIAGRAMA ENTIDAD RELACIÓN

El diagrama entidad-relación se usó para el diseño de la base de datos. Para la creación de la aplicación web se hizo uso de *DataBase First* para lo cual se creó primero la base de datos y con la ayuda de *Entity Framework* se crearon las clases necesarias para ser utilizadas en la aplicación. En la Figura 2.9 se muestra el diagrama a usar.

2.2.3 DIAGRAMA RELACIONAL

El diagrama relacional se muestra en la Figura 2.10, será el que se use para crear la base de datos, toda esta información ha sido obtenida de los requerimientos y se obtuvieron las tablas que se indican en la Tabla 2.3

Tabla 2.3 Tablas de la base de datos (Parte 1 de 2)

Nombre de la tabla	Detalle
Usuario	Contiene los registros del usuario como nombres, apellidos, teléfono, cédula, contraseña entre otros
Carrera	Contiene los registros de las carreras de la facultad
Facultad	Contiene los registros de las facultades de la universidad
Plantilla	Contiene los registros necesarios para una solicitud como: nombre de la autoridad a la que está siendo dirigida la solicitud, tiempo de respuesta de la solicitud entre otros

Tabla 2.3 Tablas de la base de datos (Parte 2 de 2)

Nombre de la tabla	Detalle
Requisito	Contiene los registros de los requisitos que se necesita para la plantilla
PlantillaRequisito	Tabla que sirve para romper la relación N:N de la tabla Plantilla y Requisito.
Solicitud	Contienen los registros de la solicitud que será creada por el usuario

La tabla Usuario cuenta con roles de usuario es decir se guarda información tanto del administrador, autoridades y de los estudiantes, en la Tabla 2.4 se muestra la descripción de los campos de la tabla Usuario.

La contraseña después de ser convertida en Hash es codificada en base64 utilizando esta función de C# (Convert.ToBase64String), la cual permite almacenar la contraseña como un *string* y no una cadena de bytes.

Tabla 2.4 Campos de la tabla Usuario

Columna	Tipo de dato	Detalle
IDUsuario	Int	Clave primaria.
IDCarrera	Int	Clave foránea que corresponde al identificador único de la Carrera.
Nombre	varchar(50)	Nombre del usuario.
Apellido	varchar(50)	Apellido del usuario.
CorreoElectronico	varchar(50)	Correo electrónico.
Telefono	varchar(10)	Número de teléfono.
Cedula	varchar(10)	Número de cédula.
Contrasena	varchar(512)	Contraseña.
Activo	Bit	¿Está vigente en la universidad?
RolFacultad	Int	Que rol desempeña: Autoridad, Secretario, Estudiante.
UsuarioCreacion	Int	ID del usuario que creó al usuario
UsuarioModificacion	Int	ID del usuario que modificó la información del usuario
FechaCreacion	Datetime	Fecha de creación del usuario
FechaCreacion	Datetime	Fecha de creación del usuario

La tabla Facultad tiene un único campo que es el nombre de la facultad; para esta tabla se tendrá un solo registro que será el de la Facultad de Ingeniería Eléctrica y Electrónica en

vista que el alcance es solo para la carrera de Ingeniería en Electrónica y Redes de Información. En la Tabla 2.5 se detallan los campos de esta tabla.

Tabla 2.5 Campos de la tabla Facultad

Columna	Tipo de dato	Detalle
IDFacultad	Int	Clave primaria
NombreFacultad	varchar(50)	Nombre de la facultad

La tabla Carrera registra el nombre de la carrera, en este caso se tendrá un solo registro que será el de la carrera de Ingeniería en Electrónica y Redes de Información, dado el alcance de este trabajo de titulación, aunque al igual que con la facultad si se desea implementar este trabajo para toda la universidad solo es cuestión de agregar nuevos registros a las tablas respectivas.

En la Tabla 2.6 se detallan los campos de esta tabla.

Tabla 2.6 Campos de la tabla Carrera

Columna	Tipo de dato	Detalle
IDCarrera	Int	Clave primaria
NombreCarrera	varchar(50)	Nombre de la carrera
IDFacultad	Int	Clave foránea que corresponde al identificador único de la Facultad

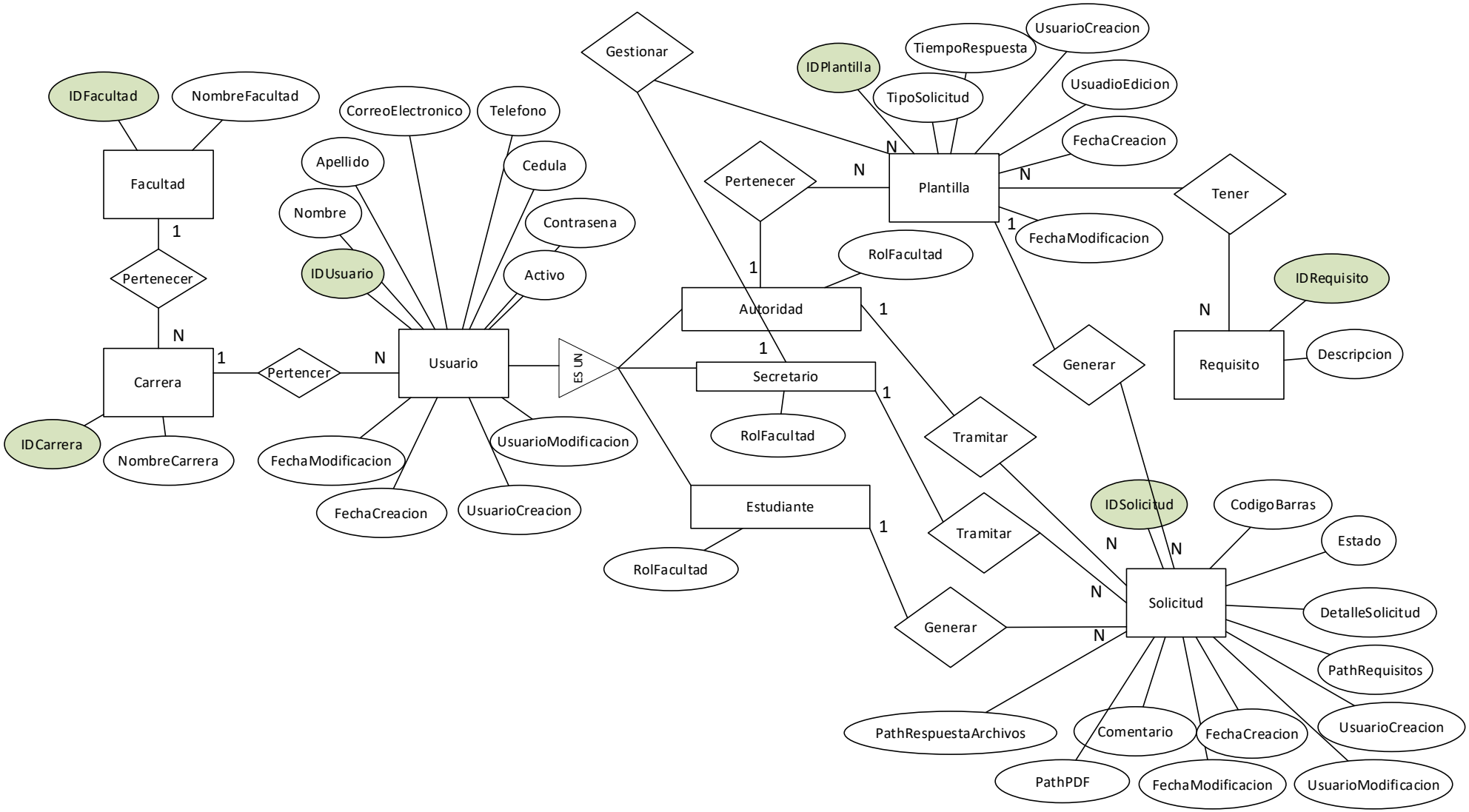


Figura 2.9 Diagrama Entidad-Relación

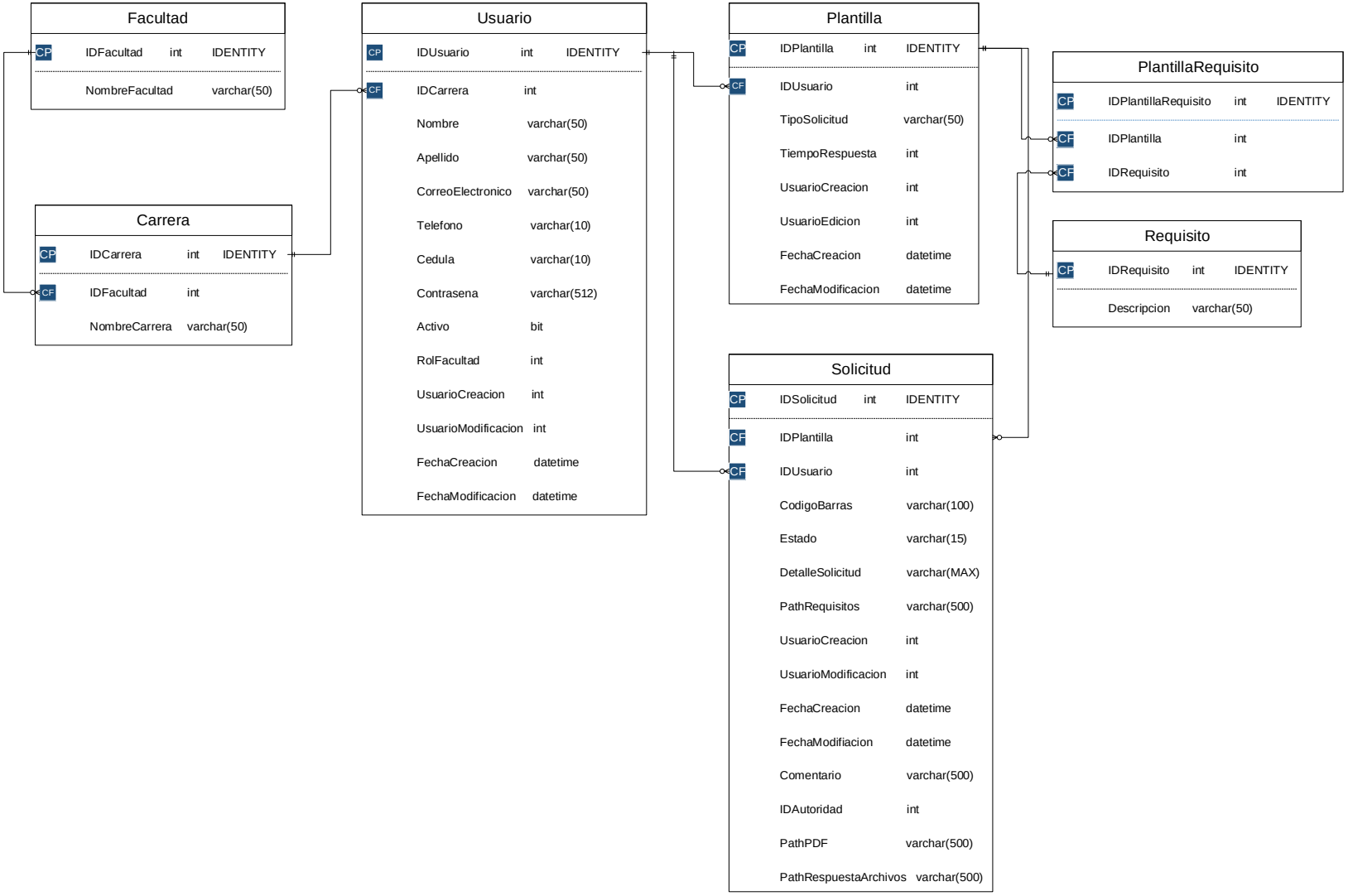


Figura 2.10 Diagrama relacional

Como se muestra en la Tabla 2.7, la tabla Plantilla contiene información referente a datos preestablecidos que tendrá una solicitud especifica según el tipo de pedido que se seleccione.

Tabla 2.7 Campos de la tabla Plantilla

Columna	Tipo de dato	Detalle
IDPlantilla	Int	Clave primaria
IDUsuario	Int	Clave foránea correspondiente al identificador único del usuario, con el rol autoridad a la que va dirigida el pedido
TipoSolicitud	varchar(50)	Tipo de pedido que se tendrá
TiempoRespuesta	Int	Tiempo estimado en el que el pedido va a tener una respuesta
UsuarioCreacion	Int	ID del usuario que creó la plantilla para la solicitud
UsuarioModificacion	Int	ID del usuario que modificó la plantilla para la solicitud
FechaCreacion	Datetime	Fecha de creación de la plantilla
FechaModificacion	Datetime	Fecha de modificación de la plantilla

La tabla Requisito contiene información de los diferentes requisitos que se pueden necesitar para poder dar el trámite respectivo al pedido que realice un estudiante.

En la Tabla 2.8 se muestra el detalle de sus campos.

Tabla 2.8 Campos de la tabla Requisito

Columna	Tipo de dato	Detalle
IDRequisito	Int	Clave primaria
Descripcion	varchar(50)	Tipo de requerimientos que va a tener una solicitud

La Tabla 2.9, referente a la Solicitud contiene información completa del detalle de una solicitud, con las relaciones a las claves foráneas se puede acceder a la información personal del usuario como: nombre, apellido, cédula etc., y también acceder a la información de la plantilla en caso de que el usuario haya decidido generar una solicitud con plantilla definida.

Tabla 2.9 Campos de la tabla Solicitud

Columna	Tipo de dato	Detalle
IDSolicitud	Int	Clave primaria.
IDPlantilla	Int	Clave foránea que indica el identificador único de Plantilla.
IDUsuario	Int	Clave foránea que indica el identificador único de Usuario.
CodigoBarras	varchar(MAX)	Código de barras para identificación del usuario.
Estado	varchar(15)	Estado de la solicitud, este puede ser: Enviado, Aceptado, Aprobado o Rechazado.
DetalleSolicitud	varchar(MAX)	Texto de la solicitud con toda la información completa.
PathRequisitos	varchar(500)	Ruta de los requisitos que se han subido a la aplicación.
UsuarioCreacion	Int	ID del usuario que creó la solicitud.
UsuarioModificacion	Int	ID del usuario que modificó la solicitud.
FechaCreacion	datetime	Fecha y hora de creación de la solicitud.
FechaModificacion	datetime	Fecha y hora de la modificación de la solicitud.
Comentario	varchar(500)	Campo necesario para especificar el porqué de la respuesta a la solicitud.
IDAutoridad	Int	ID que carga la información de la autoridad a la que va dirigida la solicitud.
PathPDF	varchar(500)	Ruta para poder descargar la solicitud en formato pdf
PathRespuestaArchivos	varchar(500)	Ruta para descargar el archivo de respuesta a una solicitud.

2.2.4 DIAGRAMA DE CLASES

Especifica el nombre, atributos y métodos de las clases.

En ASP.NET MVC el Modelo está representado por clases al igual que los Controladores

2.2.4.1 Modelo

En este proyecto se separaron las clases que se han creado a partir de la base de datos usando *Database first Entity Framework* y las clases que se crearon para la manipulación de los datos sin tener que afectar el diseño actual de la base de datos. Las clases creadas de forma automática al generar el Modelo con *Entity Framework* se muestran en la Figura 2.11 y son:

- *Usuario.cs*: que es la representación de la tabla Usuario, una de sus funciones es desplegar una lista de todos los usuarios de la aplicación.

- *Carrera.cs* que es la representación de la tabla Carrera, una de sus funciones es desplegar una lista de carreras que tiene cierta facultad.
- *Facultad.cs* que es la representación de la tabla Facultad, una de sus funciones es desplegar una lista de facultades que tiene la universidad.
- *Plantilla.cs* que es la representación de la tabla Plantilla, una de sus funciones es deseplegar una lista de pedidos de solicitudes más comunes que son realizadas por los *Estudiantes.*
- *Requisito.cs* que es la representación de la tabla Requisito, una de sus funciones es desplegar una lista de requisitos los cuales se irán relacionando con el tipo de pedido que se vaya creando en una *Plantilla.*
- PlantillaRequisito.cs que es la representación de la tabla PlantillaRequisito, su principal función es la asociación de *Plantillas* con *Requisitos*.
- Solicitud.cs que es la representación de la tabla Solicitud, una de sus funciones es desplegar una lista de solicitudes que el *Estudiante* haya creado ya sean estas con una plantilla definida o una plantilla redactada de manera personal

En la Figura 2.12, se muestra el diagrama de clases que se crearon de forma manual en la carpeta Modelo correspondiente al proyecto son:

- *Login.cs* se crea con el fin de verificar las credenciales para acceder a la aplicación.
- *CambioContrasena.cs* clase creada con el fin de permitir el cambio de contraseña de los usuarios.
- *Registro.cs* clase creada para el registro de usuarios nuevos, estos se crearán con el rol *Estudiante*.
- *SolicitudSinPlantilla.cs* clase creada para las solicitudes que se van a escribir sin un formato establecido.
- *PlantillaRequisitoViewModel.cs* clase creada para hacer uso de los atributos de las clases *Plantilla* y *Requisito* en una sola Vista.

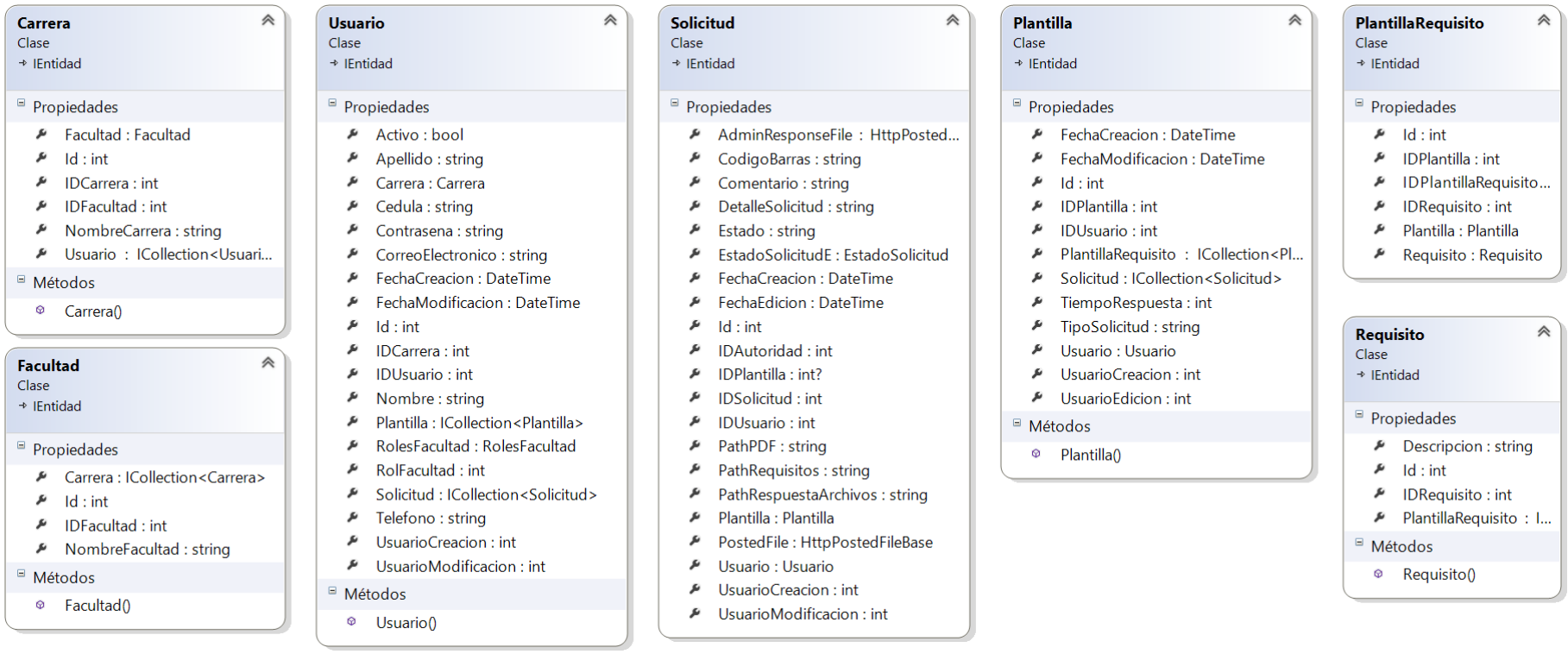


Figura 2.11 Diagrama de clases del Modelo

También se crearon las clases en la *DAL, BLL* y clases *partial*:

Las clases partial: tiene algún o algunos atributos de la clase original se crearon con el fin de agregar validaciones del lado del cliente, esas clases son: *UsuarioPartial, CarreraPartial, FacultadPartial, PlantillaPartial, RequisitoPartial, Plantilla_RequisitoPartial, SolicitudPartial.*

Las clases repositorio se crearon para el acceso a datos Figura 2.13, la separación en capas mejora el desarrollo de la aplicación, las clases repositorio son: *UsuarioRepositorio, CarreraRepositorio, FacultadRepositorio, PlantillaRepositorio, RequisitoRepositorio, SolicitudRepositorio*, en esas clases está la instancia a la base de datos.

La Tabla 2.10 muestra los métodos de la clase repositorio, serán los mismos para todas las clases, lo que cambia es el Modelo al que se accede para obtener la información.

Tabla 2.10 Métodos que se implementan en las clases repositorio

Operación	Descripción
Add	Inserta un registro a la tabla
Update	Actualiza un registro a la tabla
Delete	Elimina un registro de la tabla
List	Devuelve todos los registros de la tabla
FindById	Devuelve un solo registro de la tabla basado en la clave primaria

La capa de negocios fue implementada en la carpeta *Servicios*. La capa de negocios se convierte en un servicio que provee la información que necesita la capa de presentación en este caso los Controladores, se utiliza la inyección de dependencia para enviar el contexto de la base de datos desde el constructor del Controlador, esto permite crear una solución en la cual la capa de presentación no depende de la capa de datos, donde la capa de datos es definida por la capa de presentación.

Las clases creadas son: *UsuarioServicio.cs, CarreraServicio.cs, PlantillaServicio.cs, RequisitoServicio.cs y SolicitudServicio.cs.*

El diagrama para las clases Servicio se indica en la Figura 2.14.

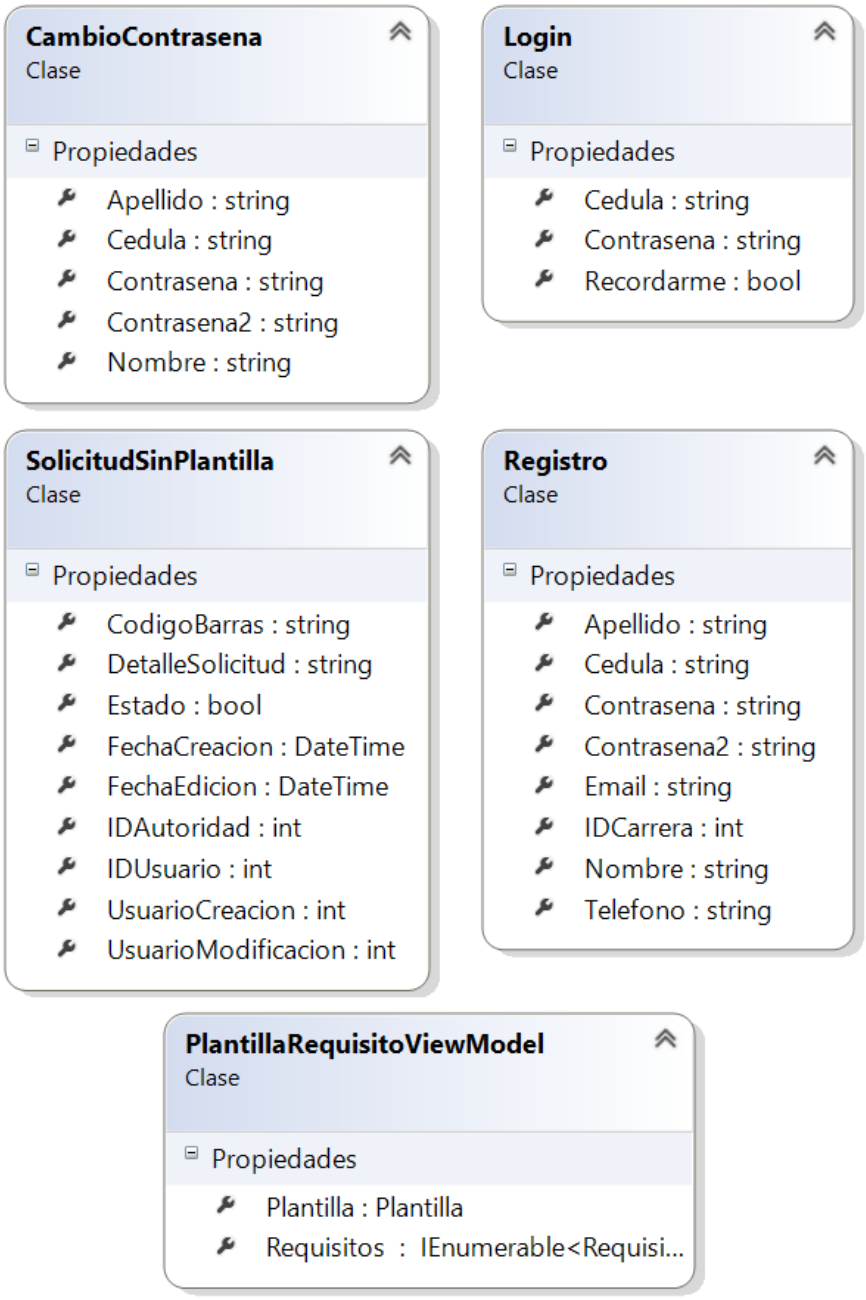


Figura 2.12 Diagrama de clases de los Modelos creados manualmente

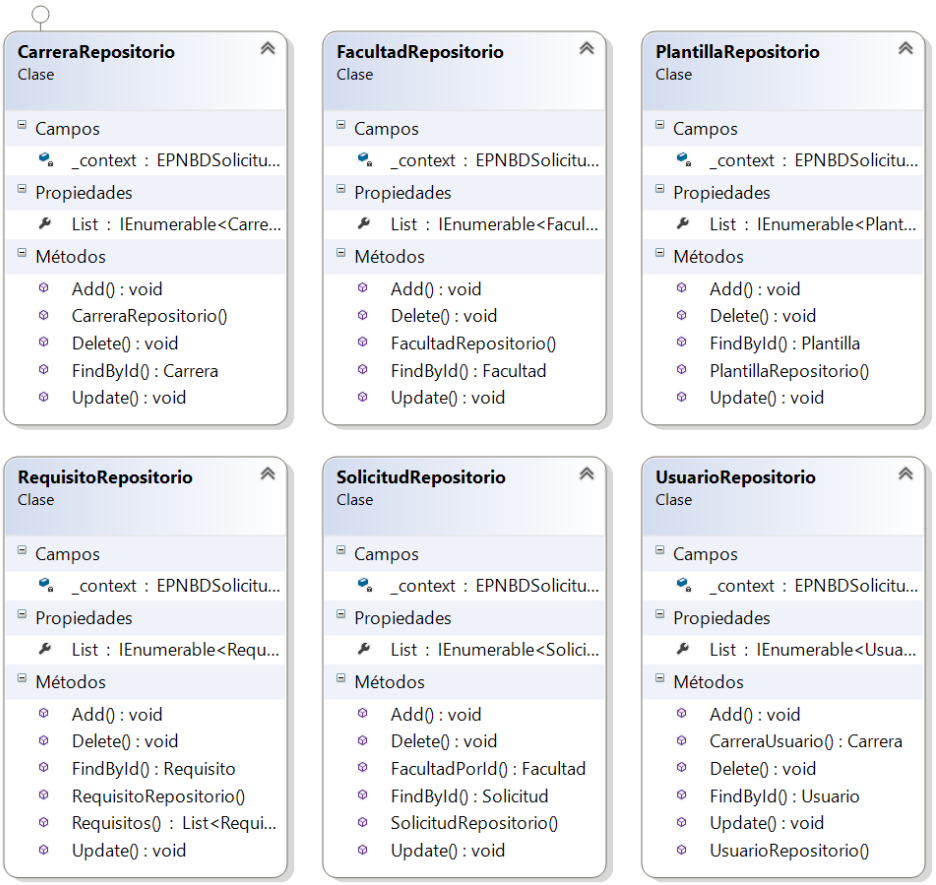


Figura 2.13 Diagrama de clases de las clases Repositorio

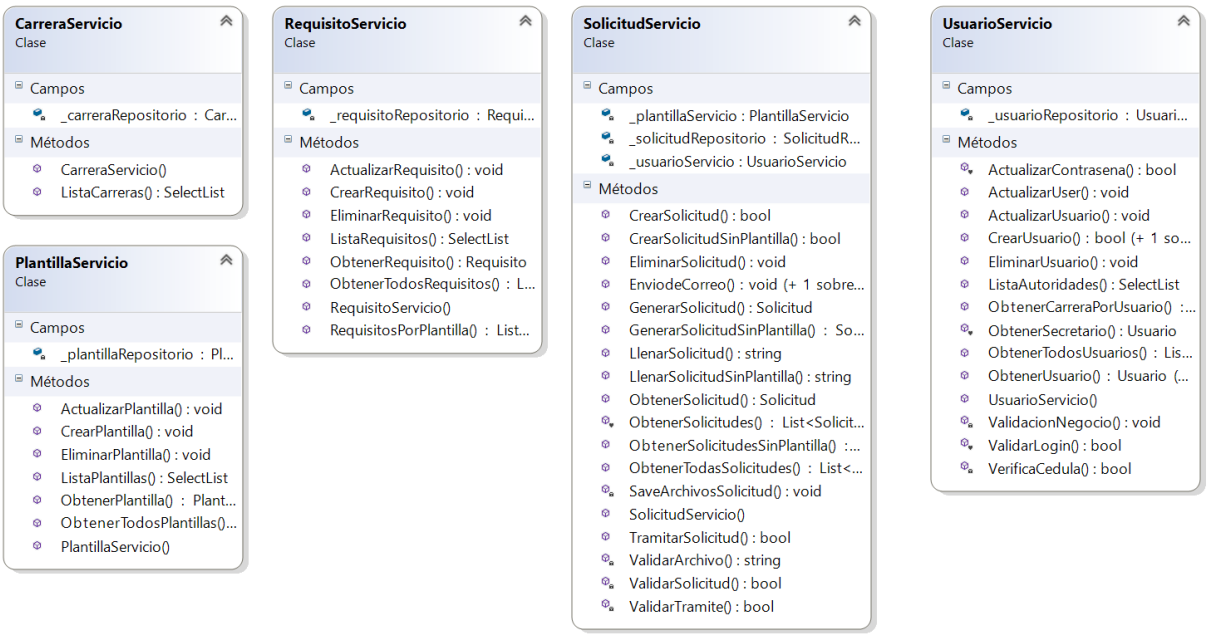


Figura 2.14 Diagrama de clases de las clases Servicio

2.2.4.2 Controlador

Cada Controlador representa una clase como se muestra en la Figura 2.15, en esta aplicación se cuenta con los siguientes Controladores:

- *AdminSolicitudController.cs*: contiene métodos para ver y modificar solicitudes, los campos que son modificables son el *estado* y el *comentario*; también contiene métodos para ver y descargar requisitos, y un método para la realización de la gráfica de estadísticas.
- *HomeController.cs*: usado para desplegar información estática de cómo funciona la aplicación.
- *LoginController.cs*: contiene métodos para validar el acceso de la aplicación, método para salida de la aplicación, y un método para permitir el registro de nuevos usuarios.
- *MiPerfilController.cs* contiene el método necesario para el cambio de contraseña.
- *UsuarioController.cs*: contiene métodos y atributos para crear, leer, modificar y eliminar usuarios, y un método especial para diferenciar entre si el usuario creado tiene el rol de estudiante o si tiene un rol diferente
- *PlantillaController.cs*: contiene métodos y atributos para crear, leer, modificar y eliminar plantillas.
- *RequisitoController.cs*: contiene métodos y atributos para crear, leer, modificar y eliminar requisitos.
- *Plantilla_RequisitoController.cs*: contiene métodos y atributos para crear nuevas asignaciones entre las plantillas y requisitos.
- *SolicitudesController.cs:* contiene métodos y atributos para crear, leer, modificar y eliminar solicitudes con una plantilla definida, también contiene métodos para cargar el detalle de la solicitud y los requisitos que ésta requiere.
- *SolicitudSinPlantillaController.cs:* contiene métodos y atributos para crear, leer, modificar y eliminar solicitudes sin una plantilla definida, también contiene el método para cargar el detalle de la solicitud.

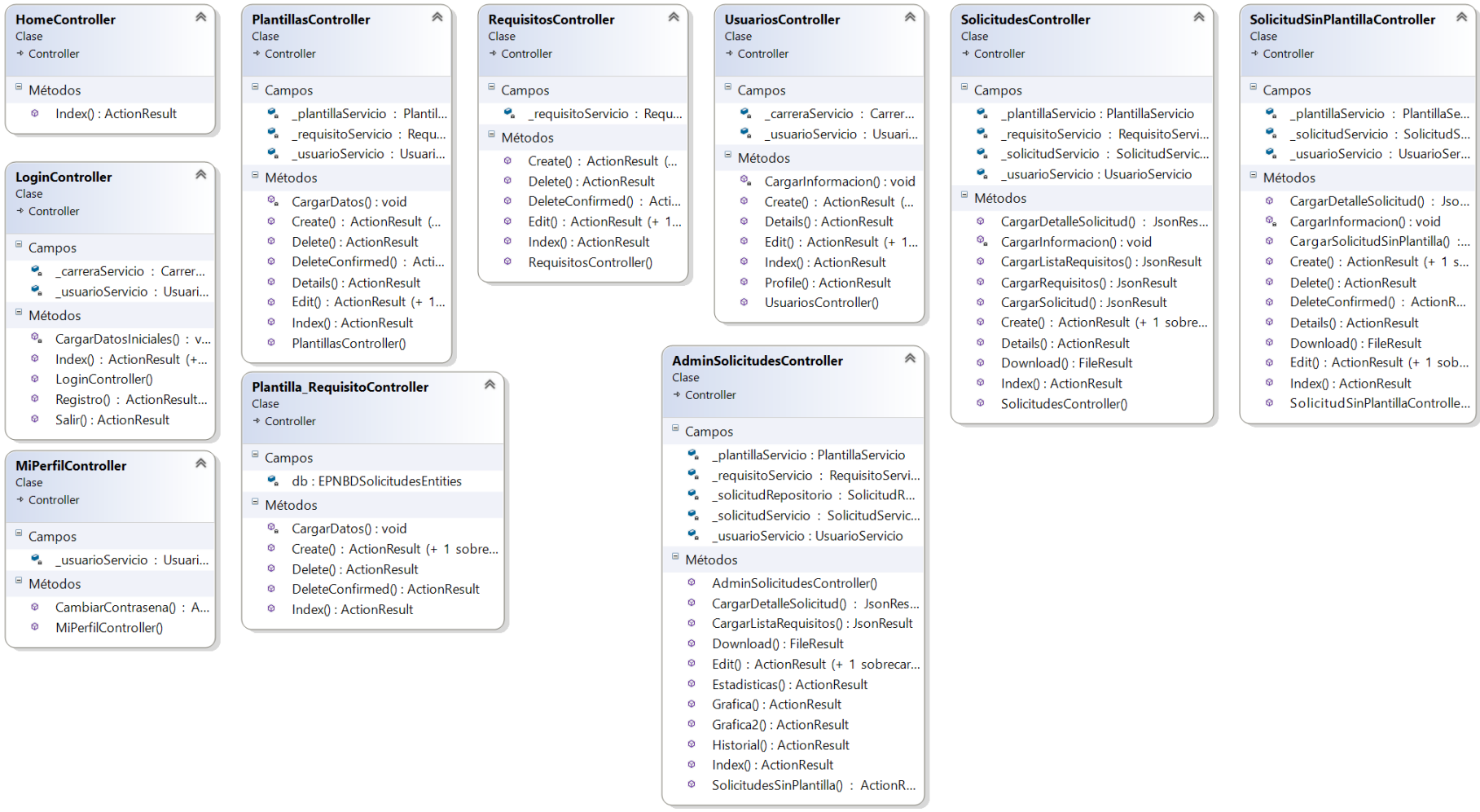


Figura 2.15 Diagrama de clases del Controlador

2.2.5 PRODUCT BACKLOG

Se listan todas las tareas que se van a realizar durante el proceso de desarrollo de la aplicación web.

2.2.5.1 Definición de roles

Antes de proceder a crear el *product backlog* se definen los roles de quien o quienes asumen las funciones que *Scrum* requiere. En la Tabla 2.11 se representan los roles para desarrollar la aplicación.

Tabla 2.11 Definición de roles

Rol	Nombre	Detalles
Product Owner	Pablo Hidalgo	Coordinador de la Carrera de Ingeniería Electrónica y Redes de Información.
Scrum Master	Franklin Sánchez	Líder del proyecto y moderador entre el *Product Owner* y el Desarrollador
Desarrollador	Vanessa Chenaz	Desarrollador de software

2.2.5.2 Definición del *product backlog*

Se ha detallado en tareas más fáciles y simples los requerimientos necesarios para el desarrollo de la aplicación. La Tabla 2.12, representan las tareas que se van a realizar por cada requerimiento.

Tabla 2.12 Product Backlog (Parte 1 de 1)

Prioridad	Tareas
1	Crear nuevos Usuarios
2	Leer datos de los usuarios
3	Modificar información de los Usuarios
4	Autenticar las credenciales de usuario para acceder a la aplicación.
5	Crear nueva Plantilla
6	Leer datos de la Plantilla
7	Modificar información de la Plantilla
8	Eliminar una Plantilla
9	Crear nuevo Requisito
10	Leer Requisito
11	Modificar Requisito
12	Eliminar Requisito
13	Crear asignación de los Requisitos a su respectiva Plantilla.

Tabla 2.12 Product Backlog (Parte 2 de 2)

Prioridad	Tareas
14	Leer las respectivas asignaciones.
15	Crear una Solicitud con plantilla definida
16	Leer datos de la solicitud con plantilla
17	Crear una Solicitud sin plantilla definida
18	Leer datos de la solicitud sin plantilla
19	Obtener todas las solicitudes que requieran ser tramitadas
20	Dar el respectivo trámite a cada una de las solicitudes.
21	Dar una respuesta a las solicitudes recibidas
22	Enviar un correo electrónico en cada cambio de estado de la solicitud
23	Permitir cargar requisitos cuando se crea una solicitud
24	Permitir cargar archivos como respuesta a una solicitud
25	Permitir mostrar un historial de solicitudes
26	Permitir mostrar un gráfico de estadísticas de las solicitudes.

2.2.5.3 Sprint backlog

Con los requerimientos y los roles definidos se procedió a crear el *Sprint Backlog* como se muestra en la Tabla 2.13.

Tabla 2.13 Sprint Backlog (Parte 1 de 2)

Sprint	Nombre	Requerimiento
SPRINT 1	Acceso a la aplicación.	1. Permitir creación, lectura y modificación de Usuarios. 2. Permitir el ingreso del usuario a la aplicación.
SPRINT 2	Gestión de Plantillas	3. Permitir creación, lectura, modificación y eliminación de Plantillas 4. Permitir creación, lectura, modificación y eliminación de Requisitos 5. Permitir la asignación de Requisitos a las respectivas Plantillas
SPRINT 3	Gestión de Solicitudes	6. Permitir creación y lectura de solicitudes con una plantilla definida 7. Permitir creación y lectura de solicitudes sin plantilla definida 8. Permitir dar trámite a una solicitud.

Tabla 2.13 Sprint Backlog (Parte 2 de 2)

Sprint	Nombre	Requerimiento
SPRINT 4	Gestión de respuesta	9. Permitir el envío de correo electrónico en cada cambio de estado de la Solicitud 10. Agregar archivos a la aplicación. Permitir la visualización de estadísticas de las solicitudes tramitadas
SPRINT 5	Publicación de la aplicación	11. Publicar la aplicación en un sitio web.

2.2.6 SPRINT 1 – ACCESO A LA APLICACIÓN

2.2.6.1 Creación de la base de datos

Cuando se empezó a desarrollar el proyecto se utilizó *Code First* pero debido a las múltiples complicaciones que daba en cuanto al desarrollo del proyecto se decidió utilizar *DataBase First* comprobando de esa manera que *DB* es más rápido y eficiente de utilizar en este proyecto porque se cuenta con una base de datos pequeña.

Para la creación de la base de datos se hizo uso de *SQL Server Management Studio 2017*; la codificación de scripts se realizó en el lenguaje *SQL*. El *script* de la creación de la base de datos se puede observar en el ANEXO C. En el Código 2.1 se muestra la creación de la base de datos *EPNBDSolicitudes*.

```
CREATE DATABASE [EPNBDSolicitudes]
USE [EPNBDSolicitudes]
```

Código 2.1 Creación de la base de datos

2.2.6.2 Creación de la aplicación web

La aplicación se desarrolló utilizando el *IDE Visual Studio 2017* y usando el lenguaje de programación C#.

El proyecto se llama GestionSolicitudesEPN, dentro de la solución que lleva el mismo nombre. El *framework* que se utilizó fue el .Net 4.6.1. En la Figura 2.16 se indica la manera en la que el proyecto fue creado.

- File > New > Project

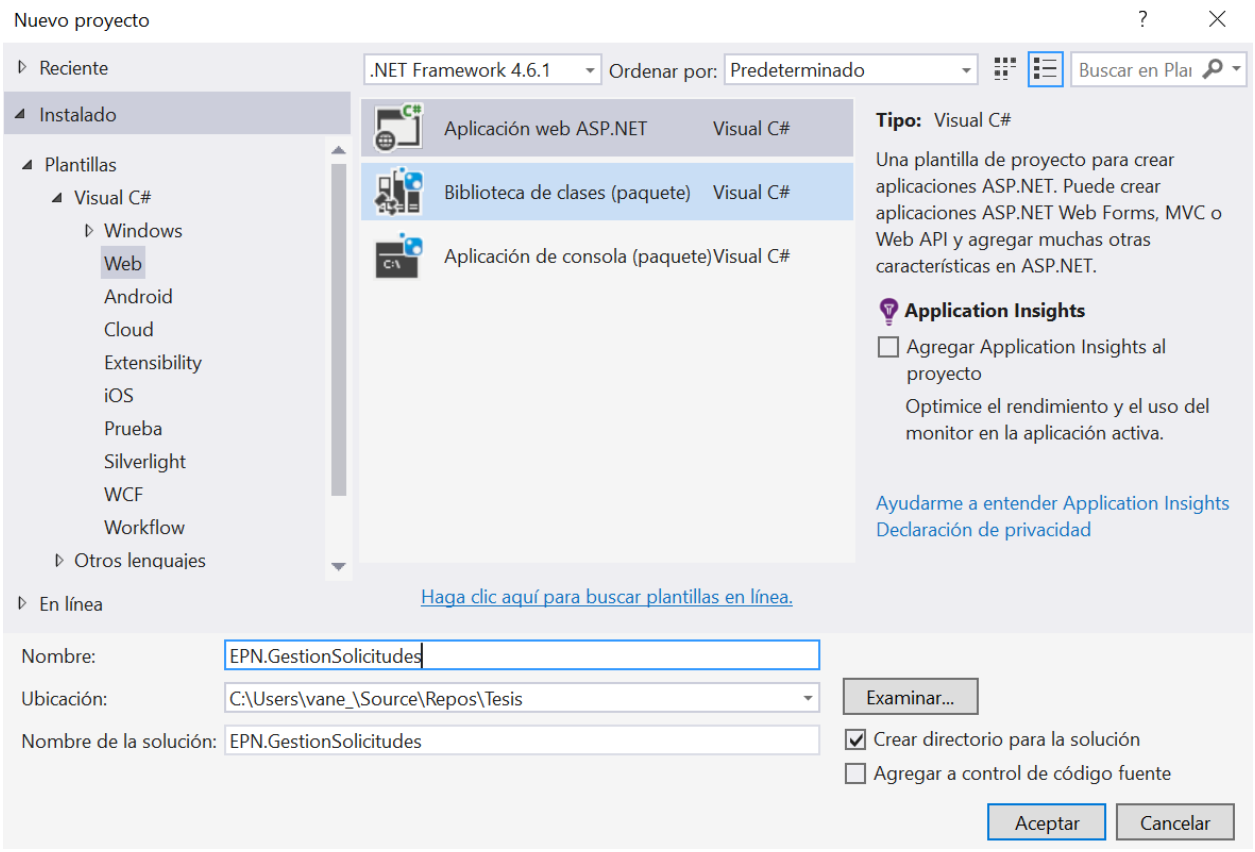


Figura 2.16 Creación del proyecto EPN.GestionSolicitudes

La aplicación se desarrolló en ASP.NET MVC por lo que en la Figura 2.17 muestra la selección de la plantilla MVC para este proyecto, también se muestra la elección de agregar pruebas unitarias en la solución.

Entre las carpetas que se crean en la aplicación están las carpetas: *Models, Controllers, Views, Scripts, Content*, que son las que más se va a utilizar. El archivo Web.config es de suma importancia para la configuración de la cadena de conexión como se muestra en el Código 2.2.

Entre los parámetros que se configuraron están: *name* que indica el nombre de su cadena de conexión, *connectionString* que indica con qué servidor comunicarse y el nombre de la base de datos, *providerName* que informa a los usuarios de la cadena de conexión qué proveedor de datos de *.NET Framework* debe usar al comunicarse con la base de datos [40]

El código del proyecto se encuentra en el ANEXO D.

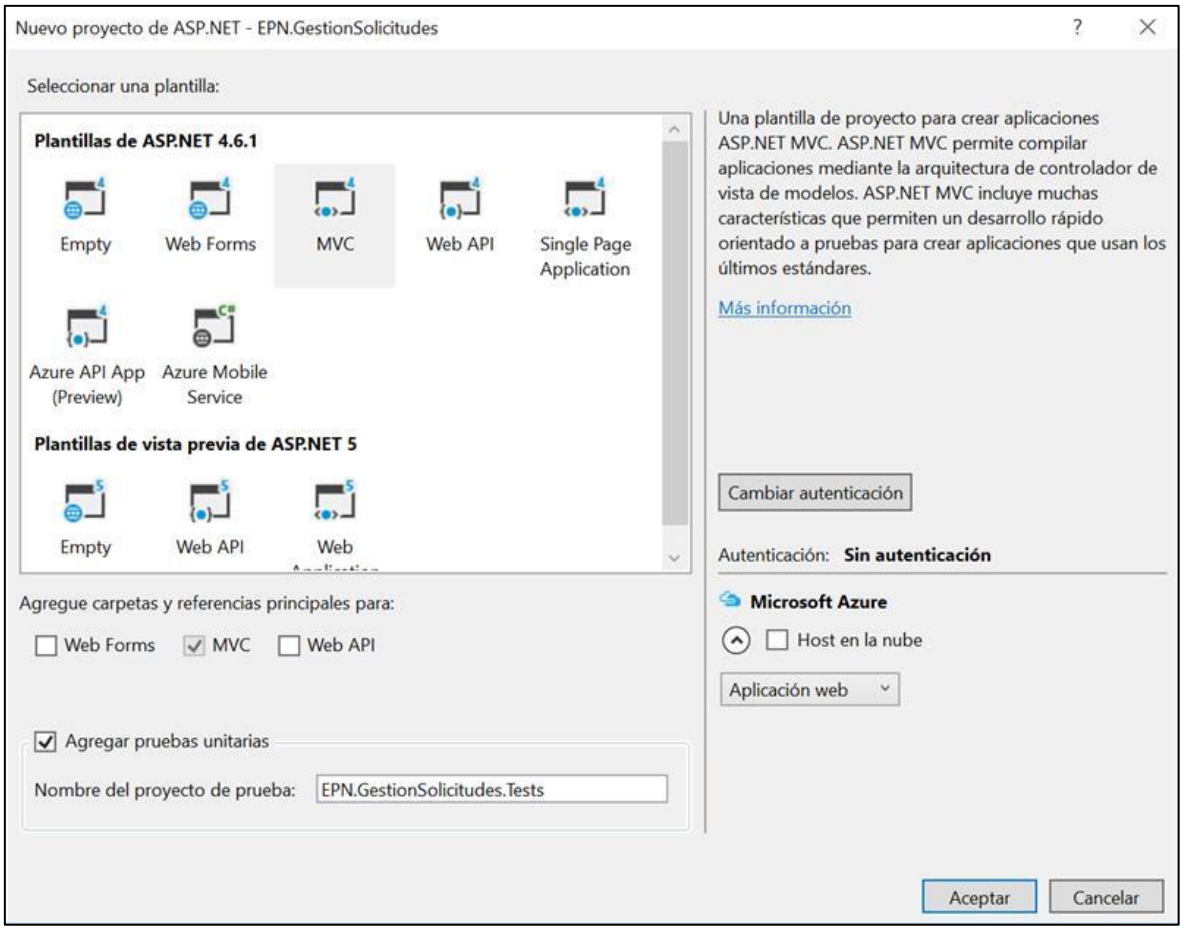


Figura 2.17 Selección de la plantilla MVC para el proyecto

```
11  <connectionStrings>
12    <add name="EPNBDSolicitudesEntities" connectionString="metadata=res://*/DAL.Modelo.EPNModel.csdl|res://*/
      DAL.Modelo.EPNModel.ssdl|res://*/DAL.Modelo.EPNModel.msl;provider=System.Data.SqlClient;provider connection
      string=&quot;data source=DESKTOP-LSRDT4N;initial catalog=EPNBDSolicitudes;integrated
      security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />
13  </connectionStrings>
```

Código 2.2 Configuración de la conexión a la base de datos

2.2.6.3 Generación del modelo de objetos utilizando Database First

El desarrollo de esta aplicación web se realizó con arquitectura en capas, por lo que se creó la carpeta *DAL* donde se generó el *EDM (Entity Data Model)* haciendo uso de *Entity Framework DataBase First.*

Dentro de la carpeta DAL de creó una subcarpeta llamada Modelo donde se agregó el nuevo elemento *ADO.NET Entity Data* Model como se muestra en la Figura 2.18, y se procedió a asignarle el nombre necesario.

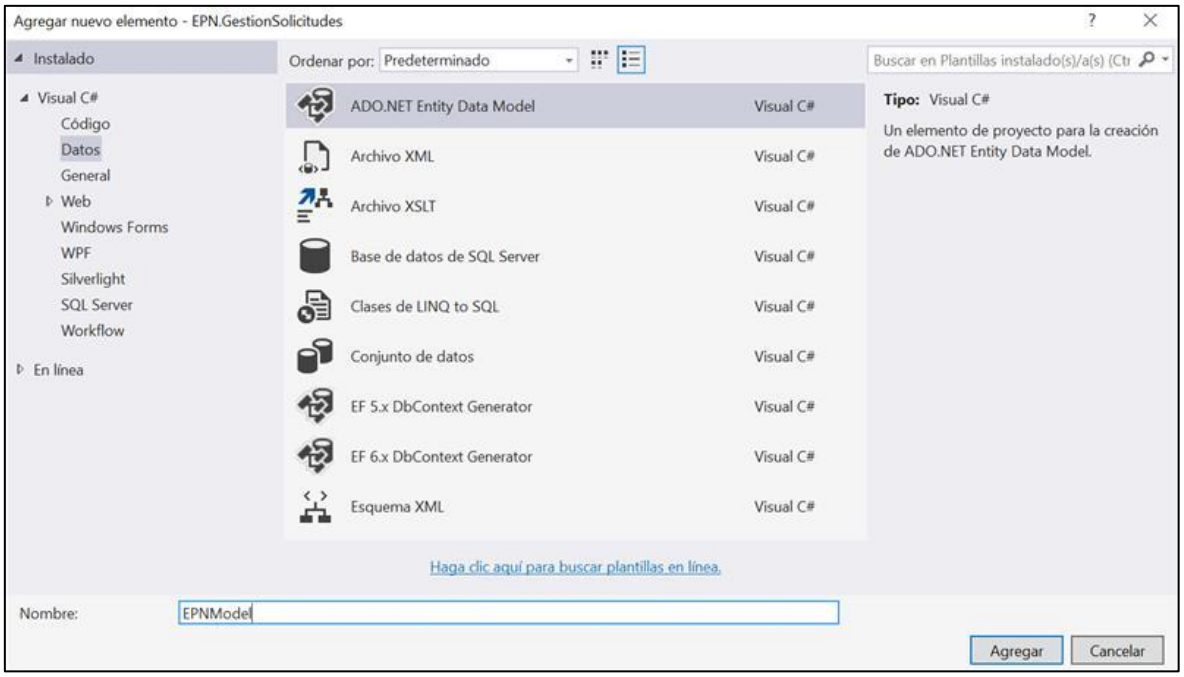


Figura 2.18 Agregar nuevo elemento para generar el Modelo

En la Figura 2.19, se escogió la primera opción que es *EF Designer desde base de datos*, debido a que el Modelo que se generó se lo hizo con el método *DataBase First.* El archivo se creó con la extensión .edmx.

Cuando se creó el Modelo *EPNModel.edmx*, se crearon de manera automática las siguientes Clases Entidad: *Carrera, Facultad, Plantilla, PlantillaRequisito, Requisito, Solicitud, Usuario* cada una de estas clases Entidad representan una tabla de la base de datos.

Se proceden a crear clases *partial*[11], en ellas se tendrán las *Data Annotatios* para realizar las validaciones de lado del cliente. Un ejemplo de este tipo de validaciones implementadas es la clase *UsuarioPartial.cs* que tiene algunos atributos de la clase *Usuario.cs,* como se observa en el Código 2.3.

[11] Partial Class: implementa la funcionalidad de una sola clase en múltiples archivos y todos estos archivos de clase se combinan en un solo archivo cuando la aplicación se compila

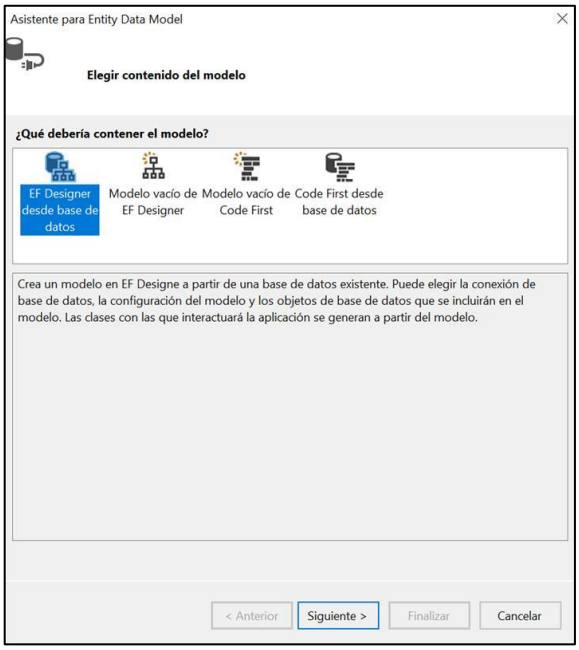


Figura 2.19 Generar Modelo a partir de la base de datos

Para el atributo *Nombre* se indica que es un campo requerido y si se deja en blanco se mostrará un mensaje de error, la longitud máxima para este atributo será de 50 caracteres y se va a desplegar en la Vista como “Nombres”. Los mismo *Data Annotations* se utilizan para el atributo *Apellido*, para el atributo *CorreoElectronico* se utiliza el campo *Required* con su respectivo mensaje de error si se deja en blanco y *Emailaddress* para comprobar que el formato del correo electrónico ingresado sea el correcto, también se va a desplegar en la Vista como “Correo Electronico”.

```
[StringLength(50)]
[DisplayName("Nombres")]
[Required(ErrorMessage = "Los nombres son requeridos")]
0 referencias | wilson geovanny panjon quinde, Hace 57 días | 1 autor, 2 cambios
public string Nombre { get; set; }

[StringLength(50)]
[DisplayName("Apellidos")]
[Required(ErrorMessage = "Los apellidos son requeridos")]
0 referencias | wilson geovanny panjon quinde, Hace 57 días | 1 autor, 2 cambios
public string Apellido { get; set; }

[DisplayName("Correo Electronico")]
[Required(ErrorMessage = "El Correo Electronico es requerido")]
[DataType(DataType.EmailAddress, ErrorMessage = "El correo no es valido")]
0 referencias | wilson geovanny panjon quinde, Hace 57 días | 1 autor, 2 cambios
public string CorreoElectronico { get; set; }
```

Código 2.3 Ejemplo de clase UsuarioPartial.cs

Se creó de manera automática la clase *Context EPNBDSolicitudesEntities* como se muestra en el Código 2.4, la cual permitió disponer de una sesión con la base de datos para obtener o guardar información en ella. En este proyecto se hizo uso de esta clase *Context* en las clases *repositorio*, en las cuales se hacen las respectivas instancias para acceder a la información de todas las tablas de la base de datos.

```
10  namespace EPN.GestionSolicitudes.DAL.Modelo
11  {
12      using System;
13      using System.Data.Entity;
14      using System.Data.Entity.Infrastructure;
15
16      public partial class EPNBDSolicitudesEntities : DbContext
17      {
18          public EPNBDSolicitudesEntities()
19              : base("name=EPNBDSolicitudesEntities")
20          {
21          }
22
23          protected override void OnModelCreating(DbModelBuilder modelBuilder)
24          {
25              throw new UnintentionalCodeFirstException();
26          }
27
28          public virtual DbSet<Carrera> Carrera { get; set; }
29          public virtual DbSet<Facultad> Facultad { get; set; }
30          public virtual DbSet<PlantillaRequisito> PlantillaRequisito { get; set; }
31          public virtual DbSet<Requisito> Requisito { get; set; }
32          public virtual DbSet<Usuario> Usuario { get; set; }
33          public virtual DbSet<Plantilla> Plantilla { get; set; }
34          public virtual DbSet<Solicitud> Solicitud { get; set; }
35      }
36  }
37
```

Código 2.4 Clase Context

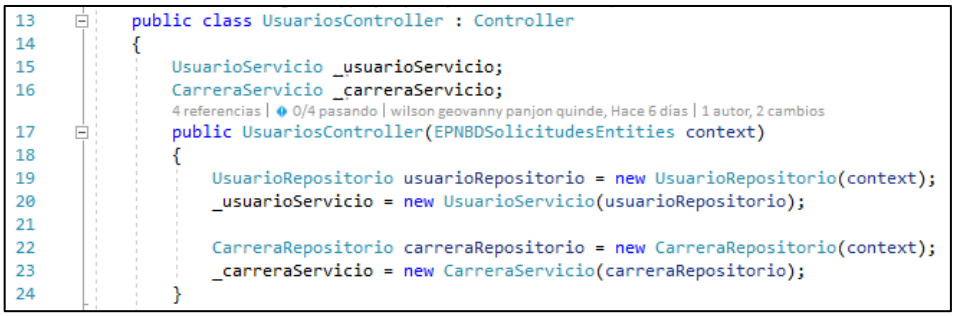
En el Código 2.5 se muestra un ejemplo del uso de la clase *Context* en la clase *UsuarioRepositorio*.

```
1   using EPN.GestionSolicitudes.DAL.Core;
2   using EPN.GestionSolicitudes.DAL.Modelo;
3   using System.Collections.Generic;
4   using System.Linq;
5
6   namespace EPN.GestionSolicitudes.DAL.Repositorio
7   {
8       public class UsuarioRepositorio
9       {
10          private EPNBDSolicitudesEntities _context = null;
11
12          public UsuarioRepositorio(EPNBDSolicitudesEntities context)
13          {
14              _context = context;
15
16          }
17
```

Código 2.5 Uso de clase context

En el Código 2.6 se indica un ejemplo de las instancias a las diferentes clases, información del Controlador que se está usando (línea 13), instancias a las clases *UsuarioServicio* y *CarreraServicio* para poder acceder a los métodos que hay en ellas (líneas 15 y 16).

Para acceder a los datos de la base, las clases servicio reciben como parámetro a la clase repositorio (líneas 20 y 23), la que a su vez tiene como parámetro a la clase *Context* que es en sí la conexión con la base de datos (líneas 19 y 22).

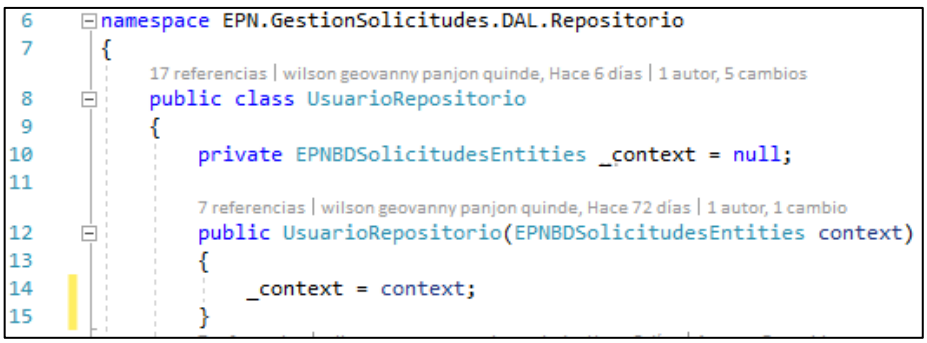


```
13  public class UsuariosController : Controller
14  {
15      UsuarioServicio _usuarioServicio;
16      CarreraServicio _carreraServicio;
        4 referencias | 0/4 pasando | wilson geovanny panjon quinde, Hace 6 días | 1 autor, 2 cambios
17      public UsuariosController(EPNBDSolicitudesEntities context)
18      {
19          UsuarioRepositorio usuarioRepositorio = new UsuarioRepositorio(context);
20          _usuarioServicio = new UsuarioServicio(usuarioRepositorio);
21
22          CarreraRepositorio carreraRepositorio = new CarreraRepositorio(context);
23          _carreraServicio = new CarreraServicio(carreraRepositorio);
24      }
```

Código 2.6 Ejemplo de Instancia de las clases de servicio y repositorio

2.2.6.4 CRUD de usuarios

Una vez que ya se tienen los Modelos se procedió a crear las clases respectivas. Para este *sprint* se empezó con las clases *UsuarioRepositorio* como se muestra en el Código 2.7, declaración de una variable de tipo *EPNBDSolicitudesEntities* (línea 10), constructor que va a recibir como parámetro la base de datos (línea 12) y se le asignó la clase *Context* (línea 14) a la variable que se creó al inicio (línea 10).



```
6   namespace EPN.GestionSolicitudes.DAL.Repositorio
7   {
        17 referencias | wilson geovanny panjon quinde, Hace 6 días | 1 autor, 5 cambios
8       public class UsuarioRepositorio
9       {
10          private EPNBDSolicitudesEntities _context = null;
11
            7 referencias | wilson geovanny panjon quinde, Hace 72 días | 1 autor, 1 cambio
12          public UsuarioRepositorio(EPNBDSolicitudesEntities context)
13          {
14              _context = context;
15          }
```

Código 2.7 Clase UsuarioRepositorio

En el Código 2.8, se muestran los métodos que se utilizó para: obtener una lista de usuarios (línea 18), agregar un nuevo usuario (línea 23), si todo es correcto se guardan los datos en la base (línea 24), eliminar un usuario de la tabla (línea 28), obtener un usuario específico de acuerdo con su clave primaria (línea 33).

Las clases repositorio van a tener los mismos métodos de acceso a la base de datos, lo que va a cambiar será el Modelo al que se haga referencia.

```
16          public IEnumerable<Usuario> List
17          {
18              get { return _context.Usuario; }
19
20          }
            2 referencias | wilson geovanny panjon quinde, Hace 72 días | 1 autor, 1 cambio
21          public void Add(Usuario entity)
22          {
23              _context.Usuario.Add(entity);
24              _context.SaveChanges();
25          }
            1 referencia | wilson geovanny panjon quinde, Hace 72 días | 1 autor, 1 cambio
26          public void Delete(Usuario entity)
27          {
28              _context.Usuario.Remove(entity);
29              _context.SaveChanges();
30          }
            1 referencia | wilson geovanny panjon quinde, Hace 72 días | 1 autor, 2 cambios
31          public Usuario FindById(int Id)
32          {
33              var result = (from r in _context.Usuario where r.IDUsuario == Id select r).FirstOrDefault();
34              return result;
35          }
            2 referencias | wilson geovanny panjon quinde, Hace 72 días | 1 autor, 1 cambio
36          public void Update(Usuario entity)
37          {
38              _context.Entry(entity).State = System.Data.Entity.EntityState.Modified;
39              _context.SaveChanges();
40          }
```

Código 2.8 Clase Repositorio para el Usuario

La clase *UsuarioServicio* fue la encargada de consultar la información de la base de datos a través de la clase *UsuarioRepositorio* y el resultado fue entregado al Controlador.

Para la explicación de los principales métodos *Index*, *Create*, *Edit* y *Delete* se dará un solo ejemplo de su uso debido que se requiere las mismas acciones lo que cambia es el Modelo.

- **Acción Index**

En el Código 2.9, se indica el método *Index*, perteneciente al Controlador *UsuariosController*, este método muestra todos los usuarios que se encuentren registrados en la base de datos, este método es de tipo *ActionResult*, se hace uso de la clase Servicio como se ve en el Código 2.10 y este a su vez hace uso de la clase repositorio para acceder a la información.

```
public ActionResult Index()
{
    return View(usuarioServicio.ObtenerTodosUsuarios());
}
```

Código 2.9 Método de acción Index en el Controlador

```
public List<Usuario> ObtenerTodosUsuarios()
{
    var listaUsuarios = _usuarioRepositorio.List.ToList();
    return listaUsuarios;
}
```

Código 2.10 Método ObtenerTodosUsuarios() en la clase de servicio

- **Acción Create – método Get**

Para la creación del método *get* de la acción *Create*, se devuelve una Vista en la que se van a ingresar los datos necesarios para la creación de un usuario, como se muestra en el Código 2.11, el método *CargarInformacion*() devuelve una lista de carreras, y el usuario deberá escoger una de ellas.

```
public ActionResult Create()
{
    CargarInformacion();
    return View();
}

2 referencias | wilson geovanny panjon quinde, Hace 28 días | 1 autor, 1 cambio
private void CargarInformacion()
{
    ViewBag.ListaCarreras = carreraServicio.ListaCarreras();
}
```

Código 2.11 Método Get de la acción Create

En el Código 2.12, se indicó que el método creado en la clase *UsuarioServicio* llama a la clase repositorio para las consultas a la base de datos y de esa manera retorna una lista de carreras para que el usuario pueda seleccionar la adecuada.

```
public SelectList ListaCarreras()
{
    var lista = _carreraRepositorio.List.Select(t => new { t.IDCarrera, t.NombreCarrera }).ToList();

    return new SelectList(lista, "IDCarrera", "NombreCarrera");
}
```

Código 2.12 Método en la Clase UsuarioServicio para obtener las Carreras

- **Acción Create – método Post**

En el Código 2.13, se muestra el método *Post* de la acción *Create*, que se utilizó para crear un nuevo *Usuario*, las validaciones se las realizó en la clase *UsuarioServicio* por eso se hace uso de la instancia de esta clase (línea 64).

Se verifica que los datos del Modelo que vienen desde la Vista sean válidos (línea 61), se obtiene la información del usuario que ingresa a la aplicación (línea 63), se hizo el llamado al método *CrearUsuario* (línea 64) si se cumple con las validaciones que se van a realizar en este método el valor que nos retornará será true, con ello ingresa y devuelve la Vista *Index* (línea 67) que es la que muestra la lista de Usuarios que se crean en el sistema.

Caso contrario, si el valor que retorna el método es falso muestra la Vista para indicar en donde está el error, por el cual el usuario no fue creado con éxito (línea 71).

```
57          [HttpPost]
58          [ValidateAntiForgeryToken]
            0 referencias | wilson geovanny panjon quinde, Hace 10 días | 1 autor, 10 cambios
59          public ActionResult Create([Bind(Include =
              "IDUsuario,IDCarrera,Nombre,Apellido,CorreoElectronico,Telefono,Cedula,Contrasena,Activo,RolesFacultad,UsuarioCreacion,
              UsuarioModificacion, FechaCreacion, FechaModificacion")] Usuario usuario)
60          {
61              if (ModelState.IsValid)
62              {
63                  Usuario usuarioLogin = _usuarioServicio.ObtenerUsuario(User.Identity.Name);
64                  var esValido = _usuarioServicio.CrearUsuario(ModelState,usuario, usuarioLogin.IDUsuario);
65                  if (esValido)
66                  {
67                      return RedirectToAction("Index");
68                  }
69              }
70              CargarInformacion();
71              return View(usuario);
72          }
```

Código 2.13 Método para crear usuario en el Controlador

Para leer la información de un registro en especial se hizo uso del método *ObtenerUsuario* que se encuentra en la clase *UsuarioServicio* en el Código 2.14, se observa la referencia a la clase *UsuarioRepositorio* para poder acceder a la información de la base de datos.

```
20          public Usuario ObtenerUsuario(int id)
21          {
22              return _usuarioRepositorio.FindById(id);
23          }
```

Código 2.14 Método para ver un usuario

El método *CrearUsuario* que se muestra en el Código 2.15, se encuentra en la clase *UsuarioServicio*, se valida que se ingrese el campo de la contraseña (líneas de la 35 a la 37), se crea la encriptacion de la contraseña (línea 42) ya que no se puede guardar en texto plano en la base de datos por seguridad, se muestra cómo se fueron asignando cada uno de los datos que el usuario ingreso a la aplicación (líneas 43 a la 58).

En el Código 2.16, se observan las validaciones de negocio, para este caso se recibe como uno de los parámetros el número de cédula y se van a mostrar mensajes de error en el

caso que el número de cédula no sea válido o que ya haya un usuario registrado con el mismo número, este método de verificacion de cédula se encuentra en el Código 2.17, donde se encuentra el algoritmo que se utiliza en el pais para verificar la validez de una cédula Ecuatoriana.

Si el Modelo es válido es decir toda la información es correcta, se va a llamar a la clase *repositorio* para añadir al nuevo usuario (línea 65).

```
33          public bool CrearUsuario(ModelStateDictionary modelState, Usuario usuario, int usuarioLogeado)
34          {
35              if (string.IsNullOrWhiteSpace(usuario.Contrasena))
36              {
37                  modelState.AddModelError("Contrasena", "La Contraseña es requerida.");
38              }
39              //Validaciones del cliente
40              if (modelState.IsValid)
41              {
42                  var hashPassword = SHAEncriptacion.HashPassword(usuario.Contrasena);
43                  Usuario usuarioNuevo = new Usuario
44                  {
45                      IDCarrera = usuario.IDCarrera,
46                      Nombre = usuario.Nombre,
47                      Apellido = usuario.Apellido,
48                      CorreoElectronico = usuario.CorreoElectronico,
49                      Telefono = usuario.Telefono,
50                      Cedula = usuario.Cedula,
51                      Contrasena = hashPassword.Item2,
52                      Activo = usuario.Activo,
53                      RolesFacultad = usuario.RolesFacultad,
54                      UsuarioCreacion = usuarioLogeado,
55                      UsuarioModificacion = usuarioLogeado,
56                      FechaCreacion = DateTime.Now,
57                      FechaModificacion = DateTime.Now
58                  };
59
60                  //Validaciones del negocio
61                  ValidacionNegocio(modelState, usuarioNuevo);
62
63                  if (modelState.IsValid)
64                  {
65                      _usuarioRepositorio.Add(usuarioNuevo);
66                      return true;
67                  }
68              }
69              return false;
70          }
```

Código 2.15 Creación de usuario en la clase UsuarioServicio

```
private void ValidacionNegocio(ModelStateDictionary modelState, Usuario record)
{
    bool cedulaEsValida = VerificaCedula(record.Cedula);
    if (!cedulaEsValida)
    {
        modelState.AddModelError("Cedula", "El numero de cedula no es valido.");
        return;
    }

    bool cedulaExiste = _usuarioRepositorio.List.Where(t => t.Cedula == record.Cedula).Any();
    if (cedulaExiste)
        modelState.AddModelError("Cedula", "El usuario con este numero de cedula ya existe.");
}
```

Código 2.16 Validaciones de negocio

```
private bool VerificaCedula(string validarCedula)
{
    int aux = 0, par = 0, impar = 0, verifi;
    for (int i = 0; i < 9; i += 2)
    {
        aux = 2 * int.Parse(validarCedula[i].ToString());
        if (aux > 9)
            aux -= 9;
        par += aux;
    }
    for (int i = 1; i < 9; i += 2)
    {
        impar += int.Parse(validarCedula[i].ToString());
    }

    aux = par + impar;
    if (aux % 10 != 0)
    {
        verifi = 10 - (aux % 10);
    }
    else
        verifi = 0;
    if (verifi == int.Parse(validarCedula[9].ToString()))
        return true;
    else
        return false;
}
```

Código 2.17 Verificación de existencia de cédula

- **Acción Edit – método GET**

El método *Get* de la acción *Edit* como se muestra en el Código 2.18, recibe como párametro un *id* con el cual se procede a ejecutar la consulta que devuelve el registro correspondiente, la información será desplegada en la Vista y de acuerdo a la Vista se mostrará qué información se podrá editar.

- **Acción Edit – método POST**

Cuando se usa el método *Post* como se muestra en el Código 2.19, significa que se va a obtener toda la información que se haya ingresado a la Vista, se comprueba que la información de los campos que se editó sea correcta (línea 90), se hace uso de la clase *UsuarioServicio* para actualizar la información del usuario (línea 92).

Se usó la misma acción para editar información del usuario ya sea que cada usuario edite su propia información o que el administrador del sistema edite esta información, se colocan sentencias que indiquen que tipo de Vista desplegar dependiendo del rol de usuario, si un usuario Estudiante está editando su información, cuando este proceso concluya se lo redirigirá hacia la Vista Index, pero cuando el usuario con el rol de Coordinador que es el unico que puede editar esta información haga algun cambio, a él si se le debe mostrar toda la lista de usuarios a los cuales el puede editar su información (línea 94 a 98).

```
public ActionResult Edit(int id)
{
    Usuario usuario = usuarioServicio.ObtenerUsuario(id);
    return View(usuario);
}
```

Código 2.18 Acción Edit método GET

```
86      [HttpPost]
87      [ValidateAntiForgeryToken]
        0 referencias | wilson geovanny panjon quinde, Hace 4 días | 1 autor, 7 cambios
88      public ActionResult Edit( Usuario usuario)
89      {
90          if (ModelState.IsValid)
91          {
92              _usuarioServicio.ActualizarUser(usuario, User.Identity.Name);
93              Usuario user = _usuarioServicio.ObtenerUsuario(User.Identity.Name);
94              if (user.RolesFacultad != RolesFacultad.Coordinador)
95              {
96                  return RedirectToAction("Profile", "Usuarios", new { cedula = User.Identity.Name });
97              }
98              return RedirectToAction("Index");
99          }
100         CargarInformacion();
101         return View(usuario);
102     }
```

Código 2.19 Acción Edit método POST

- **Acción Eliminar – método GET**

El método *Get* de la acción *Delete* como se muestra en el Código 2.20, es similar al método *Get* de la acción *Edit*, ya que se encargará de desplegar en la Vista la información de un usuario en particular, por eso tiene como parámetro de entrada un *id* la diferencia es en la Vista ya que la información será solo de lectura.

```
public ActionResult Delete(int id)
{
    Usuario usuario = usuarioServicio.ObtenerUsuario(id);
    return View(usuario);
}
```

Código 2.20 Acción Delete método GET

- **Acción Delete – método POST**

Para la acción *Eliminar* un registro como se muestra en el Código 2.21, se necesita de un parámetro *id* que indique qué usuario fue seleccionado, de esta manera se busca el registro en la base de datos y se procede a eliminarlo.

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias | wilson geovanny panjon quinde, Hace 57 días | 1 autor, 3 cambios
public ActionResult DeleteConfirmed(int id)
{
    Usuario usuario = usuarioServicio.ObtenerUsuario(id);
    usuarioServicio.EliminarUsuario(usuario);
    return RedirectToAction("Index");
}
```

Código 2.21 Método para eliminar usuario en el Controlador

- **Acción Details – método GET**

La acción *Details* como se muestra en Código 2.22, recibe como parámetro el *id* del usuario seleccionado del cual se quiere obtener la información.

```
public ActionResult Details(int id)
{
    Usuario usuario = usuarioServicio.ObtenerUsuario(id);
    return View(usuario);
}
```

Código 2.22 Acción Details método GET

Las acciones *Edit*, *Delete* y *Details* difieren solo de la forma en que los datos van a ser mostrados al usuario, ya sea que se le permita modificar estos datos o que solo sean de lectura.

Estas acciones son comunes para todos los Controladores la diferencia radica en los Modelos a utilizar.

Entre las tareas que se requiere para este *Sprint* también se tiene la opción de *Login* como se muestra en el Código 2.23, para ello se creó el Controlador *LoginController*, que envía los datos que se han ingresado en la Vista de *login* y se valida que la cédula y la contraseña sean correctas a la hora de ingresar (línea 32), si esto es correcto se enviará al usuario a la Vista *Index* del Controlador *HomeController*.

```
29          [HttpPost]
            0 referencias | wilson geovanny panjon quinde, Hace 11 días | 1 autor, 6 cambios
30          public ActionResult Index(Login record)
31          {
32              bool result = _usuarioServicio.ValidarLogin(ModelState, record);
33              if (result)
34              {
35                  var user = _usuarioServicio.ObtenerUsuario(record.Cedula);
36                  FormsAuthentication.SetAuthCookie(record.Cedula, record.Recordarme);
37                  var authTicket = new FormsAuthenticationTicket(1, user.Cedula, DateTime.Now, DateTime.Now.AddMinutes(20), false,
                      user.RolesFacultad.ToString());
38                  string encryptedTicket = FormsAuthentication.Encrypt(authTicket);
39                  var authCookie = new HttpCookie(FormsAuthentication.FormsCookieName, encryptedTicket);
40                  HttpContext.Response.Cookies.Add(authCookie);
41                  return RedirectToAction("Index", "Home");
42              }
43              return View(new Login());
44          }
```

Código 2.23 Login del usuario

Una de las opciones que también se les presenta a los usuarios es el cambio de contraseña como se observa en el Código 2.24, con el método *Get* se obtiene el usuario que ha ingresado a la aplicación (línea 25), y se obtienen solo ciertos campos del usuario como cédula, nombre y apellido para mostrarlos en la Vista (línea 26).

En el método *Post* se va a obtener la contraseña nueva y la comprobación del cambio, se valida que los datos sean correctos y se proceda a actualizar la contraseña (línea 34).

```
23          public ActionResult CambiarContrasena()
24          {
25              Usuario usuario = _usuarioServicio.ObtenerUsuario(User.Identity.Name);
26              CambioContrasena cambioContrasena = new CambioContrasena { Cedula = usuario.Cedula, Nombre = usuario.Nombre, Apellido =
                    usuario.Apellido };
27              return View(cambioContrasena);
28          }
29
30          [HttpPost]
31          [ValidateAntiForgeryToken]
            0 referencias | wilson geovanny panjon quinde, Hace 11 días | 1 autor, 3 cambios
32          public ActionResult CambiarContrasena([Bind(Include = "Cedula, Nombre, Apellido, Contrasena, Contrasena2")] CambioContrasena
                cambioContrasena)
33          {
34              if (_usuarioServicio.ActualizarContrasena(ModelState, cambioContrasena))
35              {
36                  return RedirectToAction("Index");
37              }
38
39              return View(cambioContrasena);
40          }
```

Código 2.24 Cambio de contraseña

2.2.7 SPRINT 2 – GESTIÓN DE PLANTILLAS

Para la gestión de plantillas se realizaron las acciones ya detalladas anteriormente que son *Index, Create, Edit, Delete* y *Details* la diferencia fue el Modelo para acceder como se explicó.

Una de las clases que se debió añadir a Modelo es la clase *PlantillaRequisitoViewModel*: esta clase sirve solo para mostrar información al usuario de una Vista que tiene información de dos Modelos que son *Plantilla* y *Requisito*, es decir haciendo uso de esta clase se puede ver el tipo de requisitos que tiene cierta plantilla, esto sirve para solicitudes con plantilla definida.

La clase *PlantillaRequisitoViewModel* es utilizada en el Código 2.25, en base al *id* de la plantilla seleccionada se obtiene la información de la *Plantilla* y de los *Requisitos* relacionados con esa *Plantilla*, todo esto se mostrará en la Vista *Details* del Controlador *PlantillasController*.

```
public ActionResult Details(int id)
{
    PlantillaRequisitoViewModel mymodel = new PlantillaRequisitoViewModel();
    mymodel.Plantilla = plantillaServicio.ObtenerPlantilla(id);
    mymodel.Requisitos = requisitoServicio.RequisitosPorPlantilla(id);
    return View(mymodel);

}
```

Código 2.25 Ejemplo de utilización de la clase PlantillaRequisitoViewModel

2.2.8 SPRINT 3 – GESTIÓN DE SOLICITUDES

Para la gestión de solicitudes por parte del usuario con el rol *Estudiante* se crearon los métodos respectivos *Index, Create, Details*. Este usuario no tendrá la acción de Editar una solicitud ni de eliminarla, ya que una vez creada ésta es enviada al *Secretario*.

Entre los métodos que se diferencian de la solicitud se encuentran los siguientes:

El método *CargarSolicitud* se muestra en el Código 2.26, este método carga la información de la plantilla y del Estudiante para crear el detalle de la solicitud. Se obtiene la información del usuario que ingresó a la aplicación (línea 87), se genera el código de barras que se va añadir a la solicitud (línea 88), los valores precargados que tendrá la solicitud se envían en la variable *solicitud* (línea 89). Los métodos para *GenerarSolicitud* y para *LlenarSolicitud* de las (lineas 91 y 92) se explican más adelante. La variable *result* devuelve el detalle de

la solicitud y el código de barras generado (línea 94). El valor de retorno es un objeto de tipo JsonResult (línea 96).

```
85          public JsonResult CargarSolicitud(int idPlantilla)
86          {
87              Usuario usuario = _usuarioServicio.ObtenerUsuario(User.Identity.Name);
88              var codigoBarras = CodigoBarras.GenerarCodigoBarras(CodigoBarras.CrearCodigoUnico());
89              var solicitud = new Solicitud { IDPlantilla = idPlantilla, UsuarioCreacion = usuario.IDUsuario, FechaCreacion =
                    DateTime.Now, CodigoBarras = codigoBarras };
90
91              _solicitudServicio.GenerarSolicitud(solicitud);
92              _solicitudServicio.LlenarSolicitud(solicitud);
93
94              var result = new { detalle = solicitud.DetalleSolicitud, codigo = codigoBarras };
95
96              return new JsonResult() { Data = result, JsonRequestBehavior = JsonRequestBehavior.AllowGet };
97          }
```

Código 2.26 Método para cargar detalles de una solicitud

Como se muestra en el Código 2.27, el método *GenerarSolicitud* llama a la clase *EstructuraPlatilla* que crea la estructura de la plantilla (línea 253) que está en formato XML como se muestra en el Código 2.28, se observa que esa información se va agregar al campo *DetalleSolicitud* de la tabla Solicitud (línea 255).

```
250         public Solicitud GenerarSolicitud(Solicitud solicitud)
251         {
252             string _detalleSolicitud = string.Empty;
253             _detalleSolicitud = EstructuraPlantilla.ObtenerEstructuraPlantilla();
254
255             solicitud.DetalleSolicitud = _detalleSolicitud;
256             return solicitud;
257         }
```

Código 2.27 Método para generar solicitud

Para proceder a llenar la solicitud como se muestra en el Código 2.29, se requiere reemplazar las etiquetas del código XML con información del Estudiante (línea 261), información de la plantilla seleccionada (línea 262), información de la autoridad a la que va dirigida la solicitud (línea 263), se obtiene información de la carrera de la autoridad (línea 264) y se obtiene información de la carrera (línea 265).

Toda esta información va a ser reemplazada de una manera fácil ya que se tiene etiquetas que son fáciles de entender, todo esto gracias a XML, y finalmente se va a devolver el detalle de la solicitud ya con los datos cargados (línea 281).

```
public static string ObtenerEstructuraPlantilla()
{
    StringBuilder estructuraPlantilla = new StringBuilder();
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Quito, " + "<Fecha>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Ingeniero");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<AutoridadNombre>");
    estructuraPlantilla.Append(" ");
    estructuraPlantilla.Append("<AutoridadApellido>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<AutoridadCargo>");
    estructuraPlantilla.Append(" ");
    estructuraPlantilla.Append("de la");
    estructuraPlantilla.Append(" ");
    estructuraPlantilla.Append("<AutoridadFacultad>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Presente");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Yo, <Nombre> <Apellido>, con C.C <Cedula> ," +"estudiante de la Carrera <Carrera>, " +
                "solicito se me otorgue: <Solicitud>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Por la atención favorable a esta solicitud, anticipo mi agradecimiento.");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Atentamente");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Correo: <Email>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("Telefono: <Telefono>");
    estructuraPlantilla.Append("<br />");
    estructuraPlantilla.Append("<CodigoBarras>");
    return estructuraPlantilla.ToString();
}
```

Código 2.28 Formato XML de la plantilla

```
public string LlenarSolicitud(Solicitud solicitud)
{
    var infoEstudiante = _usuarioServicio.ObtenerUsuario(solicitud.UsuarioCreacion);
    var infoPlantilla = _plantillaServicio.ObtenerPlantilla(solicitud.IDPlantilla.Value);
    var infoAutoridad = _usuarioServicio.ObtenerUsuario(infoPlantilla.IDUsuario);
    var FacultadAutoridad = _solicitudRepositorio.FacultadPorId(infoAutoridad.IDUsuario);
    var infoCarrera = _usuarioServicio.ObtenerCarreraPorUsuario(infoEstudiante.IDCarrera);

    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Fecha>", solicitud.FechaCreacion.ToString());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<AutoridadNombre>", infoAutoridad.Nombre.ToString());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<AutoridadApellido>", infoAutoridad.Apellido.ToString());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<AutoridadCargo>", infoAutoridad.RolesFacultad.ToString
      ());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<AutoridadFacultad>",
      FacultadAutoridad.NombreFacultad.ToString());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Nombre>", infoEstudiante.Nombre);
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Apellido>", infoEstudiante.Apellido);
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Cedula>", infoEstudiante.Cedula);
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Carrera>", infoCarrera.NombreCarrera.ToString());
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Solicitud>", infoPlantilla.TipoSolicitud);
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<CodigoBarras>", $"<br /><img
      src='{solicitud.CodigoBarras}' />");
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Email>", infoEstudiante.CorreoElectronico);
    solicitud.DetalleSolicitud = solicitud.DetalleSolicitud.Replace("<Telefono>", infoEstudiante.Telefono);

    return solicitud.DetalleSolicitud;
}
```

Código 2.29 Método para llenar el detalle de la solicitud

Para cargar los requisitos de una plantilla específica se envía el identificador único para obtener el resultado esperado. En el Código 2.30, se muestra que se recibe como parámetro el *idPlantilla* para asociar los requisitos a esa plantilla; en el Código 2.31, se muestra la consulta que se hace a la base de datos para obtener esta información.

```
public JsonResult CargarRequisitos(int idPlantilla)
{
    var listaRequisitos = requisitoServicio.RequisitosPorPlantilla(idPlantilla).Select(t => t.Descripcion);

    return new JsonResult() { Data = listaRequisitos, JsonRequestBehavior = JsonRequestBehavior.AllowGet };
}
```

Código 2.30 Método para cargar los requisitos

```
public List<Requisito> Requisitos(int id)
{
    var result = (from p in _context.Plantilla
                  join p_r in _context.PlantillaRequisito
                  on p.IDPlantilla equals p_r.IDPlantilla
                  join r in _context.Requisito
                  on p_r.IDRequisito equals r.IDRequisito
                  where p.IDPlantilla == id
                  select r).ToList();
    return result;
}
```

Código 2.31 Consulta para saber los requisitos de una plantilla

En el Código 2.32, se muestran los métodos necesarios para cargar la información del detalle de la solicitud y los requisitos respectivos.

```
public JsonResult CargarDetalleSolicitud(int idSolicitud)
{
    var solicitud = solicitudServicio.ObtenerSolicitud(idSolicitud);
    return new JsonResult() { Data = solicitud.DetalleSolicitud, JsonRequestBehavior = JsonRequestBehavior.AllowGet };
}

0 referencias | wilson geovanny panjon quinde, Hace 33 días | 1 autor, 1 cambio
public JsonResult CargarListaRequisitos(int idSolicitud)
{
    var solicitud = solicitudServicio.ObtenerSolicitud(idSolicitud);
    var listaRequisitos = requisitoServicio.RequisitosPorPlantilla(solicitud.IDPlantilla.Value).Select(t => t.Descripcion);

    return new JsonResult() { Data = listaRequisitos, JsonRequestBehavior = JsonRequestBehavior.AllowGet };
}
```

Código 2.32 Métodos para cargar el detalle y los requisitos de la solicitud

Para la creación de una solicitud sin plantilla definida se hace uso de los pasos similares a los que se usan para crear una solicitud con plantilla, con la diferencia de que no se tiene un *idPlantilla* ni los requisitos asociados a ese *id*. Se creó un método XML para la solicitud sin plantilla, como se muestra en el Código 2.33, que ya no incluye la información de la plantilla, y en su lugar le permitirá al estudiante ingresar el pedido que el desee.

```
11      public static string ObtenerEstructuraSinPlantilla()
12      {
13          StringBuilder estructuraPlantilla = new StringBuilder();
14          estructuraPlantilla.Append("<br />");
15          estructuraPlantilla.Append("Quito, " + "<Fecha>");
16          estructuraPlantilla.Append("<br />");
17          estructuraPlantilla.Append("Ingeniero");
18          estructuraPlantilla.Append("<br />");
19          estructuraPlantilla.Append("<AutoridadNombre>");
20          estructuraPlantilla.Append(" ");
21          estructuraPlantilla.Append("<AutoridadApellido>");
22          estructuraPlantilla.Append("<br />");
23          estructuraPlantilla.Append("<AutoridadCargo>");
24          estructuraPlantilla.Append(" ");
25          estructuraPlantilla.Append("de la");
26          estructuraPlantilla.Append(" ");
27          estructuraPlantilla.Append("<AutoridadFacultad>");
28          estructuraPlantilla.Append("<br />");
29          estructuraPlantilla.Append("Presente");
30          estructuraPlantilla.Append("<br />");
31          estructuraPlantilla.Append("Yo, <Nombre> <Apellido>, con C.C <Cedula> ," +
32                                     "estudiante de la Carrera <Carrera>, " +
33                                     "solicito ");
34          estructuraPlantilla.Append("<br />");
35          estructuraPlantilla.Append("Por la atención favorable a esta solicitud, anticipo mi agradecimiento.");
36          estructuraPlantilla.Append("<br />");
37          estructuraPlantilla.Append("Atentamente");
38          estructuraPlantilla.Append("<br />");
39          estructuraPlantilla.Append("Correo: <Email>");
40          estructuraPlantilla.Append("<br />");
41          estructuraPlantilla.Append("Telefono: <Telefono>");
42          estructuraPlantilla.Append("<br />");
43          estructuraPlantilla.Append("<CodigoBarras>");
44          return estructuraPlantilla.ToString();
45      }
```

Código 2.33 Método XML para solicitud sin plantilla

La clase *SolicitudSinPlantilla* sirve para gestionar la información de las solicitudes que el usuario con el rol de Estudiante quiera redactar con sus propias palabras. Se utiliza la misma tabla Solicitud pero con la diferencia que el valor de IDPlantilla y el PathRequisitos serán nulos, ya que en esta solicitud no se le permite seleccionar una plantilla sino crear un tipo de solicitud propia del Estudiante

Para proceder a dar trámite a una solicitud se creó un nuevo Controlador llamado *AdminSolicitudController.* En el Código 2.34, se muestra el método *TramitarSolicitud* que se encarga de actualizar los campos de la tabla Solicitud como son: Estado (Aceptado, Aprobado o Rechazado) según haya seleccionado la persona que esté dando el trámite a la solicitud (línea 208), Comentario con alguna indicación para el Estudiante (línea 207), la fecha y el usuario de edición que son campos de auditoría que informarán quien modificó el estado y comentario de una solicitud y en qué fecha se realizaron los cambios.

```
200         public bool TramitarSolicitud(ModelStateDictionary modelState, Solicitud solicitud, HttpPostedFileBase file, string cedula)
201         {
202             var respuesta = ValidarTramite(modelState, solicitud, file);
203             if (respuesta && modelState.IsValid)
204             {
205                 var idUsuario = _usuarioServicio.ObtenerUsuario(cedula).IDUsuario;
206                 var record = ObtenerSolicitud(solicitud.IDSolicitud);
207                 record.Comentario = solicitud.Comentario;
208                 record.Estado = solicitud.Estado;
209                 record.FechaEdicion = DateTime.Now;
210                 record.UsuarioModificacion = idUsuario;
211                 record.PathRespuestaArchivos = solicitud.PathRespuestaArchivos;
212                 _solicitudRepositorio.Update(record);
213 
214                 if (!string.IsNullOrWhiteSpace(record.PathRespuestaArchivos))
215                 {
216                     SaveArchivosSolicitud(file, record,true);
217                 }
218 
219                 EnviodeCorreo(record.UsuarioCreacion);
220                 return true;
221             }
222             return false;
223         }
```

Código 2.34 Trámite a una solicitud

2.2.9 SPRINT 4 – GESTIÓN DE RESPUESTA

En el Código 2.34, se muestra el método para enviar un correo electrónico ya que se está actualizando la información del estado de una solicitud (línea 219); el método de envío del correo se muestra en el Código 2.35, se obtiene la información del estudiante que ha generado la solicitud (línea 123) del cual se va a obtener la información del correo electrónico para el envío de su respuesta, la información de la solicitud (línea 125) para conocer su estado, y se ingresa el tema del correo que se desea que se muestre (línea 127) al igual que el mensaje (línea 128).

```
121         public void EnviodeCorreo(int cedula, Solicitud solicitud)
122         {
123             var usuarioEstudiante = _usuarioServicio.ObtenerUsuario(cedula);
124             var usuarioSecretario = _usuarioServicio.ObtenerSecretario();
125             var infoSolicitud = ObtenerSolicitud(solicitud.Id);
126             var carrera = _solicitudRepositorio.CarreraPorId(usuarioEstudiante.Id);
127             var subject = "EPN Gestión de solicitudes";
128             var message = "";
129             if (solicitud.IDPlantilla != null)
130             {
131                 var tipoSolicitud = _plantillaServicio.ObtenerPlantilla
                      (infoSolicitud.IDPlantilla.Value);
132                 message = "Estimado " + usuarioEstudiante.Nombre + "<br /> <br /> Su solicitud con
                      asunto: " + tipoSolicitud.TipoSolicitud + " ha sido actualizada al estado: "+
                      infoSolicitud.Estado+". Por favor ingrese al siguiente LINK:
                      gestionsolicitudes.azurewebsites.net/ para mayor información. <br /> Gracias por
                      usar la aplicación, por favor no responda a este correo. <br /><br /> Saludos,<br/
                      > Coordinación de " + carrera.NombreCarrera;
133             }
134             else
135             {
136                 message = "Estimado " + usuarioEstudiante.Nombre + "<br /> <br /> Su solicitud ha
                      sido actualizada al estado: "+infoSolicitud.Estado+". Por favor ingrese al
                      siguiente LINK: gestionsolicitudes.azurewebsites.net/ para mayor información. <br /
                      > Gracias por usar la aplicación, por favor no responda a este correo. <br /><br />
                       Saludos,<br/ > Coordinación de " + carrera.NombreCarrera;
137             }
138
139             Email.SendEmailOU(usuarioEstudiante.CorreoElectronico, subject, message,
                  usuarioEstudiante.CorreoElectronico);
140         }
```

Código 2.35 Método para el envío de correo electrónico

El Código 2.34, indica que se va a subir un archivo de respuesta a la solicitud del estudiante (línea 216). En el Código 2.36 se muestra que cuando el parámetro *esTramite* está en verdadero se permite subir un archivo de respuesta, este archivo debe ser en formato .zip o .rar si se desea ampliar el formato de los archivos de respuesta se deberá cambiar el formato en el Código 2.37 (línea 368).

```
321         private void SaveArchivosSolicitud(HttpPostedFileBase file, Solicitud solicitud, bool esTramite)
322         {
323             if (esTramite)
324             {
325                 Archivos.GuardarArchivo(file, solicitud.PathRespuestaArchivos, false);
326             }
327             else
328             {
329                 if (file != null)
330                 {
331                     Archivos.GuardarArchivo(file, solicitud.PathRequisitos,true);
332                 }
333                 Archivos.GuardarArchivoPDF(solicitud.DetalleSolicitud, solicitud.PathPDF);
334             }
335
336         }
```

Código 2.36 Método para cargar archivos de respuesta

```
360         private string ValidarArchivo(HttpPostedFileBase file, ModelStateDictionary modelState,string nombreCampo)
361         {
362             if (file == null)
363             {
364                 modelState.AddModelError(nombreCampo, "La carpeta zip o rar con los requisitos es requerido");
365                 return string.Empty;
366             }
367             var extensionArchivo = Archivos.ObtenerExtensionArchivo(file);
368             if (!(extensionArchivo == ".zip" || extensionArchivo == ".rar"))
369             {
370                 modelState.AddModelError(nombreCampo, "Las extensiones permitidas son zip y rar.");
371                 return string.Empty;
372             }
373 
374             var nombreArchivo = Archivos.CrearNombreArchivo(extensionArchivo);
375             return nombreArchivo;
376         }
```

Código 2.37 Validaciones del archivo que se carga

El Código 2.38, permite visualizar estadísticas de las solicitudes tramitadas, se crean variables para obtener las solicitudes que son Aceptadas, aprobadas y rechazadas (líneas 82, 83 y 84), se muestra la forma de la figura que se va a desplegar en este caso la forma es de “Pie” (línea 91).

```
80          public ActionResult Grafica()
81          {
82              int Aprobado = _solicitudRepositorio.List.Where(t => t.Estado == EstadoSolicitud.Aprobado.GetDescription()).Count();
83              int Rechazado = _solicitudRepositorio.List.Where(t => t.Estado == EstadoSolicitud.Rechazado.GetDescription()).Count();
84              int Aceptado = _solicitudRepositorio.List.Where(t => t.Estado == EstadoSolicitud.Aceptado.GetDescription()).Count();
85 
86              new Chart(width: 800, height: 200, theme: ChartTheme.Green)
87                  .AddLegend()
88                  .AddTitle("Tipos de Solicitudes")
89              .AddSeries(
90                  name: "Chart",
91                  chartType: "Pie",
92                  xValue: new[] { "Aprobado", "Rechazado", "Aceptado" },
93                  yValues: new[] { Aprobado, Rechazado, Aceptado }
94                  ).Write("png");
95              return null;
96          }
```

Código 2.38 Método para graficar estadísticas

2.2.10 SPRINT 5 – PUBLICACIÓN DE LA APLICACIÓN

El servidor IIS sirve para publicar sitios web, en este caso la aplicación MVC C# con Microsoft, se identificó durante el proceso que Microsoft provee Azure que es un servicio en línea y con el que la Universidad tiene convenio, de esta manera se tomó ventaja de esta herramienta para publicar el sitio, ya no en IIS ya que hacía más lenta la computadora personal sino que se publicó directamente en un sitio web público, esa fue la ventaja de Azure que es visible en el internet usando la cuenta de estudiante y a su vez la velocidad de operación es mayor.

La plataforma Microsoft Azure provee servicios en la nube para construir, desplegar y administrar aplicaciones desde cualquier lugar. Para poder publicar una aplicación web se requiere tener una cuenta de usuario en esta plataforma.

La aplicación web y la base de datos serán alojadas en Azure.

Para configurar el recurso Azure App Service se siguen los pasos como: App Services → Add → y se configura los campos como se muestra en la Figura 2.20, el nombre debe ser único.

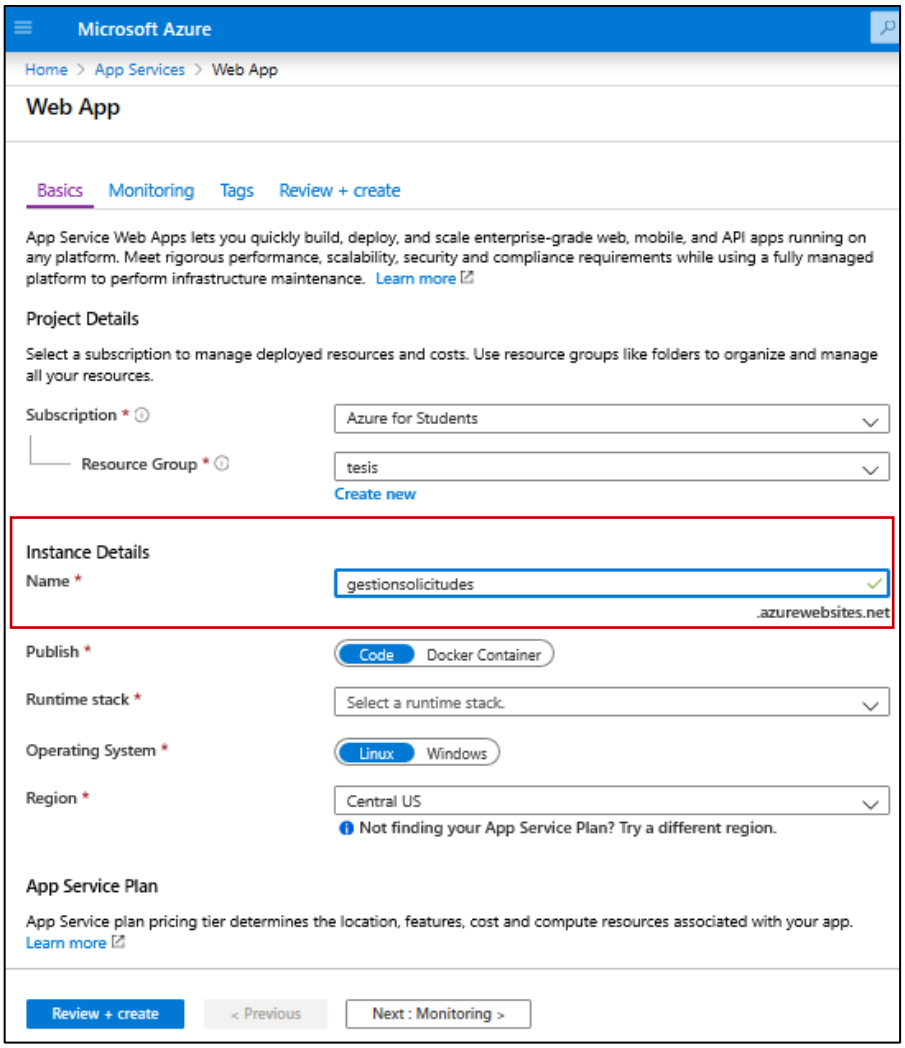


Figura 2.20 Creación del perfil de publicación

Una vez creado el perfil de publicación se procede a publicar el proyecto desde Visual Studio como se muestra en la Figura 2.21. Con el perfil descargado se da click en la opción Seleccionar existente → Importar perfil como se muestra en la Figura 2.22 y luego dar *click* en publicar.

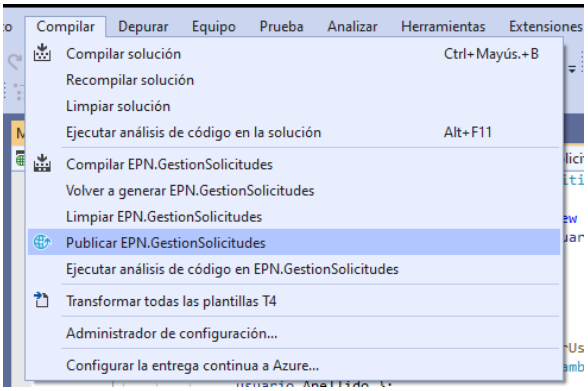


Figura 2.21 Configuración del proyecto desde Visual Studio

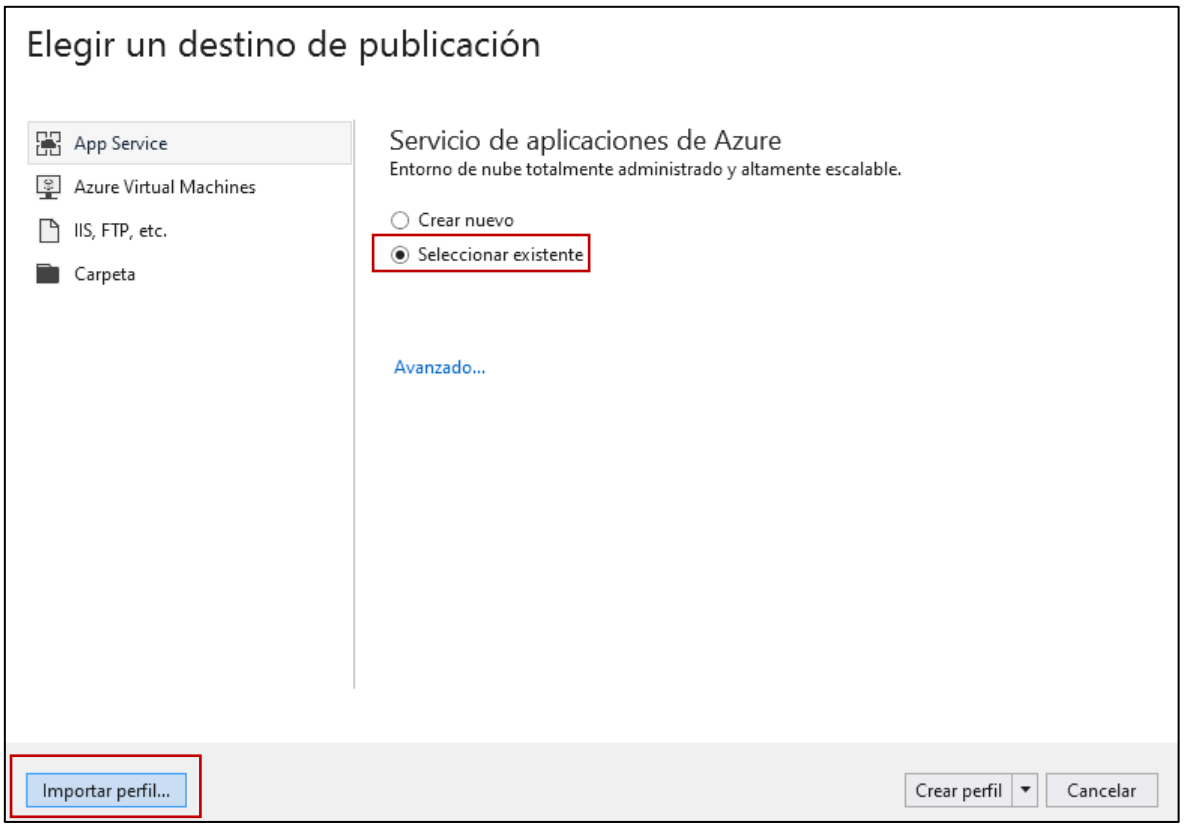


Figura 2.22 Publicación de la aplicación web

Se crea el servidor de base de datos que debe tener un nombre único, y las credenciales de acceso como se muestra en la Figura 2.23 y después se asigna la base de datos al servidor como se muestra en la Figura 2.24.

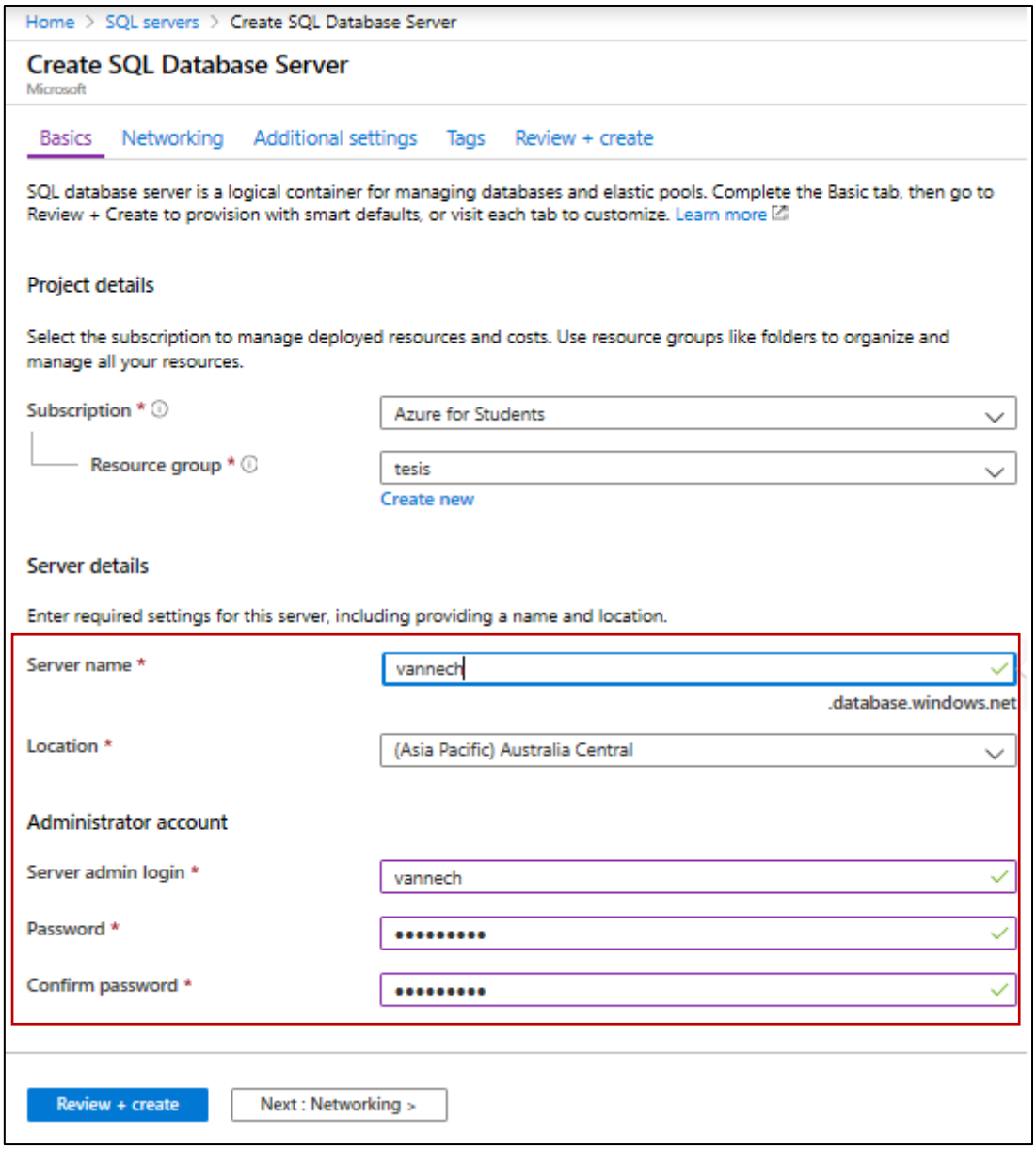


Figura 2.23 Creación del servidor de base de datos

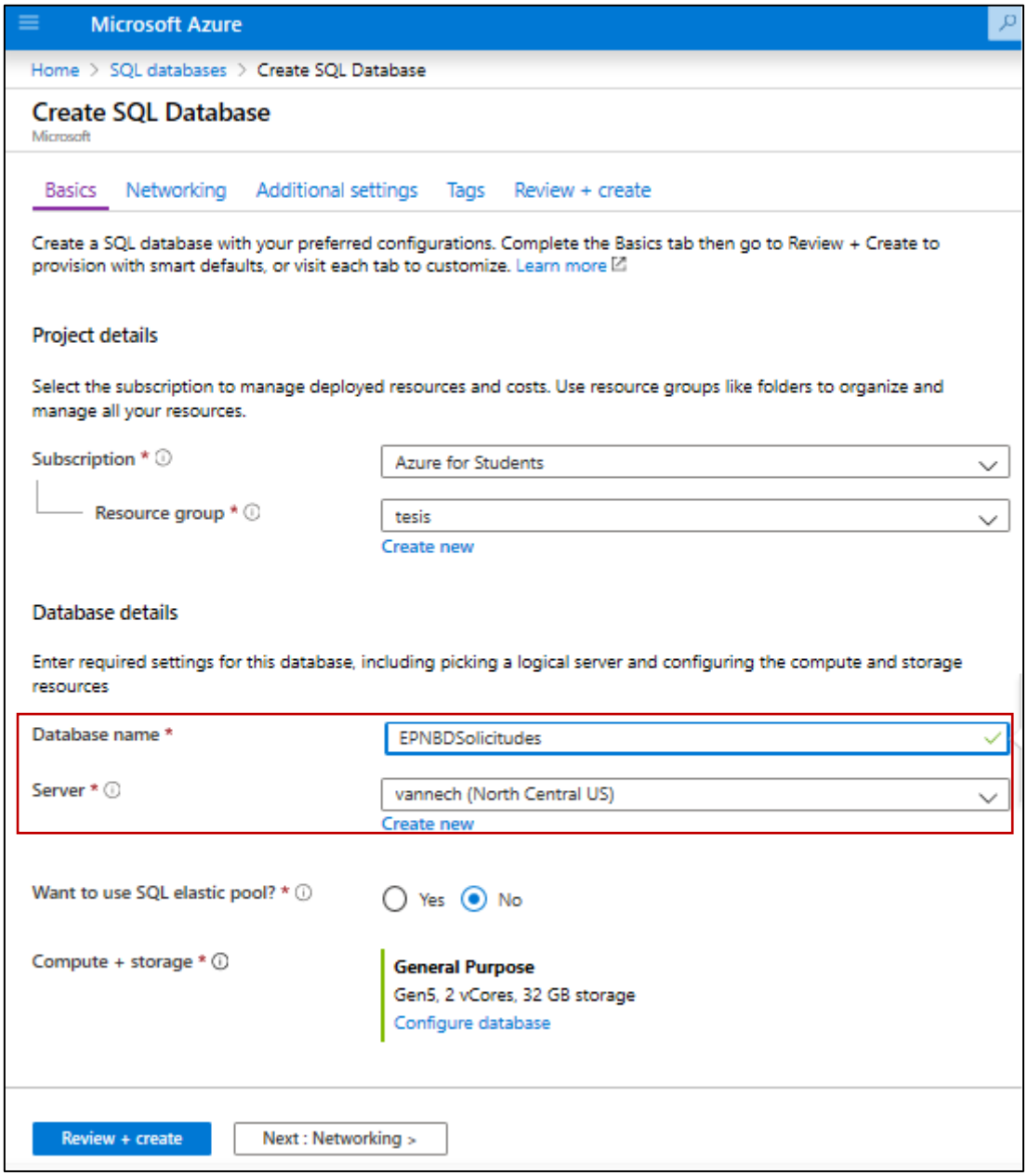


Figura 2.24 Creación de la base de datos

Con la configuración lista se procede a conectarse al servidor con las credenciales respectivas desde SQL Server Management Studio así como se muestra en la Figura 2.25.

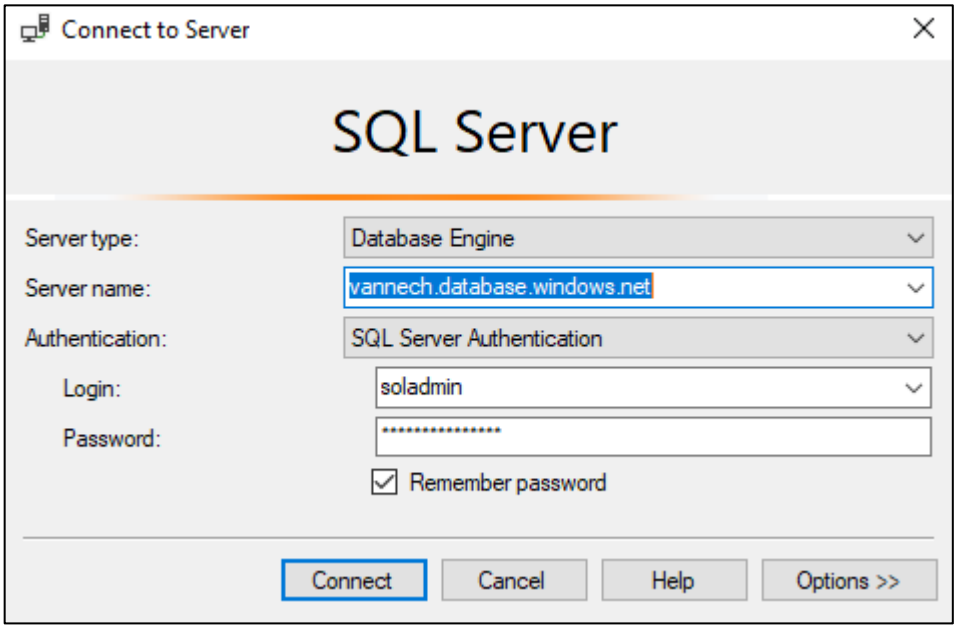


Figura 2.25 Conexión al servidor desde SQL server

Para acceder a la aplicación se debe ingresar la dirección https://gestionsolicitudes.azurewebsites.net/ y se observa la aplicación publicada como se muestra en Figura 2.26.

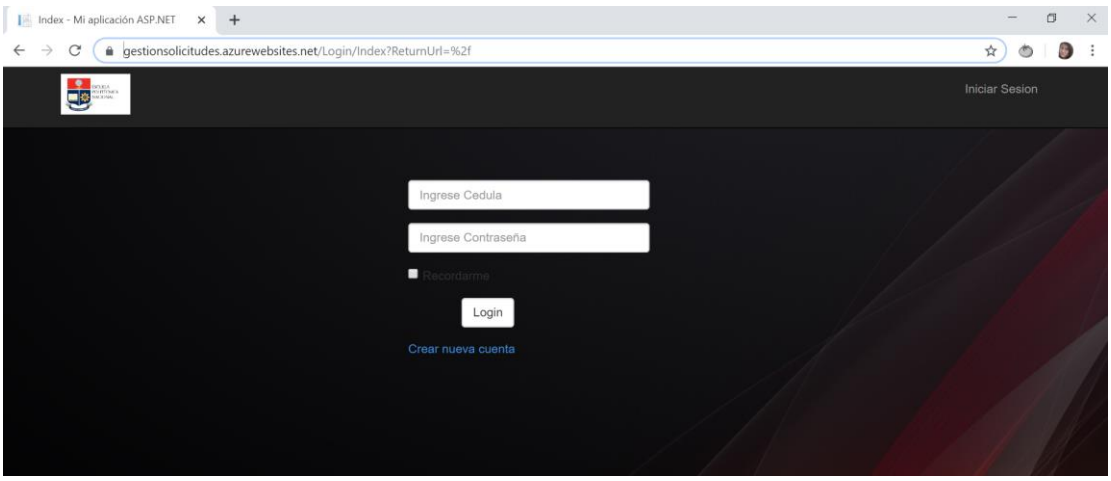


Figura 2.26 Publicación de la Aplicación Web

2.2.11 SKETCHES

Con base a los requerimientos obtenidos se presentan los siguientes *sketches* que representan las interfaces que se desarrollaron.

En la Figura 2.27 se presenta la página principal de Login, con la opción para registrarse e ingresar a la aplicación, las credenciales para acceder son: el número de cédula y la respectiva contraseña.

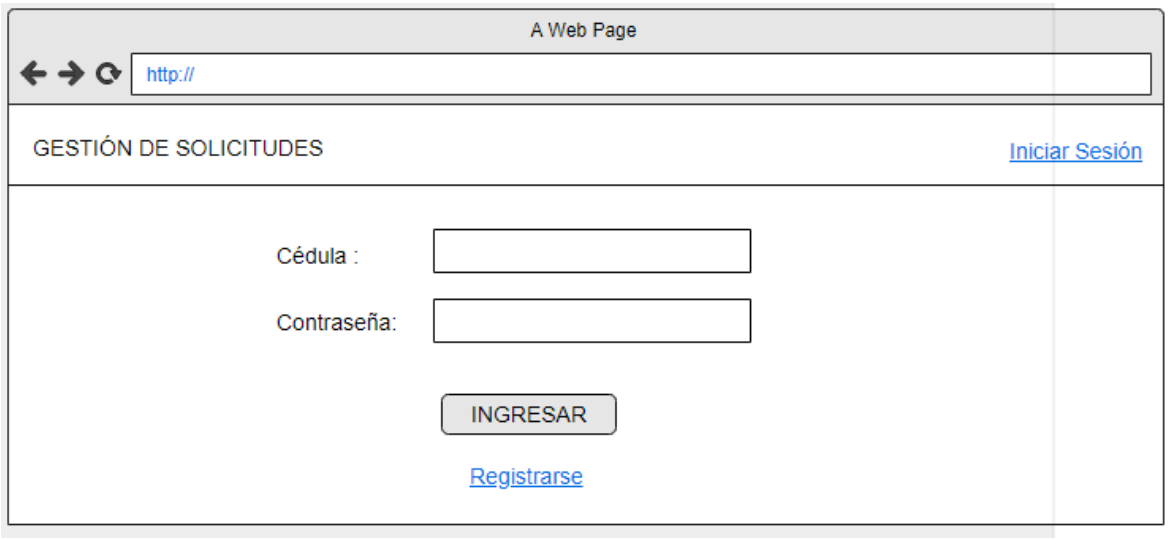


Figura 2.27 *Sketch* de la página de login

En la Figura 2.28 se muestra la página para el registro de un nuevo usuario, los datos personales que se deben ingresar son: nombres, apellidos, número de cédula, teléfono, correo electrónico, seleccionar una carrera, y crear una contraseña en las respectivas cajas de texto.

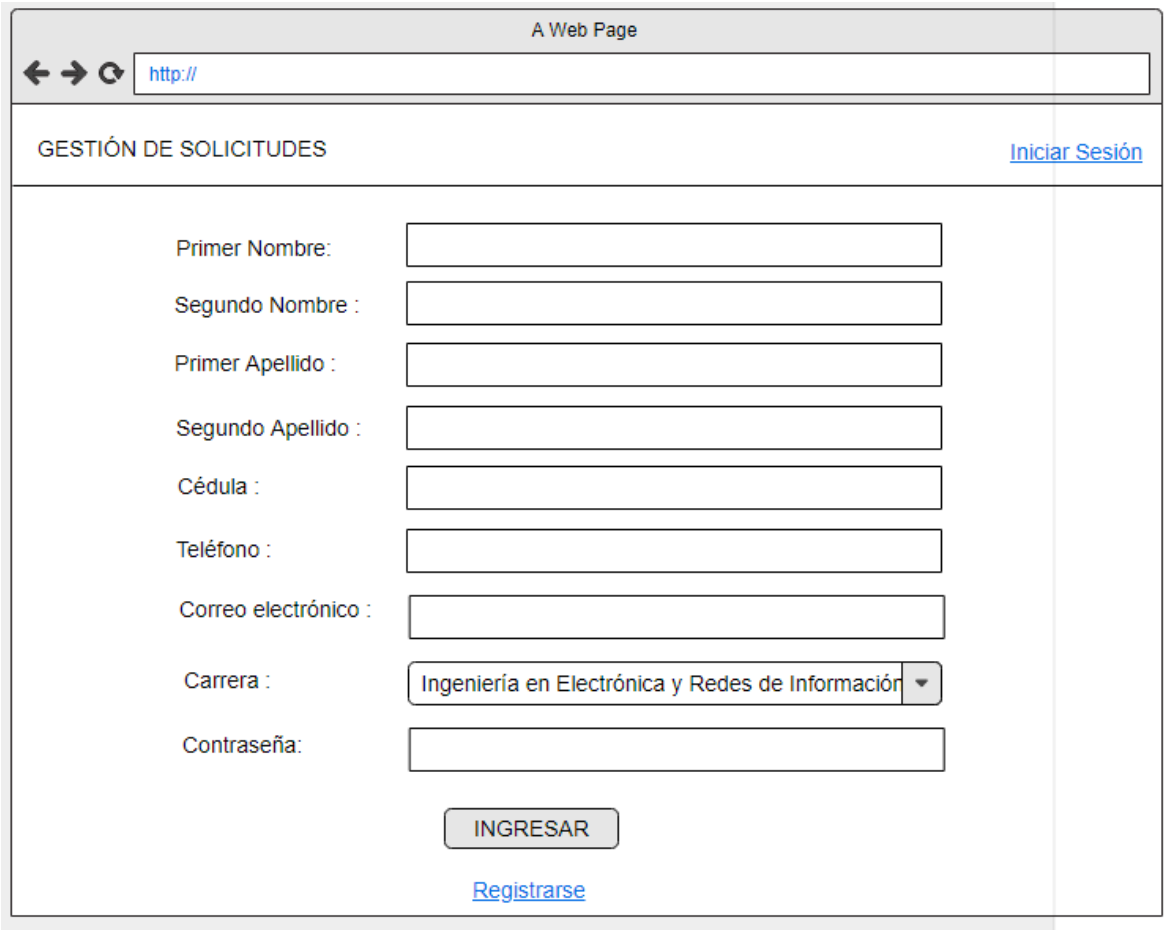


Figura 2.28 *Sketch* de la página de registro

La Figura 2.29 muestra el despliegue de un menú en el cual todos los usuarios de la aplicación pueden acceder a las pestañas:

- **Mi perfil** en la cual se desplegará la información del usuario que accede a la aplicación. Se tendrá una opción llamada Editar, con la cual el usuario podrá actualizar su información. Los campos que no podrá actualizar serán el número de cédula ya que es utilizado como credencial para el acceso a la aplicación y el otro campo es el rol de usuario que solo lo podrá cambiar el administrador, todos los demás cambios si pueden ser realizados por cada usuario.
- **Cambiar contraseña** al ingresar a esta página todos los usuarios sin importar su rol podrán hacer el cambio de su contraseña.
- **Salir** usado para cerrar su sesión de la aplicación.

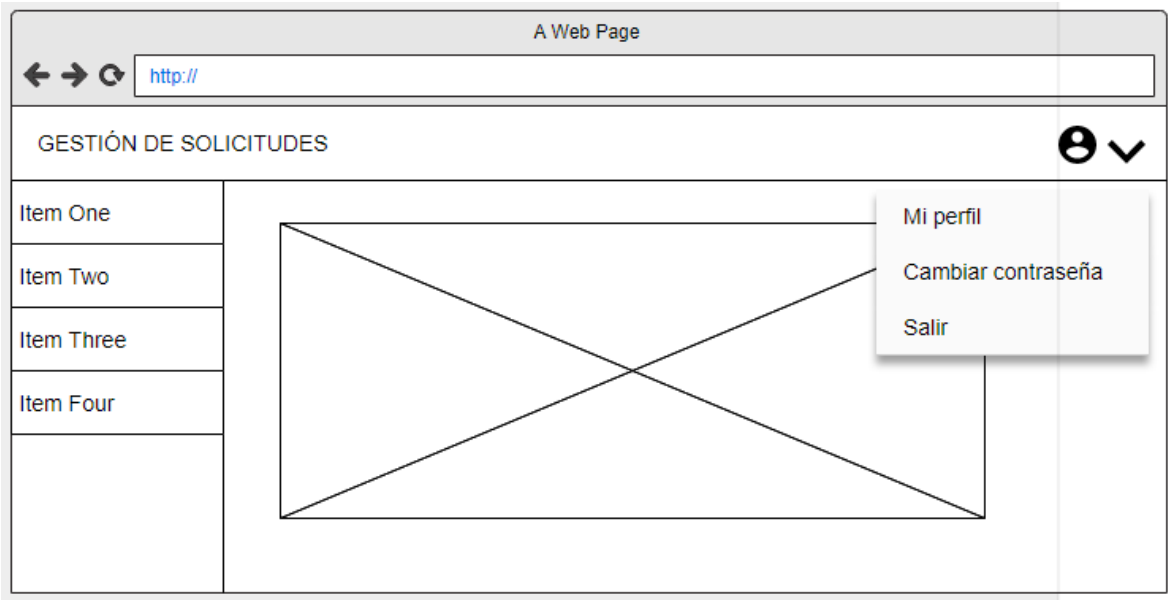


Figura 2.29 *Sketch* de la página de información personal

El administrador de la aplicación será el único que podrá crear nuevos usuarios, ver los detalles de información o actualizar los datos de los usuarios, para ello deberá seleccionar la opción Administración de Usuarios como se muestra en la Figura 2.30.

En la primera opción llamada Inicio que va a tener la aplicación independientemente del rol del usuario, se despliega información de cómo va a funcionar la aplicación es decir explica cómo se deben crear las solicitudes, la forma en que se van a revisar y algunos detalles más.

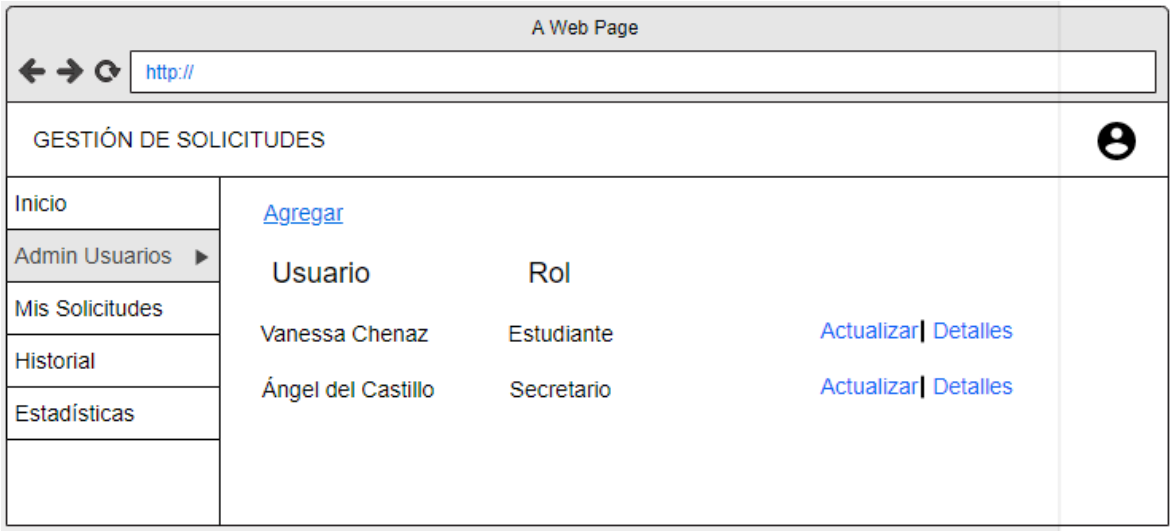


Figura 2.30 *Sketch* de la página de administración de usuarios

En la Figura 2.31 el usuario con rol de Secretario será el encargado de la administracion de plantillas, es decir todo lo que respecta al CRUD para la creación de las plantillas definidas con sus respectivos requisitos.

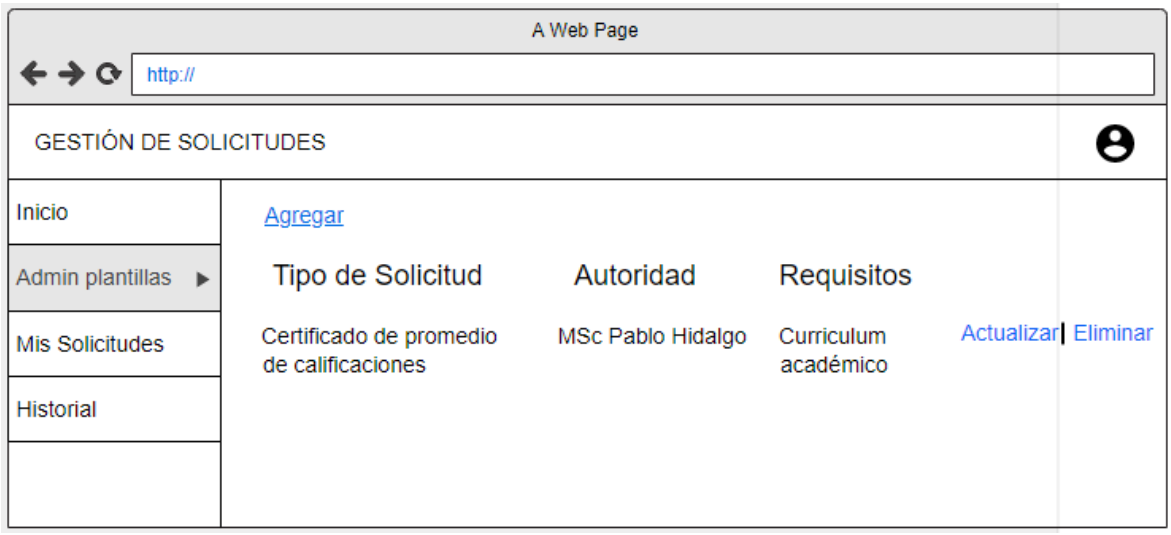


Figura 2.31 *Sketch* de la página de administración de plantillas

En la Figura 2.32, muestra la página que se desplegará para la creación de una nueva plantilla definida, en la cual se pedirá que se ingrese el tipo de solicitud, seleccione la autoridad a la que va dirigida, el tiempo de respuesta que lleva este tipo de solicitud y los requisitos que se necesitan para llevar a cabo este tipo de trámite.

La Figura 2.33 corresponde a la página que muestra todas las solicitudes con plantilla que el estudiante haya realizado; se tiene la opción de crear nuevas solicitudes y ver los detalles de las mismas. Cuando el estudiante recién empieza a usar la aplicación estas solicitudes no aparecen, solo aparece la opción nueva solicitud.

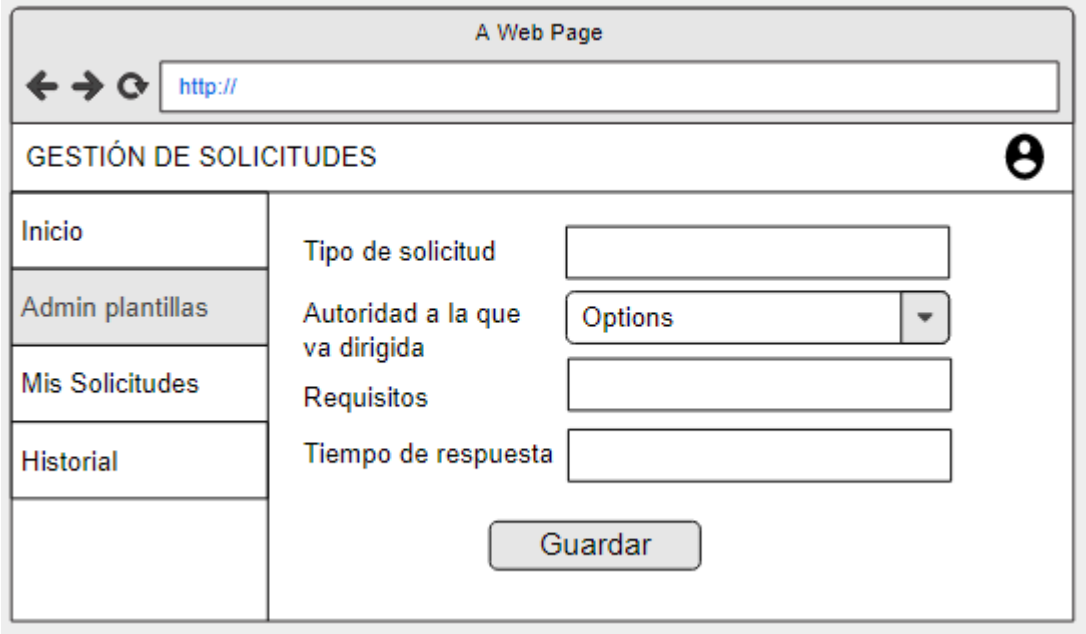


Figura 2.32 *Sketch* de la página para la creación de una plantilla

Figura 2.33 *Sketch* de la página de todas las solicitudes con plantilla del estudiante

La Figura 2.34 corresponde a la creación de una nueva solicitud que tiene una plantilla definida, el *Estudiante* escogerá el tipo de solicitud que desee generar, el detalle se cargará de forma automatica y si el tipo de solicitud que se haya seleccionado necesita requisitos se indicará cuál o cuáles son estos; los requisitos se deben cargar en formato .zip o .rar y de esta manera se procede a la creación de esta solicitud.

Mis solicitudes > Solicitud con plantilla definida > Nueva solicitud > Seleccionar el tipo de pedido que desea realizar > Cargar los documentos que se esté pidiendo > Crear.

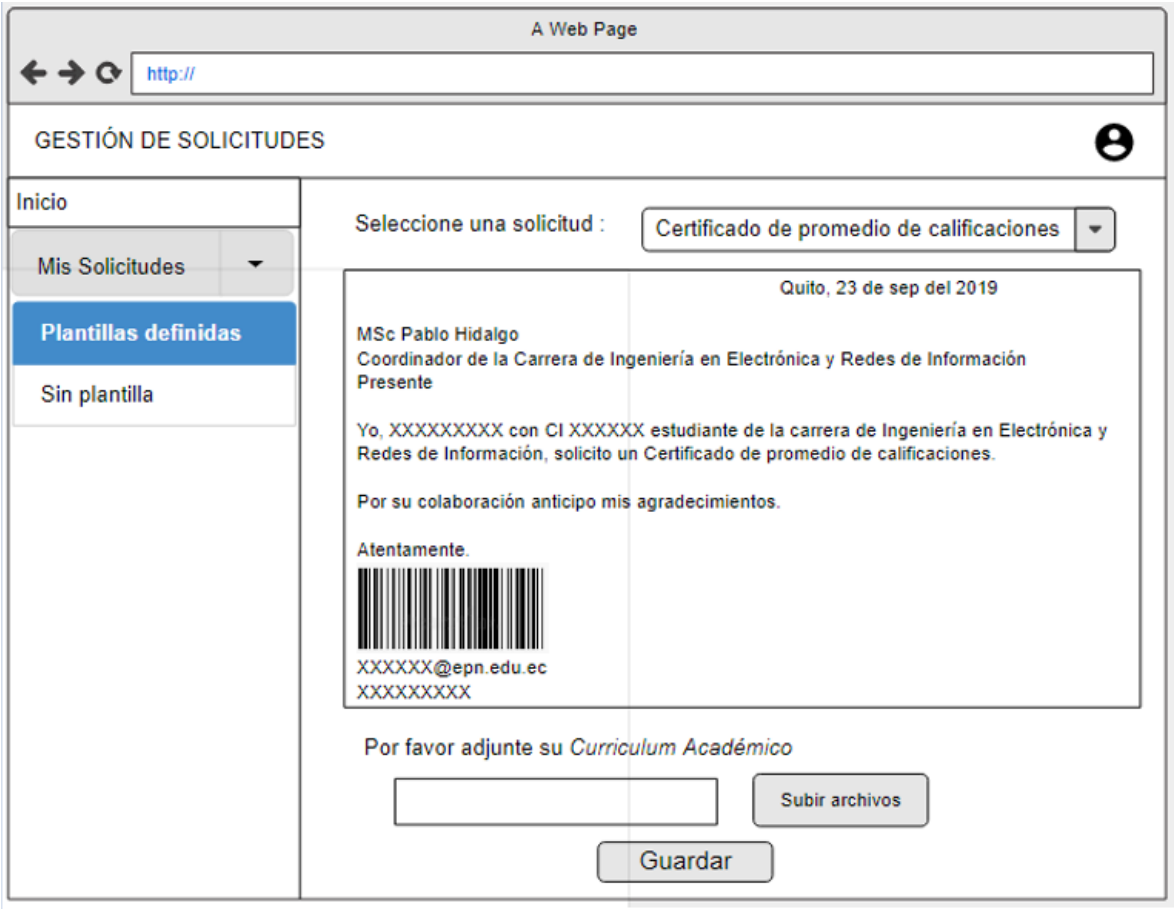


Figura 2.34 *Sketch* de la página para una solicitud con plantilla

En la Figura 2.35 se muestra cómo crear una nueva solicitud sin plantilla definida, en este tipo de solicitudes el *Estudiante* escoge la autoridad a la que va dirigida esta solicitud y se desplegará un detalle con datos precargados como: fecha, autoridad a la que va dirigida la solicitud, datos del estudiante como: nombres, apellidos, cédula, carrera, correo y número de teléfono, permitiendo que el *Estudiante* complete el tipo de solicitud que desea realizar.

Mis solicitudes > Sin Plantilla > Nueva solicitud > Seleccionar una autoridad > Llenar el pedido de la solicitud > Crear.

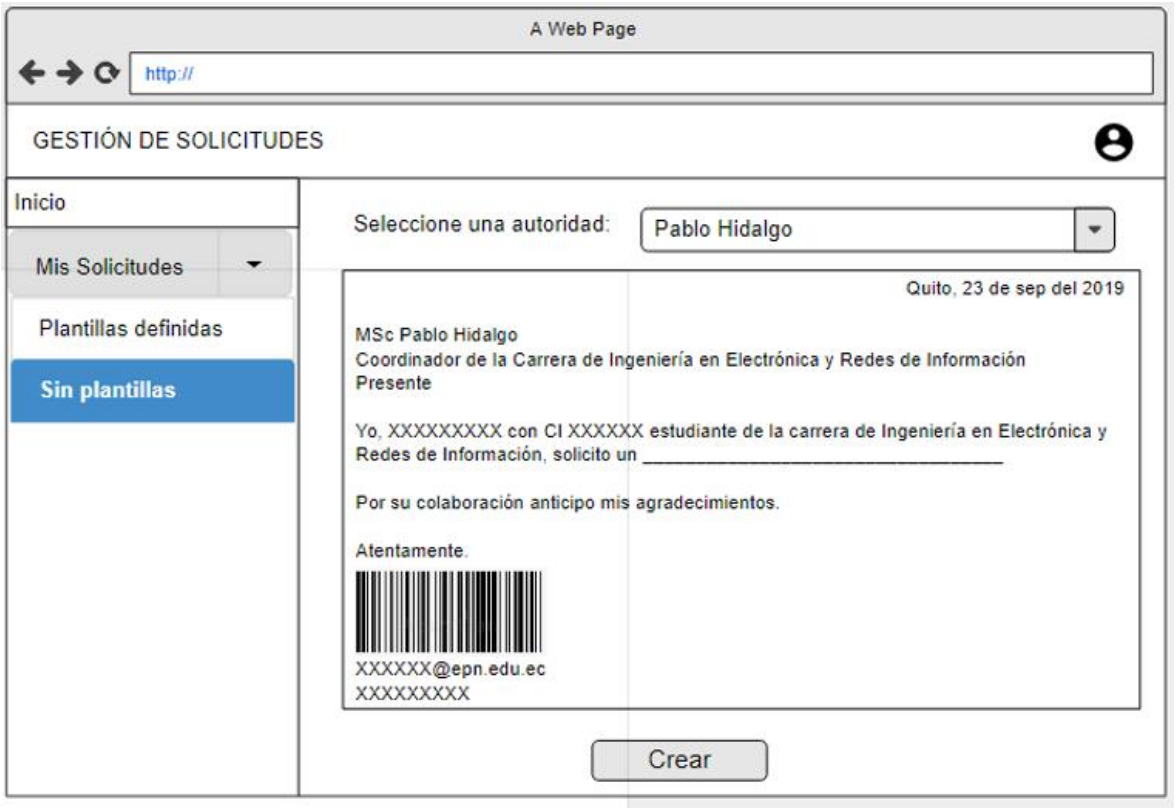


Figura 2.35 *Sketch* de la página para una solicitud sin plantilla

En la Figura 2.36, se muestra el detalle de una solicitud seleccionada como: el estado, fecha de creación, un espacio para comentario, la opción para descargar la solicitud en formato pdf y se muestra el detalle de la solicitud.

Figura 2.36 *Sketch* de la página para descargar una solicitud en .pdf

Cuando se crea una solicitud, ésta se guarda con un estado de Enviado, cuando sea revisada por el personal de secretaría el estado puede cambiar a: Aceptado o Rechazado según sea el caso y cuando la autoridad de trámite a la solicitud el estado puede cambiar a Aprobado o Rechazado todo depende del trámite dado a la solicitud, y en cada cambio de estado al *Estudiante* se le enviará un correo para indicar este estado de solicitud.

En la Figura 2.37 se despliegan todas las solicitudes que han sido creadas y con esto el personal de secretaría seleccionará la opción tramitar las solicitudes.

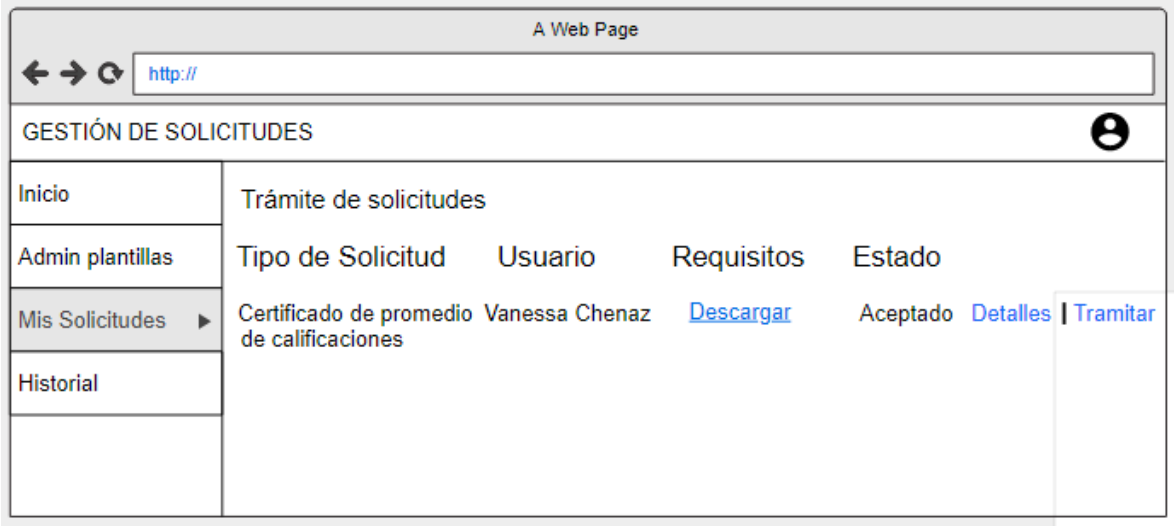


Figura 2.37 *Sketch* de la página para tramitar solicitud por el secretario

En la Figura 2.38 se muestra la página para revisar el detalle de la solicitud y los requisitos que ésta necesitaba, si se cumple con lo necesario el estado de la solicitud pasa a ser aceptado, pero si no, el estado será rechazado indicándole en el comentario el porqué del rechazo. Se tendrá la opción de cargar documentos en respuesta a la solicitud del estudiante de ser el caso.

En la opción historial se mostrará la información de todas las solicitudes.

En la Figura 2.39 se muestran todas las solicitudes con el estado aceptado que han sido enviadas a la autoridad y tiene la opción para darle el trámite respectivo.

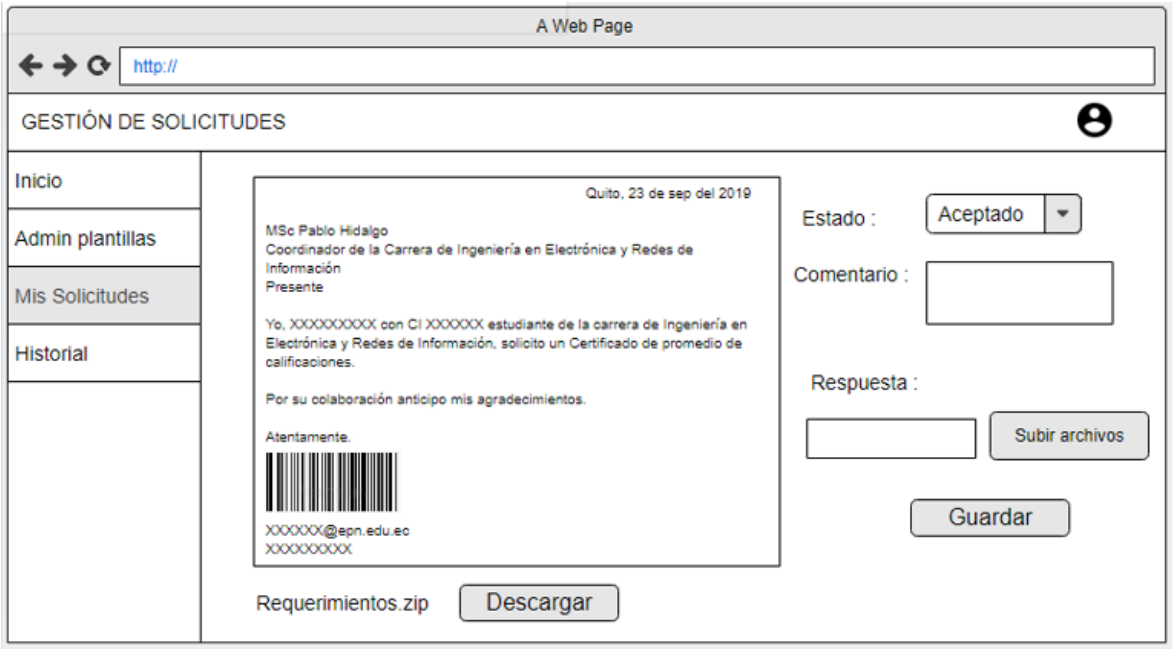


Figura 2.38 *Sketch* de la página para revisión de la solicitud por parte del Secretario

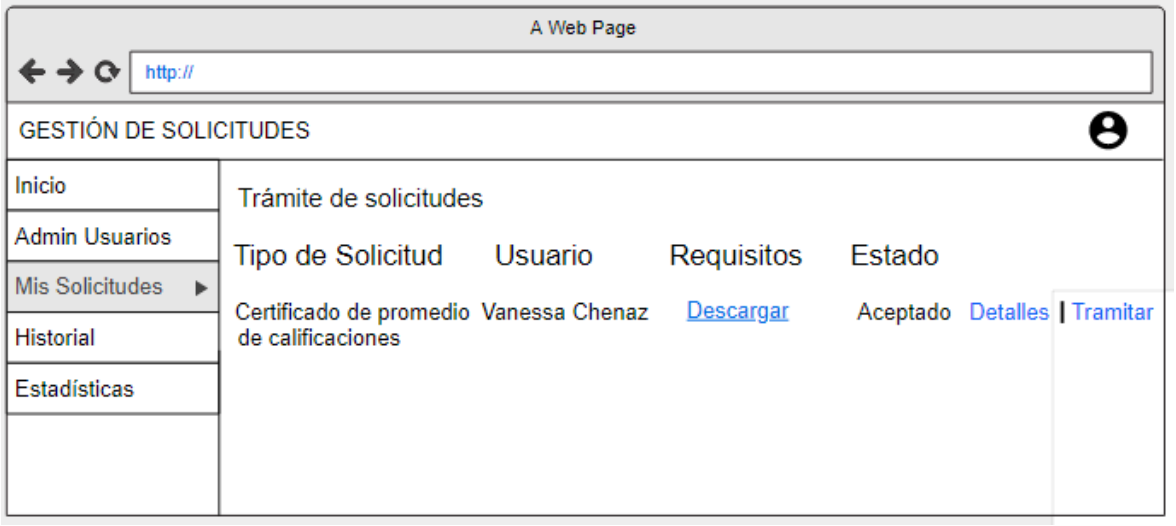


Figura 2.39 *Sketch* de la página de las solicitudes a tramitar por la autoridad

En la Figura 2.40 muestra la página para darle el respectivo trámite a la solicitud por parte de la autoridad, el estado de la solicitud que llega a la autoridad es en estado Aceptado y este puede cambiar a Aprobado o Rechazado y de igual manera se escribirá un comentario si la solicitud ha sido rechazada.

En caso de que el *Estudiante* quiera una respuesta en documento físico, la autoridad podrá cargar el documento que el estudiante solicite, en formato .zip y en la opción comentario le serán dadas indicaciones de dicha respuesta.

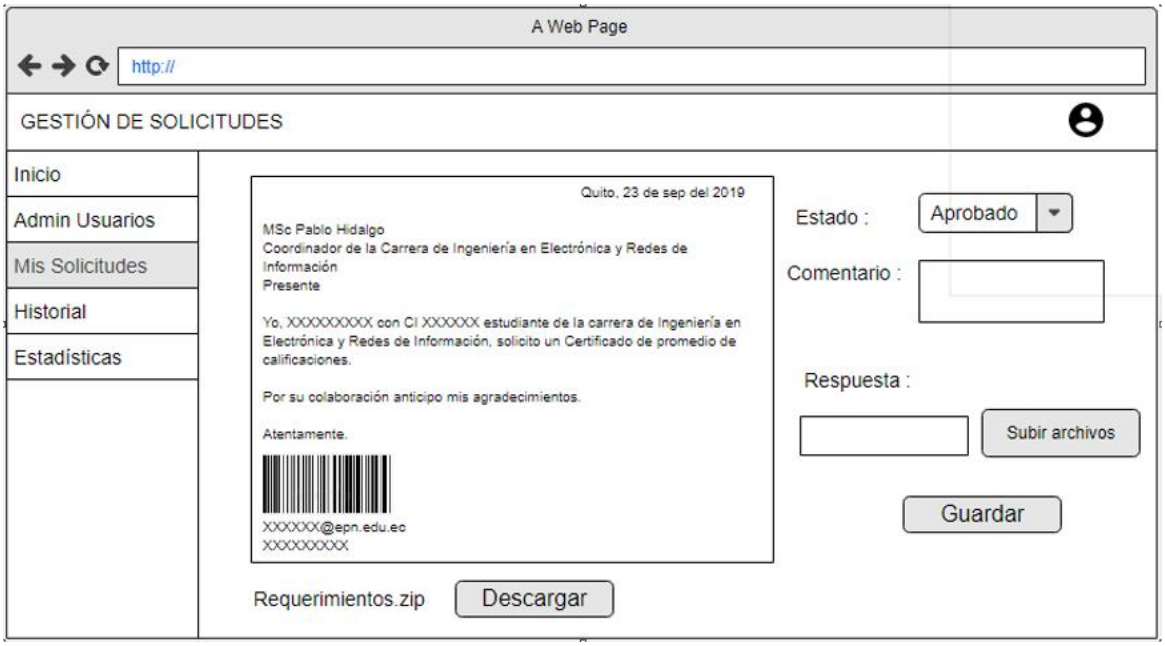


Figura 2.40 *Sketch* de la página para tramitar una solicitud por la autoridad

En la Figura 2.41 se muestra la página para el despliegue de estadísticas de las solicitudes, solo la autoridad podrá ver esta información, se tendrá un gráfico con las solicitudes que han sido rechazadas y eliminadas.

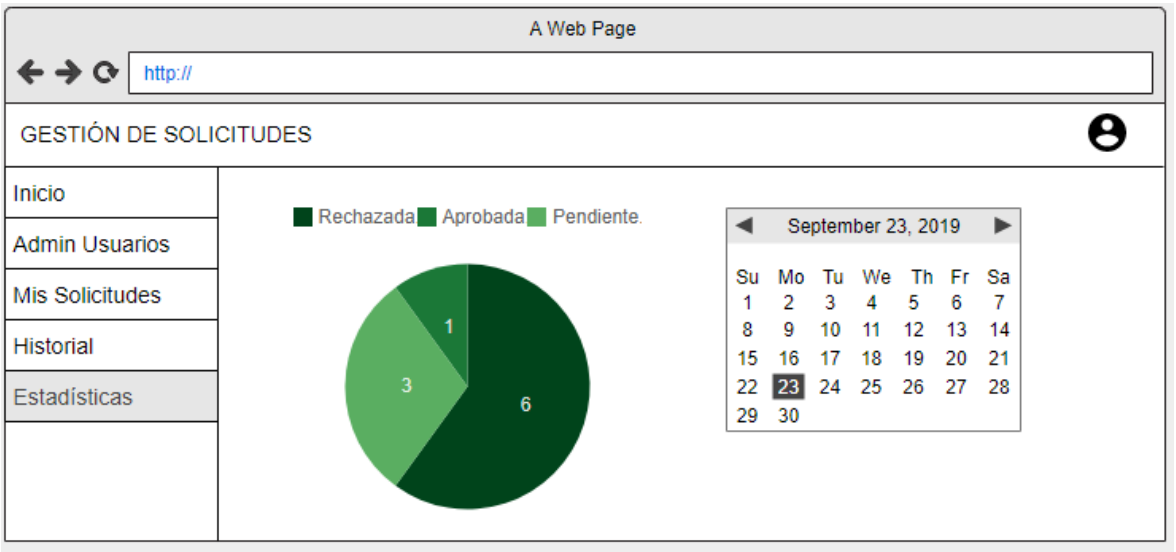


Figura 2.41 Sketch de la página de estadísticas

3. RESULTADOS Y DISCUSIÓN

3.1 PRUEBAS UNITARIAS

Para la realización de las pruebas unitarias se utilizó el *framework* de Microsoft MS Tests. Este *framework* permite crear un proyecto tipo *test* en la solución y configurarlo para que al momento de compilar la solución las pruebas unitarias se ejecuten como parte del proceso de compilación en Visual Studio 2017 como se muestra en la Figura 3.1.

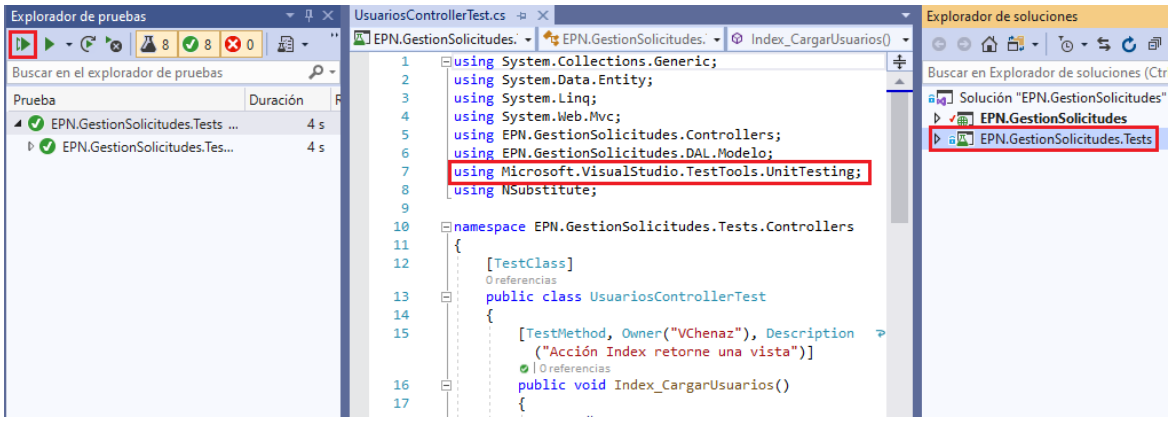


Figura 3.1 Pruebas unitarias en Visual Studio.

A continuación, se detalla un ejemplo de prueba unitaria, que se realizó en la capa presentación en el Controlador *UsuariosController*.

Esta prueba unitaria comprueba que la acción *Details* del Controlador *UsuariosController* devuelva la Vista esperada; para la acción *Details* se necesita del parámetro *id* ya que esta Vista se encarga de devolver toda la información referente al usuario seleccionado.

De acuerdo con el **¡Error! No se encuentra el origen de la referencia.**, se muestra una i nstancia sustituta de la base de datos ya que para las pruebas unitarias se necesita crear una base de datos falsa que no afecte a la base de datos real (en la línea 36). La base de datos falsa no cuenta con registros por lo cual se hace uso de objetos falsos que simulan la creación de dos registros de usuarios (líneas 38 a la 48) para probar la búsqueda del usuario con el *id* seleccionado, se crea una instancia del Controlador para proceder a acceder a sus métodos (línea 49). Para esta prueba en la se está especificando que busque el usuario que tiene el *id* 20 y se despliegue la información en la Vista *Details* (línea 50), y finalmente se pone la condición de que el valor esperado para esta prueba no debe ser *null* (línea 51).

Como se muestra en la Figura 3.2 se puede ver que la prueba se ha ejecutado exitosamente lo que quiere decir que la Vista *Details* del Controlador *UsuariosController*

está funcionando de manera correcta cuando se despliega información de un usuario especifico.

```
33      [TestMethod, Owner("VChenaz"), Description("Cargar usuario existente")]
        0 referencias | wilson geovanny panjon quinde, Hace 51 días | 1 autor, 1 cambio
34      public void Details_CargarUsuarioExistente()
35      {
36          var dbContext = Substitute.For<EPNBDSolicitudesEntities>();
37
38          var usuarios = new List<Usuario>() {
39              new Usuario { Nombre = "Marisol", Cedula = "0401617477", IDUsuario = 20 },
40              new Usuario { Nombre="Hector" , Cedula="0102030304", IDUsuario = 30 }
41          }.AsQueryable();
42          var usuarioSubstitute = Substitute.For<DbSet<Usuario>, IQueryable<Usuario>>();
43          ((IQueryable<Usuario>)usuarioSubstitute).Provider.Returns(usuarios.Provider);
44          ((IQueryable<Usuario>)usuarioSubstitute).Expression.Returns(usuarios.Expression);
45          ((IQueryable<Usuario>)usuarioSubstitute).ElementType.Returns(usuarios.ElementType);
46          ((IQueryable<Usuario>)usuarioSubstitute).GetEnumerator().Returns(usuarios.GetEnumerator());
47
48          dbContext.Usuario.Returns(usuarioSubstitute);
49          UsuariosController controller = new UsuariosController(dbContext);
50          ViewResult result = controller.Details(20) as ViewResult;
51          Assert.IsNotNull((Usuario)result.ViewData.Model);
52      }
```

Código 3.1 Prueba unitaria exitosa para un usuario existente

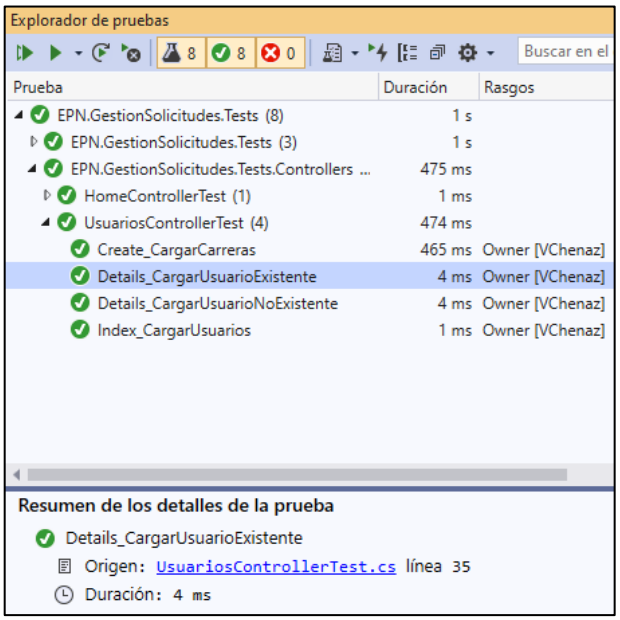


Figura 3.2 Prueba unitaria exitosa

Si se cambia la línea 50 del **¡Error! No se encuentra el origen de la referencia.** por la l ínea 50 del

, se puede ver que la prueba unitaria falla, debido a que el usuario con el *id* 25 no existe y la prueba requiere que el resultado no sea null, como se mostró en la línea 51 del **¡Error! N o se encuentra el origen de la referencia.**; el resultado fallido de la prueba se muestra en la Figura 3.3.

```
50        ViewResult result = controller.Details(25) as ViewResult;
```

Código 3.2 Cambio de condición

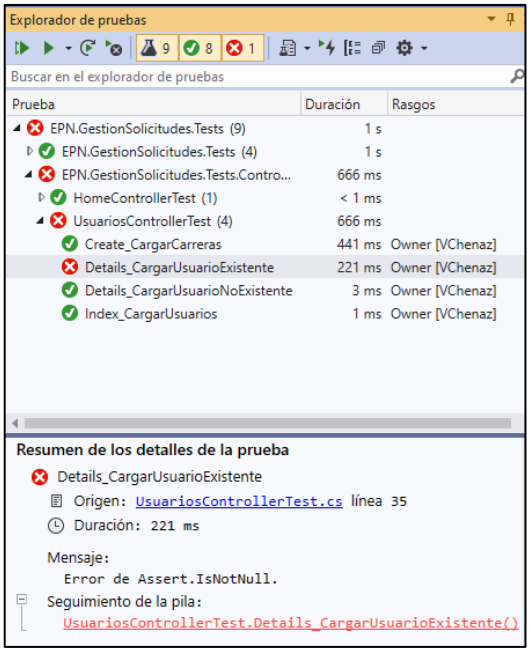


Figura 3.3 Error en prueba unitaria

Para que se pruebe la no existencia de un usuario de manera exitosa se debe cambiar la línea 51 del **¡Error! No se encuentra el origen de la referencia.** por la línea 76 que se m uestra en el **¡Error! No se encuentra el origen de la referencia.**.

```
76        Assert.IsNull(result.ViewData.Model);
```

Código 3.3 Cambio para prueba exitosa

Como se detalló en el capítulo 1, se pueden crear tantas pruebas unitarias como se desee, a continuación, se indican algunas de las pruebas realizadas y las demás se encuentran en el ANEXO E.

El Código 3.4 prueba que exista una plantilla llamada Certificado de culminación de plan de estudios, como se indicó anteriormente estos datos son los que la aplicación considera necesarios, por esto se procede a escribir datos falsos solamente para probar que todo funcione de manera correcta.

En el Código 3.5 prueba el funcionamiento de la opción *Detalles* que muestra la información de una solicitud existente y en el Código 3.6 prueba que la solicitud no exista, el resultado de las pruebas realizadas se muestra en la Figura 3.4.

```
using EPN.GestionSolicitudes.DAL.Modelo;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NSubstitute;

namespace EPN.GestionSolicitudes.Tests.Controllers
{
    [TestClass]
    0 referencias
    public class PlantillaControllerTest
    {
        [TestMethod, Owner("VChenaz"), Description("Acción Index retorne una vista")]
        0 referencias
        public void Index_CargarPlantillas()
        {
            var dbContext = Substitute.For<EPNBDSolicitudesEntities>();
            var plantilla = new List<Plantilla>() { new Plantilla { TipoSolicitud = "Certificado de culminacion de
              plan de estudios", TiempoRespuesta = 3 } }.AsQueryable();
            var plantillaSubstitute = Substitute.For<DbSet<Plantilla>, IQueryable<Plantilla>>();

            ((IQueryable<Plantilla>)plantillaSubstitute).Provider.Returns(plantilla.Provider);
            ((IQueryable<Plantilla>)plantillaSubstitute).Expression.Returns(plantilla.Expression);
            ((IQueryable<Plantilla>)plantillaSubstitute).ElementType.Returns(plantilla.ElementType);
            ((IQueryable<Plantilla>)plantillaSubstitute).GetEnumerator().Returns(plantilla.GetEnumerator());

            dbContext.Plantilla.Returns(plantillaSubstitute);
            PlantillasController controller = new PlantillasController(dbContext);
            ViewResult result = controller.Index() as ViewResult;
            Assert.IsNotNull((List<Plantilla>)result.ViewData.Model);
        }
    }
}
```

Código 3.4 Prueba unitaria para el despliegue de plantillas

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NSubstitute;

namespace EPN.GestionSolicitudes.Tests.Controllers
{
    [TestClass]
    0 referencias
    public class SolicitudControllerTest
    {
        [TestMethod, Owner("VChenaz"), Description("Cargar solicitud existente")]
        0 referencias
        public void Details_CargarSolicitudExistente()
        {
            var dbContext = Substitute.For<EPNBDSolicitudesEntities>();

            var solicitudes = new List<Solicitud>() {
                new Solicitud { IDPlantilla= 1, IDUsuario = 2, IDSolicitud=4 },
                new Solicitud { IDPlantilla= 2, IDUsuario = 3, IDSolicitud=5 }
            }.AsQueryable();
            var solicitudSubstitute = Substitute.For<DbSet<Solicitud>, IQueryable<Solicitud>>();
            ((IQueryable<Solicitud>)solicitudSubstitute).Provider.Returns(solicitudes.Provider);
            ((IQueryable<Solicitud>)solicitudSubstitute).Expression.Returns(solicitudes.Expression);
            ((IQueryable<Solicitud>)solicitudSubstitute).ElementType.Returns(solicitudes.ElementType);
            ((IQueryable<Solicitud>)solicitudSubstitute).GetEnumerator().Returns(solicitudes.GetEnumerator());

            dbContext.Solicitud.Returns(solicitudSubstitute);
            SolicitudesController controller = new SolicitudesController(dbContext);
            ViewResult result = controller.Details(4) as ViewResult;
            Assert.IsNotNull((Solicitud)result.ViewData.Model);
        }
```

Código 3.5 Prueba unitaria para cargar una Solicitud existente

```
39          [TestMethod, Owner("VChenaz"), Description("Cargar solicitud no existente")]
            0 referencias
40          public void Details_CargarSolicitudNoExistente()
41          {
42              var dbContext = Substitute.For<EPNBDSolicitudesEntities>();
43
44              var solicitudes = new List<Solicitud>() {
45                  new Solicitud { IDPlantilla= 1, IDUsuario = 2, IDSolicitud=4 },
46                  new Solicitud { IDPlantilla= 2, IDUsuario = 3, IDSolicitud=5 }
47              }.AsQueryable();
48              var solicitudSubstitute = Substitute.For<DbSet<Solicitud>, IQueryable<Solicitud>>();
49              ((IQueryable<Solicitud>)solicitudSubstitute).Provider.Returns(solicitudes.Provider);
50              ((IQueryable<Solicitud>)solicitudSubstitute).Expression.Returns(solicitudes.Expression);
51              ((IQueryable<Solicitud>)solicitudSubstitute).ElementType.Returns(solicitudes.ElementType);
52              ((IQueryable<Solicitud>)solicitudSubstitute).GetEnumerator().Returns(solicitudes.GetEnumerator());
53
54              dbContext.Solicitud.Returns(solicitudSubstitute);
55              SolicitudesController controller = new SolicitudesController(dbContext);
56              ViewResult result = controller.Details(7) as ViewResult;
57              Assert.IsNull((Solicitud)result.ViewData.Model);
58          }
```

Código 3.6 Prueba unitaria para una Solicitud no existente

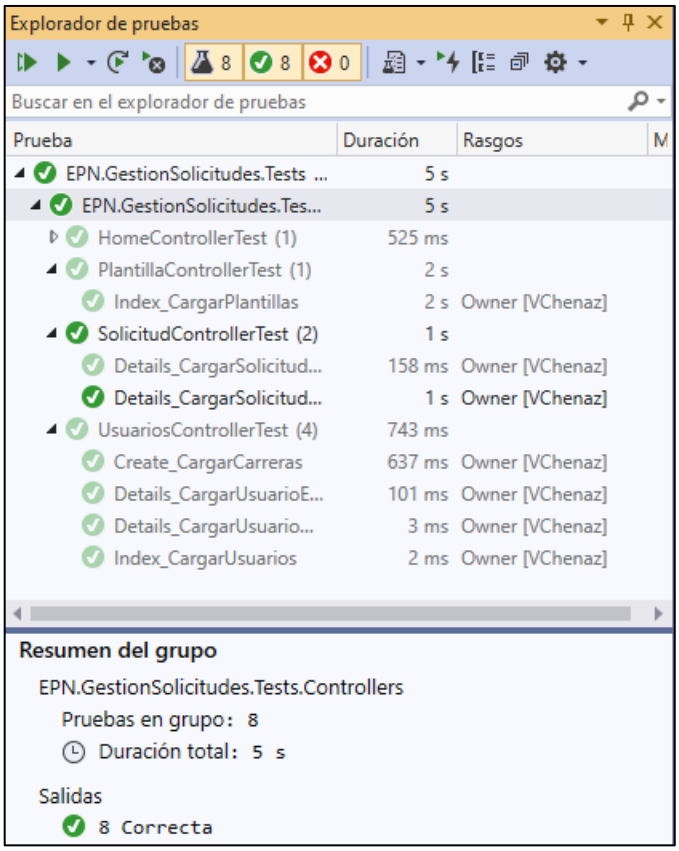


Figura 3.4 Pruebas unitarias correctas de varios controladores

3.2 PRUEBAS DE INTEGRACIÓN

La Tabla 3.1, muestra las pruebas de integración a realizar.

Tabla 3.1 Listado de pruebas de integración

Número de prueba	Detalle de la prueba
1	Creación de cuentas de usuario
2	Ingreso a la aplicación
3	Creación de plantillas base
4	Generación de solicitudes con plantilla
5	Generación de solicitudes sin plantilla
6	Trámite a una solicitud con plantilla
7	Trámite a una solicitud sin plantilla
8	Respuesta al estudiante en la aplicación
9	Visualización de estadísticas de las solicitudes

3.2.1 CREACIÓN DE CUENTAS DE USUARIO

La página principal de la aplicación despliega un enlace para crear una nueva cuenta, en la Figura 3.5, el usuario debe ingresar los datos de su información personal. Todos los campos son requeridos y estas validaciones se las hace gracias a las *Data Annotations* por lo que los mensajes de error se muestran de forma inmediata.

En la Figura 3.8, se muestra la forma correcta de llenar la información para crear un nuevo usuario y poder acceder a la aplicación, se puede ver que gracias a las *Data Annotations* que se usaron no se muestra la contraseña que el usuario está ingresando.

La información de los usuarios que se han registrado en la aplicación se guarda en la respectiva tabla de la base de datos, todo esto se puede ver en la Figura 3.9, además el campo contraseña se muestra de manera cifrada por seguridad.

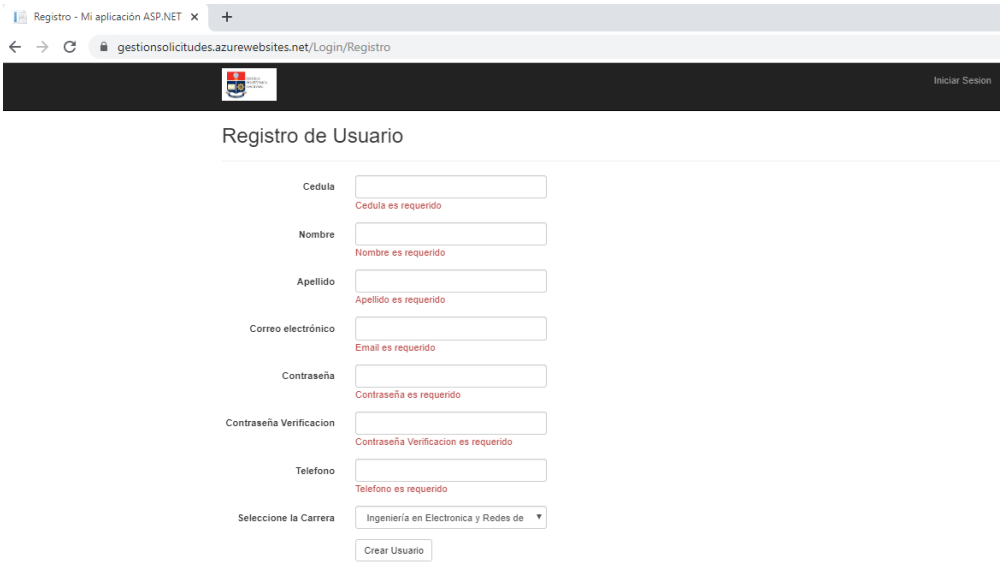


Figura 3.5 Vista para creación de nuevo usuario con mensajes de validación

Para la creación de un usuario con el rol de Secretario, se le pide al Coordinador que en este caso es el administrador de la aplicación, que cree un nuevo usuario como se muestra en la Figura 3.6.

Una vez creada la cuenta de usuario Secretario, se le asignarán las credenciales correspondientes para que acceda a la aplicación y se le pide que cambie la contraseña de su cuenta por seguridad como se muestra en la Figura 3.7.

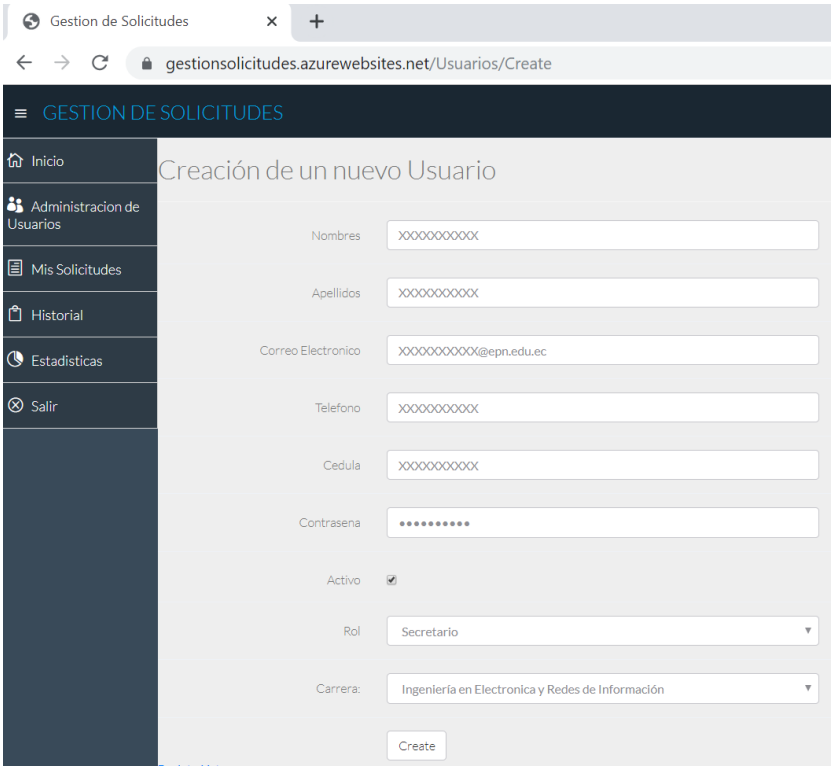


Figura 3.6 Creación del usuario Secretario

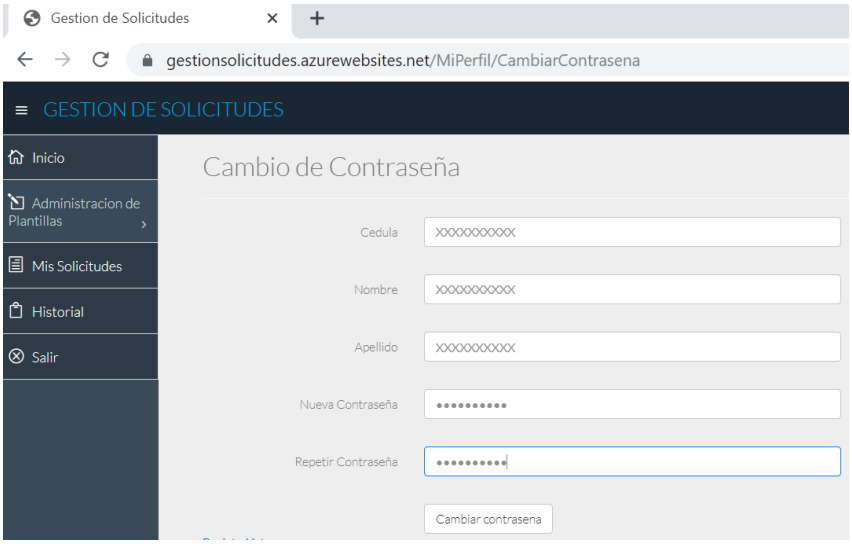


Figura 3.7 Cambio de contraseña del Secretario

3.2.2 INGRESO A LA APLICACIÓN

La página principal muestra las cajas de texto donde se va a ingresar el usuario que es el número de cédula y la contraseña que creó al momento del ingreso. Si uno de los campos es incorrecto la aplicación no le permitirá el ingreso sino hasta que haya ingresado los datos correctos.

Cuando las credenciales son correctas y dependiendo del rol del usuario, se le desplegará la respectiva página; todas estas páginas tienen un menú en el cual la opción de Inicio y la opción para salir son comunes, los diferentes cambios se describen a continuación.

Para el usuario con el rol de Estudiante se le presenta la página como se indica en la Figura 3.10, en el menú aparte de las opciones comunes se cuenta con la opción Mis Solicitudes que es el núcleo de esta página; en esta opción se listan dos sub-opciones, la primera donde el Estudiante genera una solicitud de manera automática y la segunda donde el Estudiante crea la solicitud de su propia autoría.

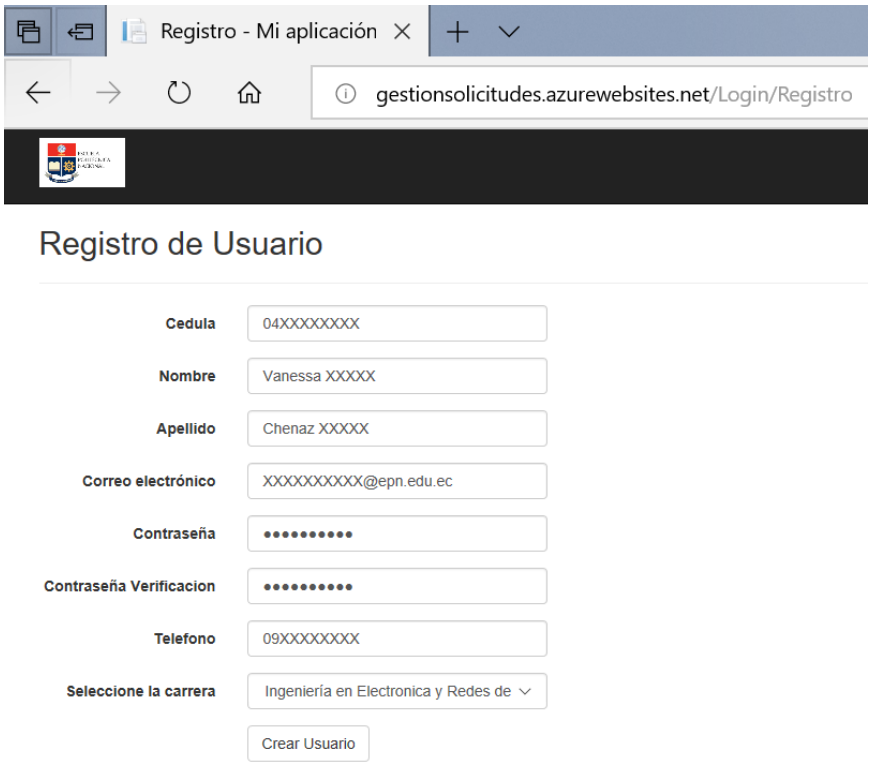


Figura 3.8 Forma correcta del registro de un nuevo usuario

	Nombre	Apellido	CorreoElectronico	Telef...	Cedula	Contrasena
4	Cristian Alexander	Rodríguez Durán	XXXXXXX @gmail.com	099...	171...	oOK9Eu4dxGjhdxKB3wGm2/7lRYLmvt6TlPybnpiH1Oxy0/tfM6PDk...
5	Byron Gonzalo	Arias Melendres	XXXXXXX i@hotmail.com	099...	172...	ujZV4voF5G2/3e932y4KndpfoLk68OdhQVtGrDK470F5fPiRAP/m0...
6	Ernesto Santiago	Gangotena Torres	XXXXXXX @gmail.com	099...	172...	1AoOLhJjwweZ9YeA1jK6ZVMAAH5w9YunmDoa0Rhi6bSMVU99o...
7	Gabriela del Carmen	Duque Sandoval	XXXXXXXXX+@epn.edu.ec	093...	100...	tPw9xJq/Owzm/nlXbtDzCAUsI/YdZ7pfYuyGn/+qNI4eFSwton6/kv...
8	Alex Fernando	Quilachamin Morocho	XXXXX @hotmail.com	099...	171...	m27a5AlHqQGiuns97Us2H81bbm85BDXByfOP5ySjgNTXC0LVGhl...
9	Willian Javier	Viracocha Verdezoto	XXXXXXXXXX @epn.edu.ec	098...	172...	use9QTUzYkKXsb4eC7hSFFXEHLjHeaDDqg+eRtf9XWhFffF83Tx...
10	Maria Belen	Cuesta Plua	XXXXXXXXX @epn.edu.ec	098...	172...	Zwt016+X9QX6tlQ9as40sFliGnEaRy0l1rxXhxGPzFkP0nRu+I2eOA...
11	Alex Alejandro	Montenegro Changotasi	XXXXXXXXXX @epn.edu.ec	098...	040...	/sucZ6lw0g2iilwwtjAKMJFxfh5BB1VhEw3to//0jpWsU4nBtkZ1U+...
12	Ana Esthela	Sanguil Llangari	XXXXXXX @epn.edu.ec	098...	180...	FhMYpC5J9tDU19GA4+LXvFMS2YdOhbYg9VcwTPaKMTX0cs6jb...
13	Fernando Jose	Herrera	XXXXXXXXXX !@epn.ed...	098...	171...	hbqbpyiEBdhaxr/dEAse75tnietB+6ht7xlDLuqtsscAoWpWcIluY1Slp...

Figura 3.9 Tabla Usuario

Figura 3.10 Página mostrada al usuario con el rol Estudiante

Para el usuario con el rol de Secretario se le muestra la página como se indica en la Figura 3.11; en el menú aparte de las opciones comunes se cuenta con la opción Administración de plantillas donde se realiza el CRUD para las plantillas y requisitos que tendrán éstas.

La opción Mis Solicitudes le permitirá al Secretario darle el trámite respectivo a las solicitudes recibidas y la opción historial muestra todas las solicitudes que han sido recibidas independientemente del estado que tengan.

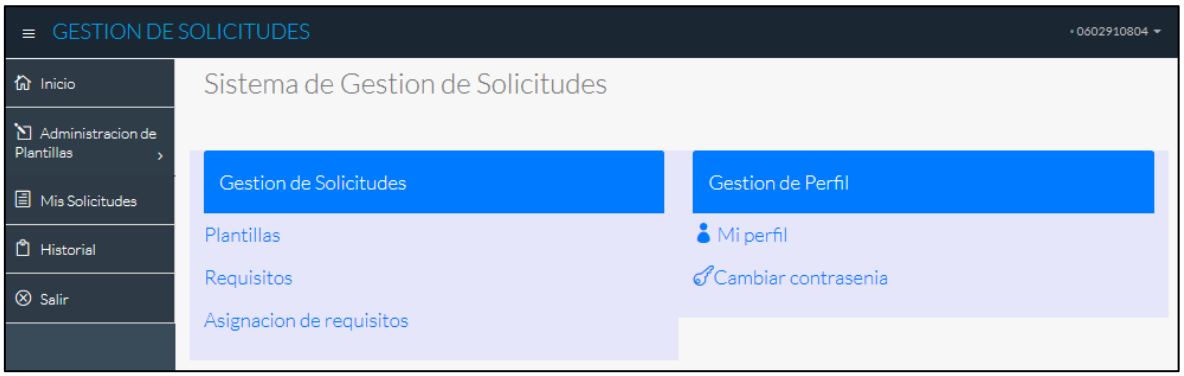


Figura 3.11 Página mostrada al usuario con el rol Secretario

Para el usuario con el rol de Coordinador se le muestra la página como se indica en la Figura 3.12; en el menú aparte de las opciones comunes se cuenta con la opción Administración de Usuarios, ya que el Coordinador será el administrador que se encargará de añadir, ver o editar usuarios.

En la opción Mis Solicitudes realizará el trámite respectivo a las solicitudes que tienen el estado de aceptado, estado que fue cambiado por el Secretario si cumple con los requisitos pedidos.

En la opción historial revisará las solicitudes que le han llegado y en estadísticas se le desplegará una imagen con un índice de las solicitudes que han sido aceptadas y rechazadas.

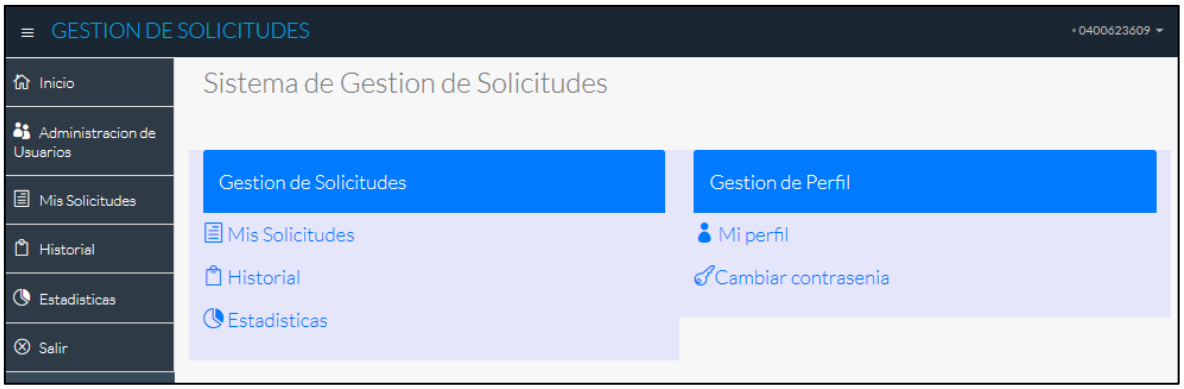


Figura 3.12 Página mostrada al usuario con el rol Coordinador

En la Figura 3.13, se muestra el submenú en el lado derecho en el cual están las opciones Mi perfil, Cambiar contraseña y salir, se está mostrando la opción Mi Perfil en la cual se despliega la información del usuario con la opción para editar la información.

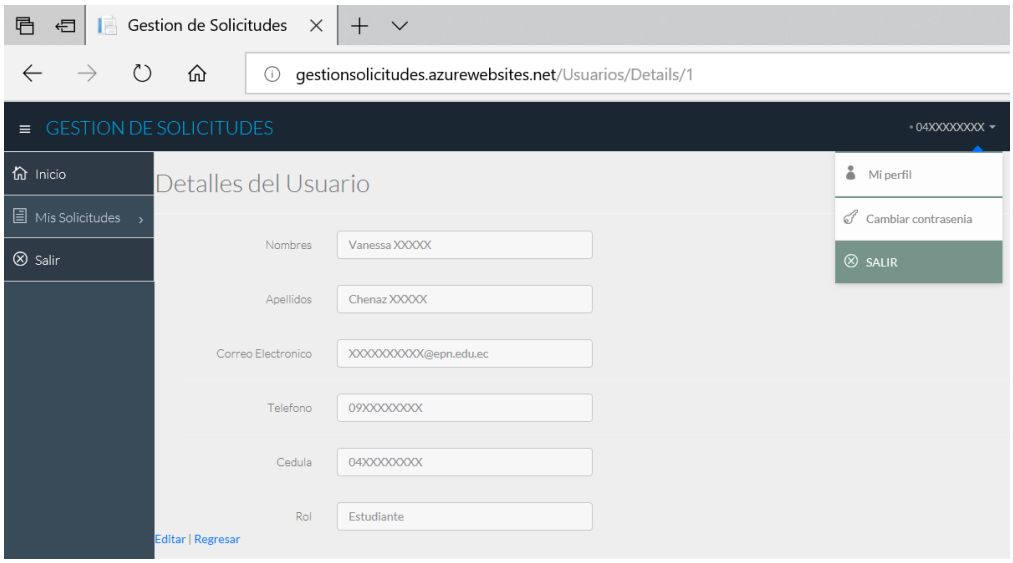


Figura 3.13 Página Mi Perfil de la aplicación

3.2.3 CREACIÓN DE PLANTILLAS BASE

El usuario con el rol de Secretario es el encargado de crear este tipo de plantillas base; para crear una nueva plantilla se sigue los siguientes pasos: seleccionar la opción Administración de Plantillas → Plantillas → Crear nueva plantilla. Se despliega una Vista como en la Figura 3.14, se selecciona la autoridad a la que va dirigida la solicitud → ingresar el nombre de la solicitud → ingresar el tiempo de respuesta y se procede a crear la nueva plantilla.

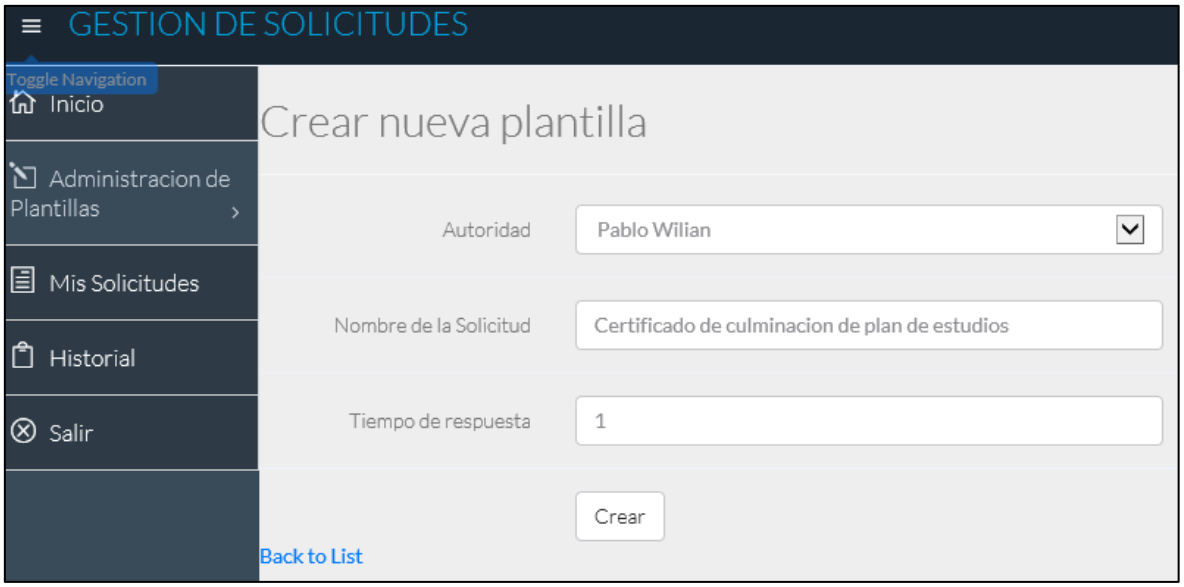


Figura 3.14 Creación de una nueva plantilla base

Ahora se procede a crear los requisitos seleccionar: Administración de Plantillas → Plantillas → Crear nuevo requisito, estos se crean uno a uno como en la Figura 3.15 y para asignar los requisitos a las plantillas se selecciona: Administración de Plantillas → Asignación de requisitos → Nueva asignación y se selecciona el tipo de plantilla y el tipo de solicitud como en la Figura 3.16 y se crea esta nueva asignación.

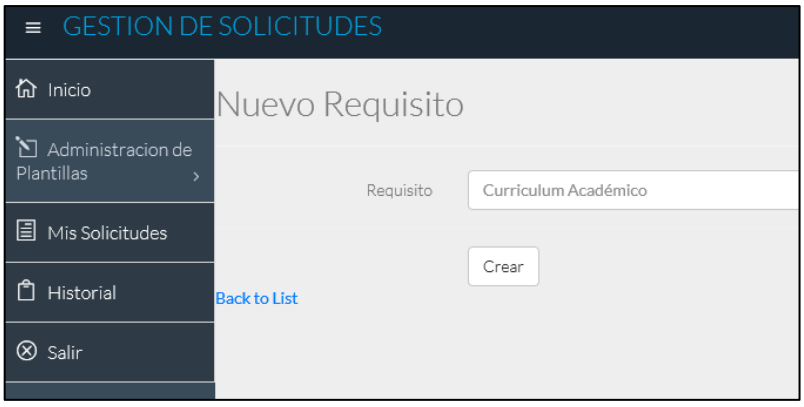


Figura 3.15 Creación de un requisito

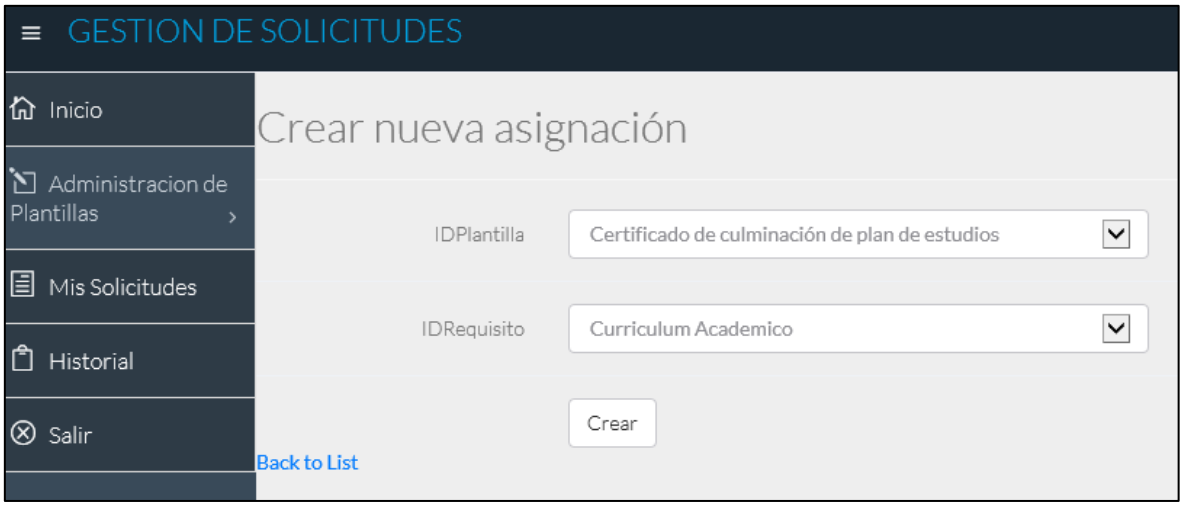


Figura 3.16 Asignación de requisitos a plantillas

Todas las plantillas se muestran en la opción Administración de Plantillas → Plantillas como se muestra en la Figura 3.17, se escoge la opción detalles para ver la información completa de las solicitudes que se van a desplegar al usuario como se muestra en la Figura 3.18.

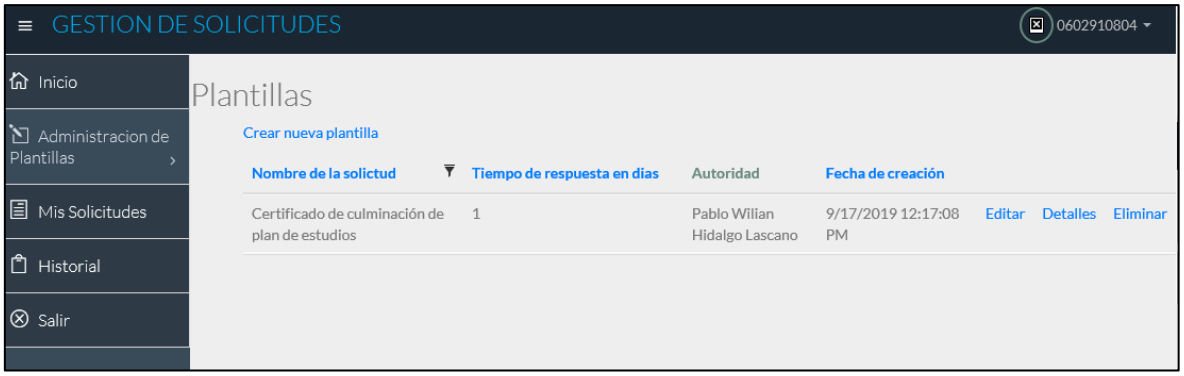


Figura 3.17 Lista de plantillas

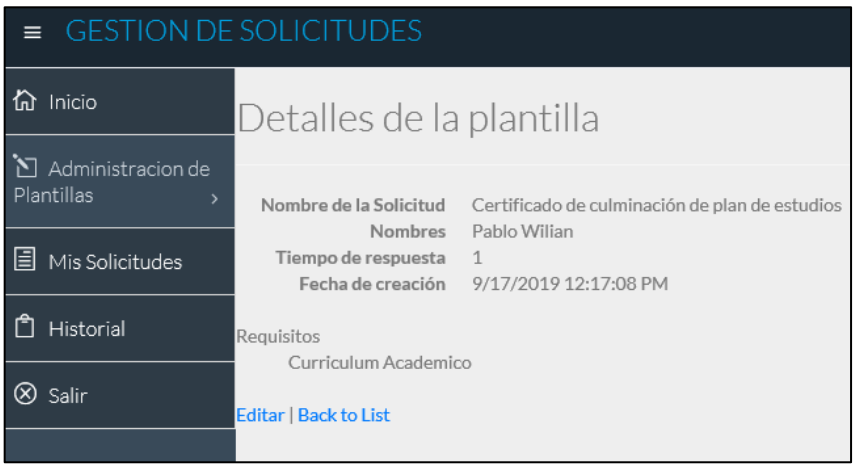


Figura 3.18 Detalles de una plantilla

3.2.4 GENERACIÓN DE SOLICITUDES CON PLANTILLA

Cuando se ingresa con el rol de Estudiante se selecciona la opción Mis Solicitudes en el submenú que se despliega se escoge la opción Plantillas definidas, se desplegará una lista de solicitudes que el estudiante haya realizado, la primera vez que ingrese a la aplicación no se le debe desplegar nada en esta sección.

Para crear una nueva solicitud se requiere seleccionar: Mis Solicitudes → Plantillas definidas → Crear nueva solicitud → Seleccione una plantilla.

En la Figura 3.19 se muestra cómo se va a crear la solicitud de manera automática con los datos precargados, esto quiere decir que se está obteniendo de manera correcta los valores de la base de datos. La aplicación le indica qué documentos debe subir, en este caso el Currículum Académico y se procede a crear la solicitud.

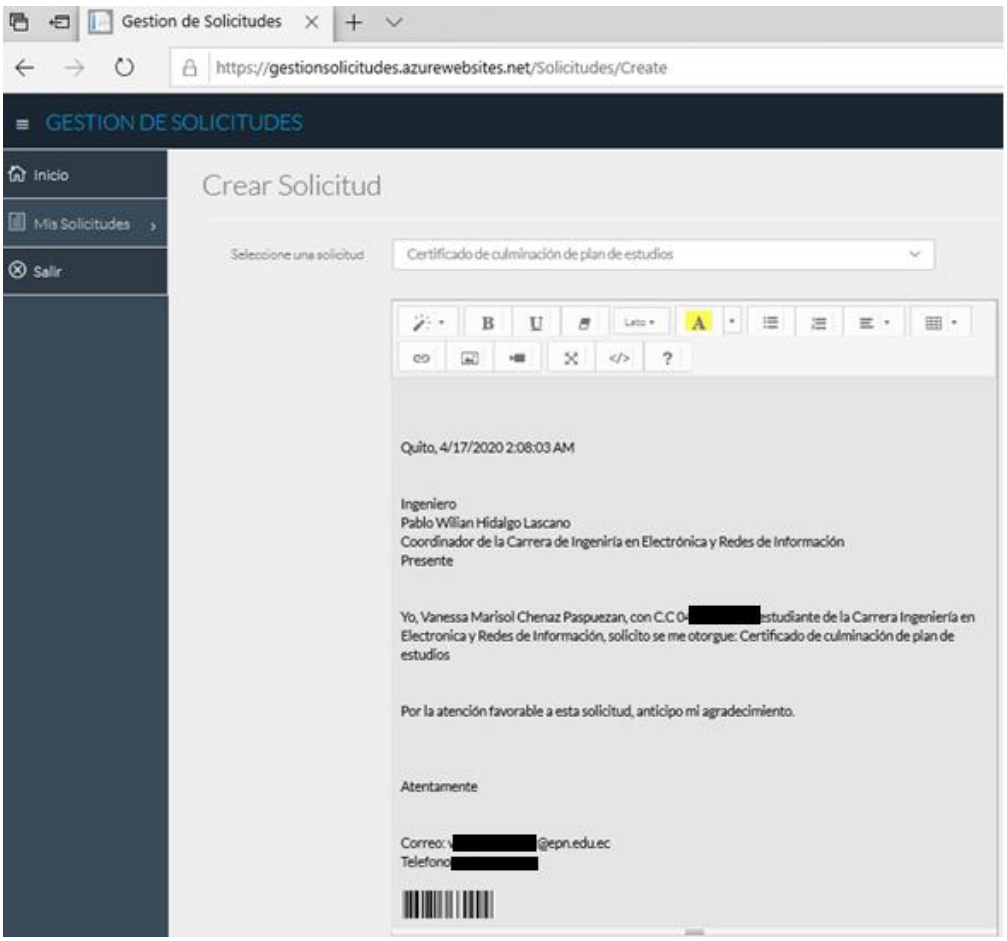


Figura 3.19 Crear una solicitud con plantilla (Parte 1 de 2)

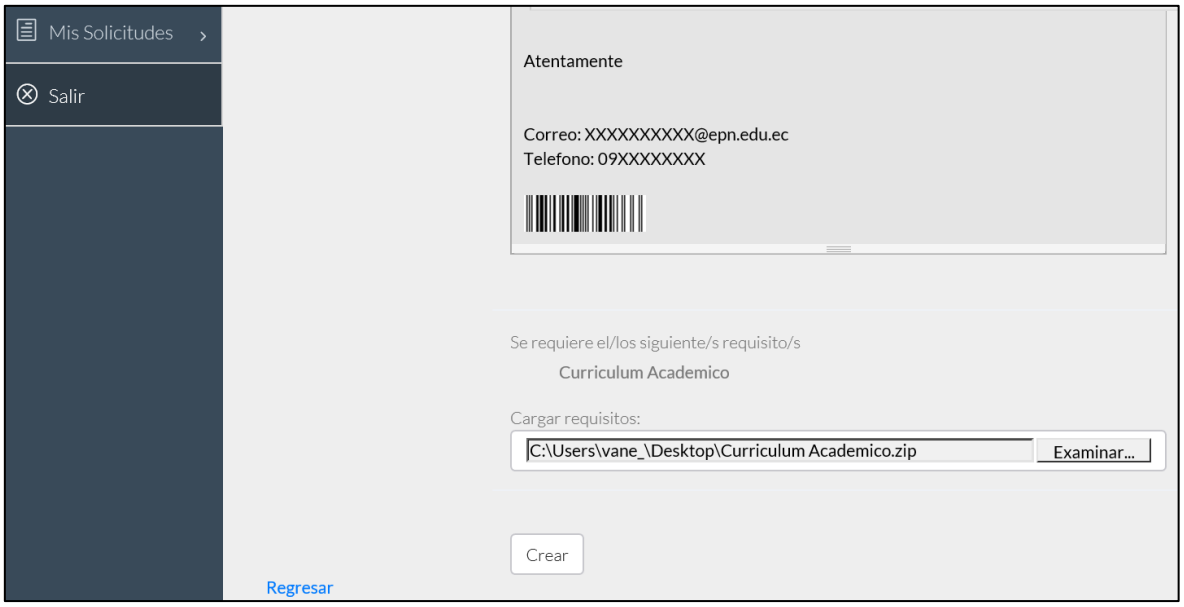


Figura 3.20 Crear una solicitud con plantilla (Parte 2 de 2)

En la Figura 3.21 se muestran todas las solicitudes con plantilla definida que se han creado, y la opción de los requisitos se puede descargar para verificación de que cumpla con lo pedido. La solicitud también se genera en formato PDF, esta se la puede descargar seleccionado *Detalles*, se cargará los datos de la solicitud y la opción *Download* como se muestra en la Figura 3.22.

Figura 3.21 Lista de solicitudes con plantilla creadas por el Estudiante

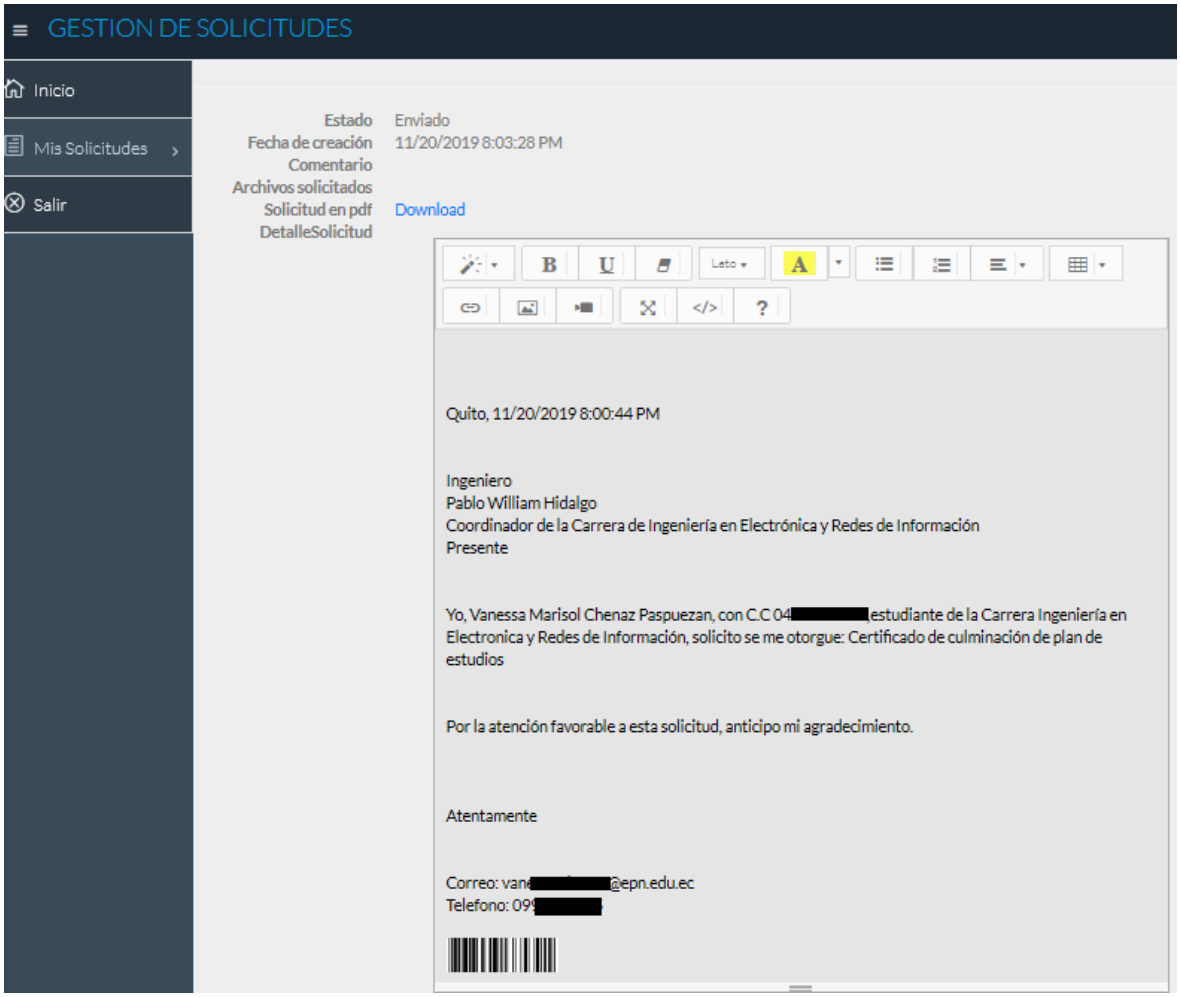


Figura 3.22 Página de detalles de la Solicitud

3.2.5 GENERACIÓN DE SOLICITUDES SIN PLANTILLA

Para generar una solicitud sin plantilla se debe seguir los mismos pasos que se siguió para crear una solicitud con plantilla, seleccionar: Mis Solicitudes → Sin plantillas → Crear nueva solicitud → Seleccionar la autoridad a la que va dirigida la solicitud, la aplicación le autocompleta datos como la fecha, autoridad según lo que haya seleccionado, y datos personales como nombres, apellidos, cédula, correo electrónico y teléfono. Lo que el estudiante debe llenar es el pedido que desea hacer.

Como se muestra en la Figura 3.23, el estudiante ha llenado información que desea y crea la solicitud, este tipo de solicitudes no cargan requisitos debido a que son solicitudes que el estudiante crea de su propia autoría.

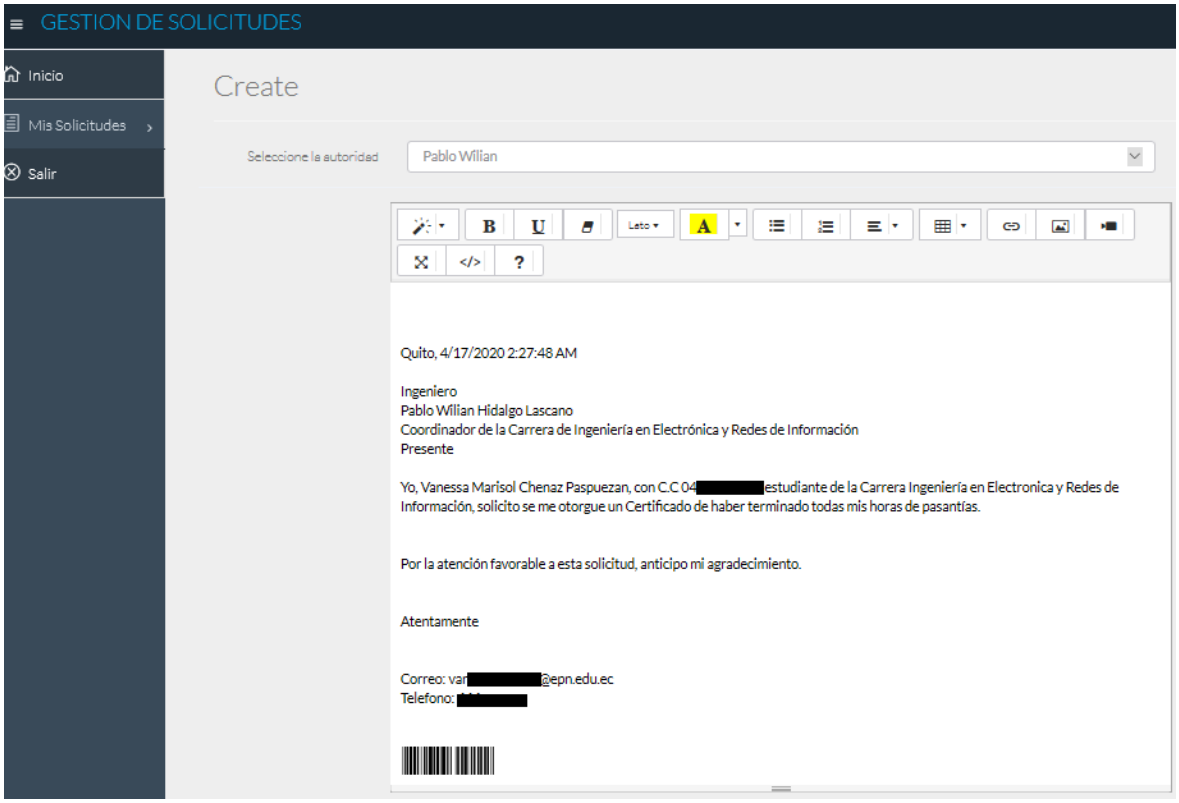


Figura 3.23 Creación de una solicitud sin plantilla

En la Figura 3.24, se despliega un listado de solicitudes sin plantilla, al igual que las solicitudes con plantilla de la Figura 3.21, si se selecciona la opción Detalles se muestra la solicitud creada y con la opción de descargar la solicitud en formato .pdf.

Cuando se crean las solicitudes con plantilla definida o sin plantilla definida, se crean con el estado de solicitud en Enviado.

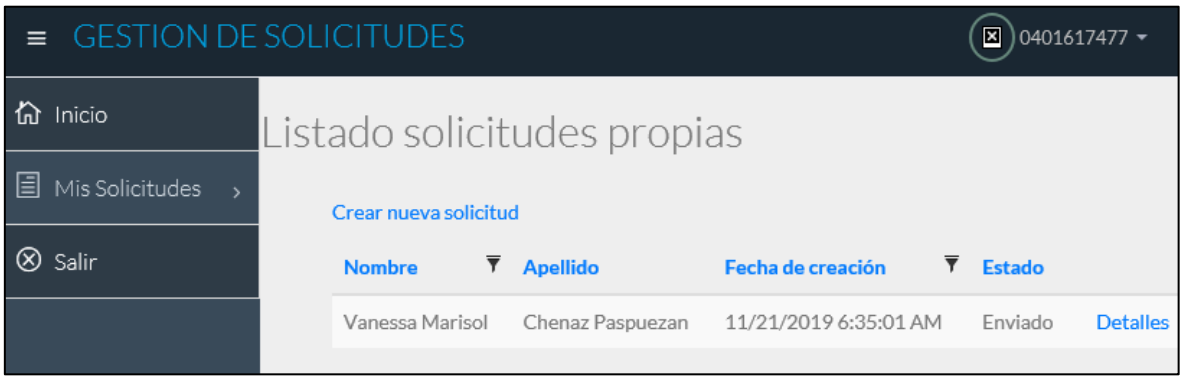


Figura 3.24 Listado de solicitudes sin plantilla

3.2.6 TRÁMITE A UNA SOLICITUD CON PLANTILLA

Para el respectivo trámite de una solicitud con plantilla, se debe pasar por la revisión primero por parte del personal de secretaría y si el Secretario acepta la solicitud, ésta pasa a ser tramitada por el Coordinador.

3.2.6.1 TRÁMITE DADO POR EL SECRETARIO

Se procede a acceder a la aplicación para ver las distintas solicitudes que le han llegado con o sin plantilla definida como muestra en la Figura 3.25, y así darles el trámite respectivo

Se debe seleccionar: Mis Solicitudes → seleccionar la solicitud que va a procesar y dar *click* en Tramitar, en este caso una solicitud con plantilla definida.

A continuación, se muestra el detalle de la solicitud que se está tramitando como se observa en la Figura 3.26, junto con los requisitos que esta solicitud ha pedido que sean cargados. El Secretario revisa el detalle de la solicitud y verifica que los datos y que los requisitos que han sido adjuntados son correctos. Se procede al cambio de estado de Enviado a Aceptado pues cumple con lo necesario y se guarda este trámite.

Después de haber guardado los cambios en el trámite de la solicitud, al Estudiante le llegará un correo electrónico como se indica en la Figura 3.28, indicándole que el estado de su solicitud ha sido actualizado y que deberá revisar la aplicación para verificar cómo va el proceso de su solicitud.

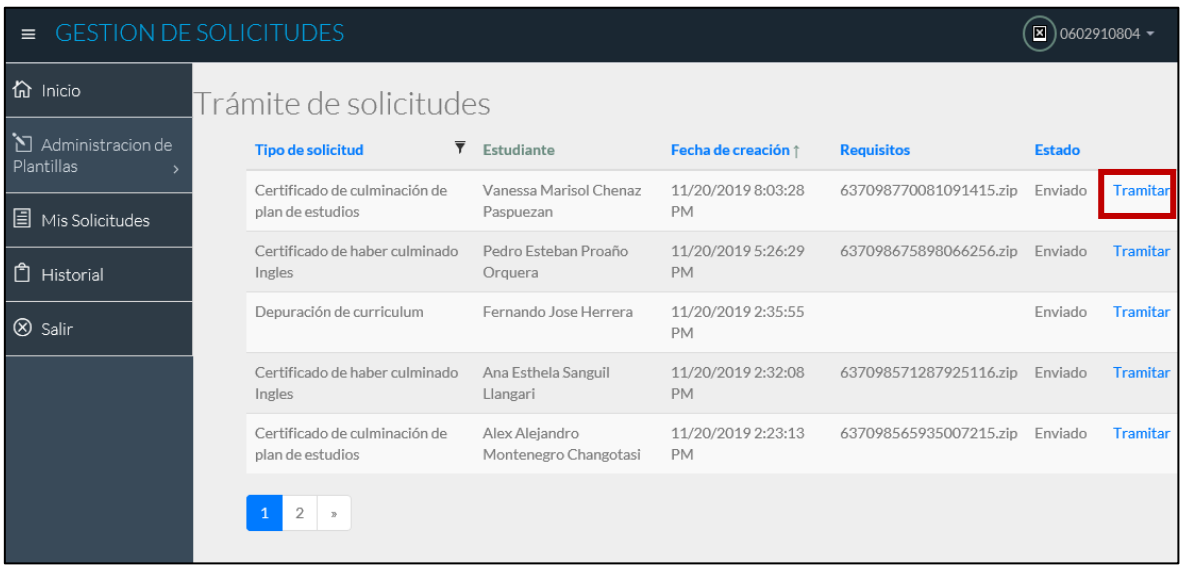


Figura 3.25 Solicitudes que han llegado a la aplicación

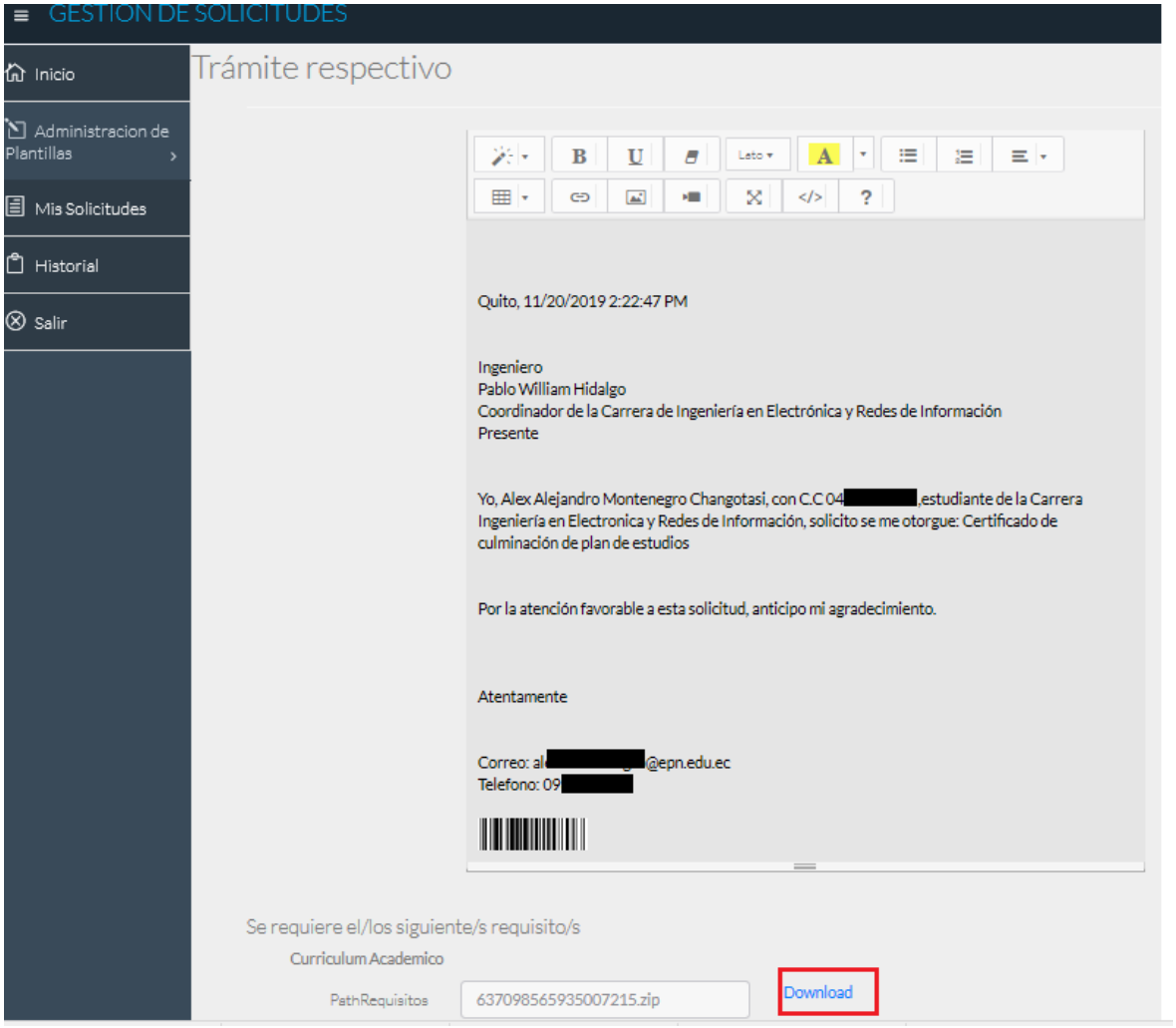


Figura 3.26 Trámite del Secretario (Parte 1 de 2)

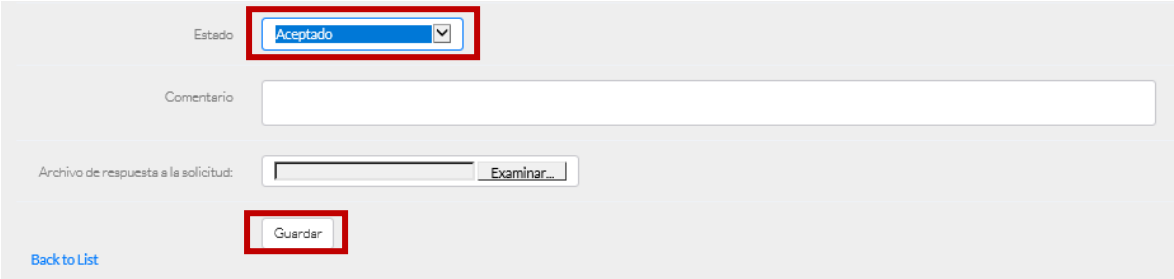


Figura 3.27 Trámite del Secretario (Parte 2 de 2)

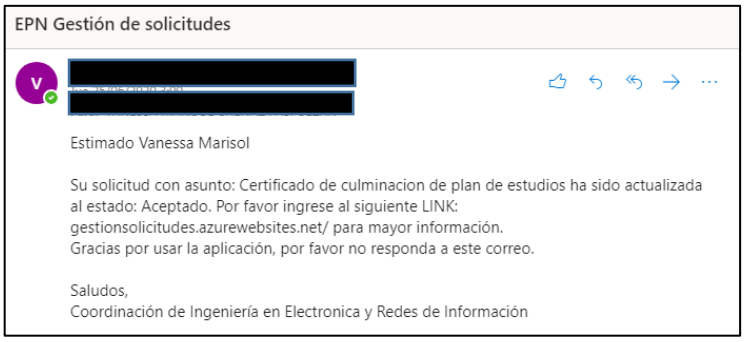


Figura 3.28 Confirmación de correo electrónico

3.2.6.2 TRÁMITE DADO POR EL COORDINADOR

Las solicitudes que le llegan al Coordinador son las que tienen el estado en Aceptado, el Coordinador selecciona la opción tramitar y procede a procesar la información como se ve en la Figura 3.29, la opción de descarga para los requisitos es opcional ya que antes ya pasó por la revisión del personal de Secretaría.

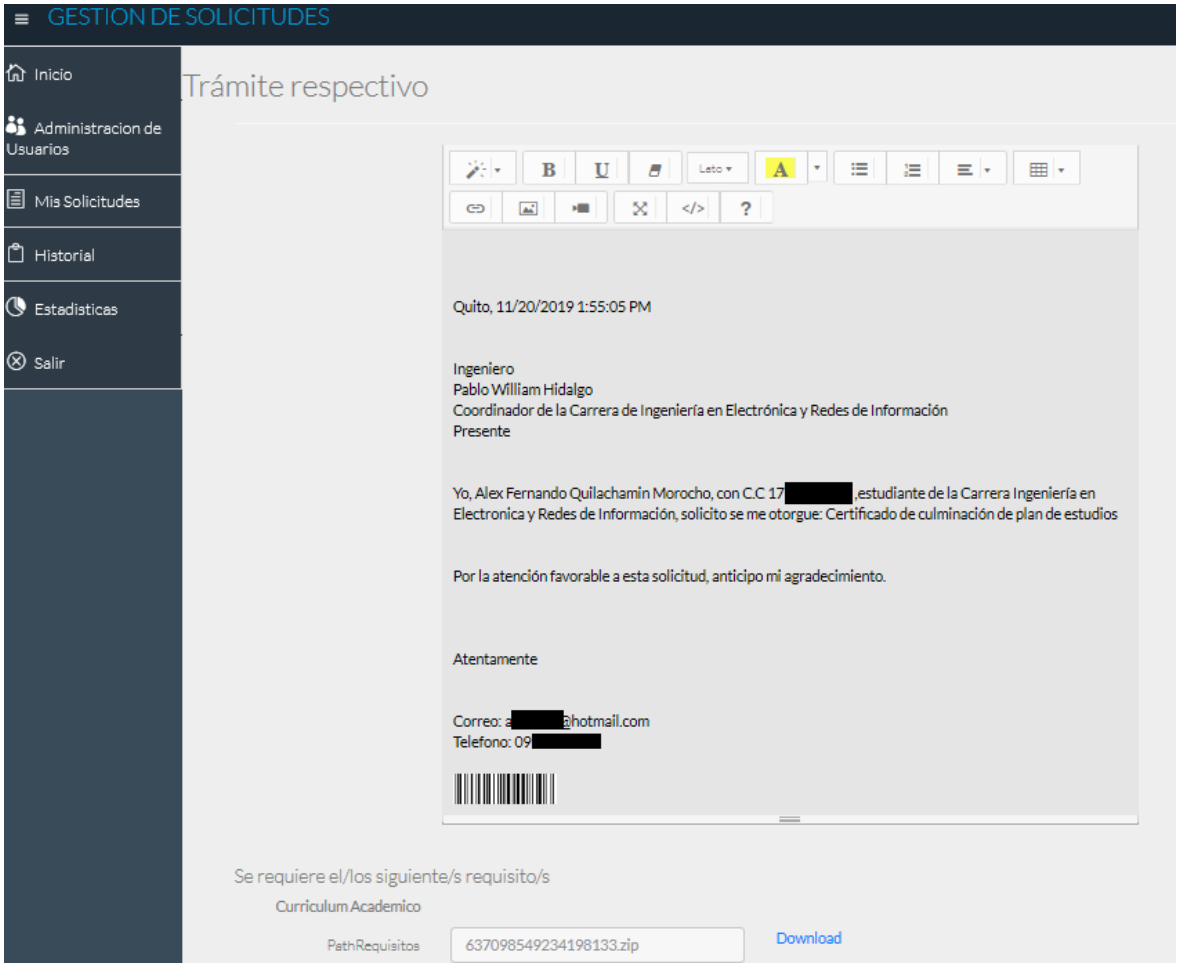


Figura 3.29 Trámite del Coordinador (Parte 1 de 2)

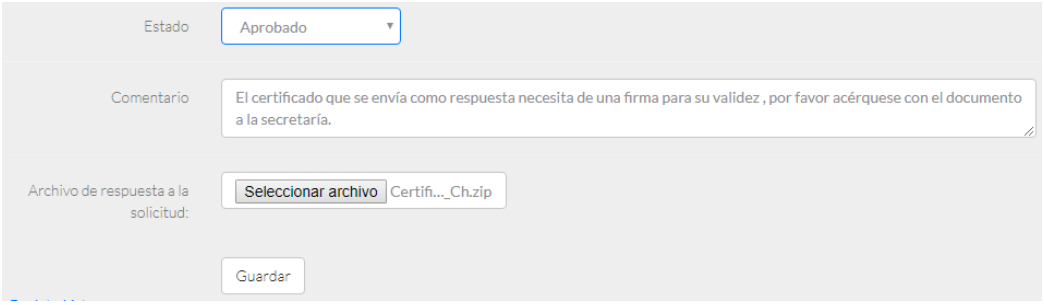


Figura 3.30 Trámite del Coordinador (Parte 2 de 2)

En la Figura 3.3027 parte 2, se puede observar que el Coordinador aprueba el pedido del Estudiante, le adjunta un archivo de la respuesta y se le pide al Estudiante que se acerque a obtener la firma del documento para validez del Certificado. Como hay un cambio de estado, el Estudiante recibirá nuevamente un correo indicándole lo mismo que en la Figura 3.28.

3.2.7 TRÁMITE A UNA SOLICITUD SIN PLANTILLA

Para el trámite que se les da a las solicitudes sin plantilla no se requiere verificar requisitos ya que éstas son de autoría propia del estudiante, el Secretario revisará datos personales y el pedido para dar la respuesta necesaria y luego pasa a ser tramitada por el Coordinador.

3.2.7.1 TRÁMITE DADO POR PARTE DEL SECRETARIO

El proceso es similar al que se les da a las solicitudes con plantilla, la diferencia es que en estas solicitudes no se requieren requisitos.

En la Figura 3.31, se muestra la respuesta a un pedido de solicitud, en este caso la solicitud ha sido rechazada y el comentario del por qué se detalla en la Figura 3.3228 parte 2, la actualización de la nueva información.

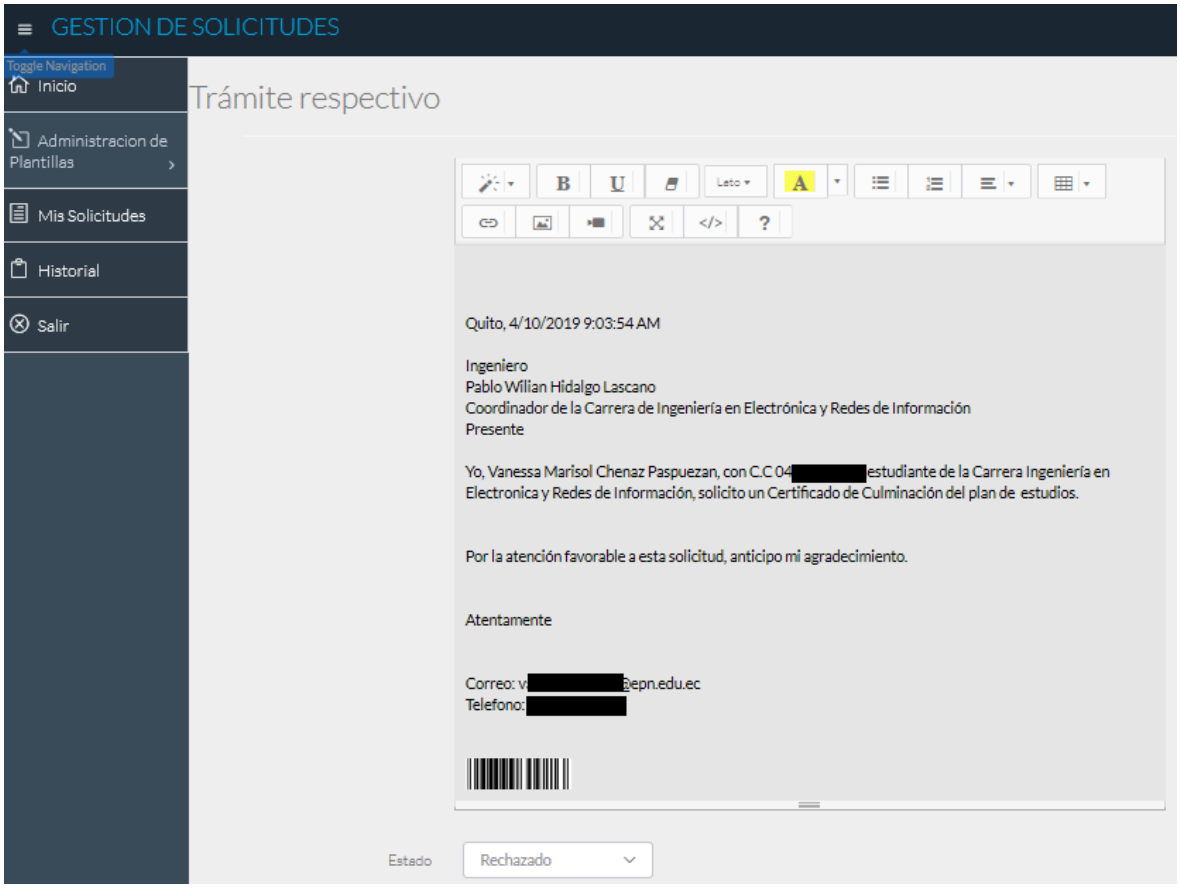


Figura 3.31 Trámite por parte del Secretario estado Rechazado (Parte 1 de 2)

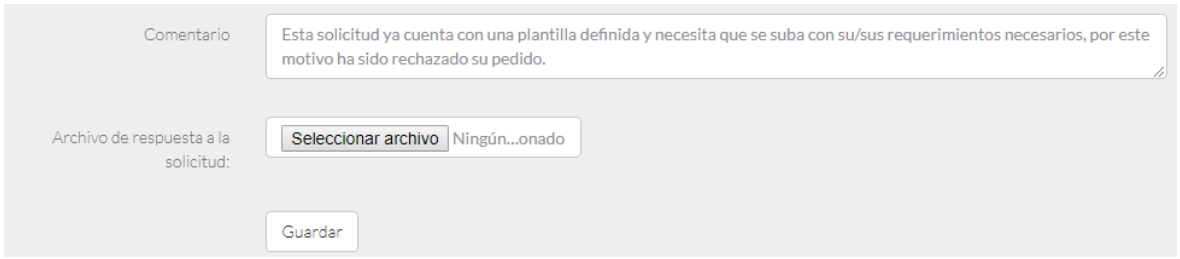


Figura 3.32 Trámite por parte del Secretario estado Rechazado (Parte 2 de 2)

La Figura 3.33 corresponde al resultado de la consulta realizada a la tabla Solicitud y Usuario, para comprobar que se ha guardado el estado de Rechazado, el comentario y obtener información del usuario a quien se le va a enviar el correo electrónico de actualización de estado de su solicitud como se ve en la Figura 3.34.

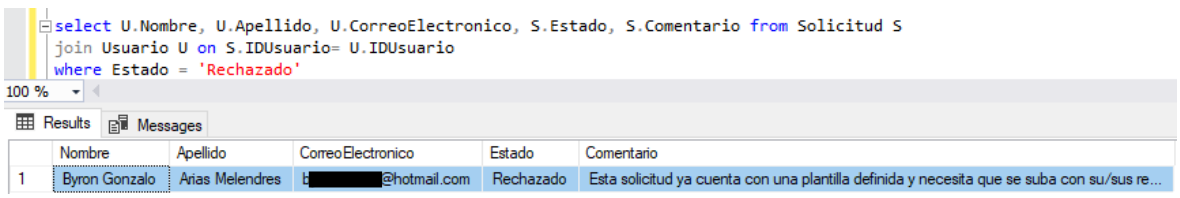


Figura 3.33 Registro de la Tabla Solicitud después de la respuesta

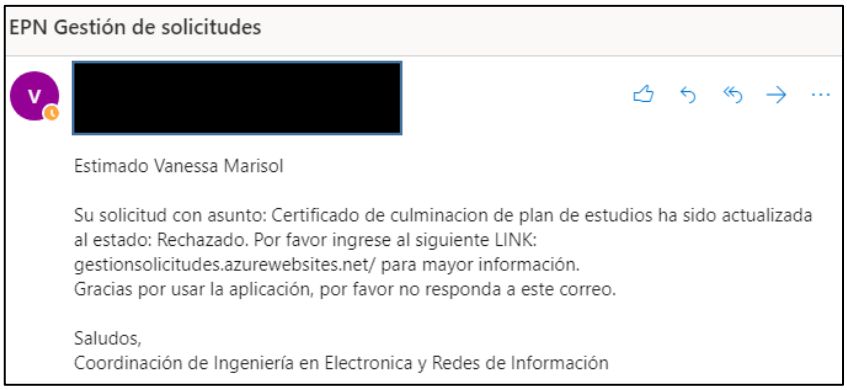


Figura 3.34 Correo electrónico por actualización de estado

En la Figura 3.35, se muestra que el Secretario ha comprobado que la solicitud cumple con los datos necesarios y procede a aceptar la solicitud.

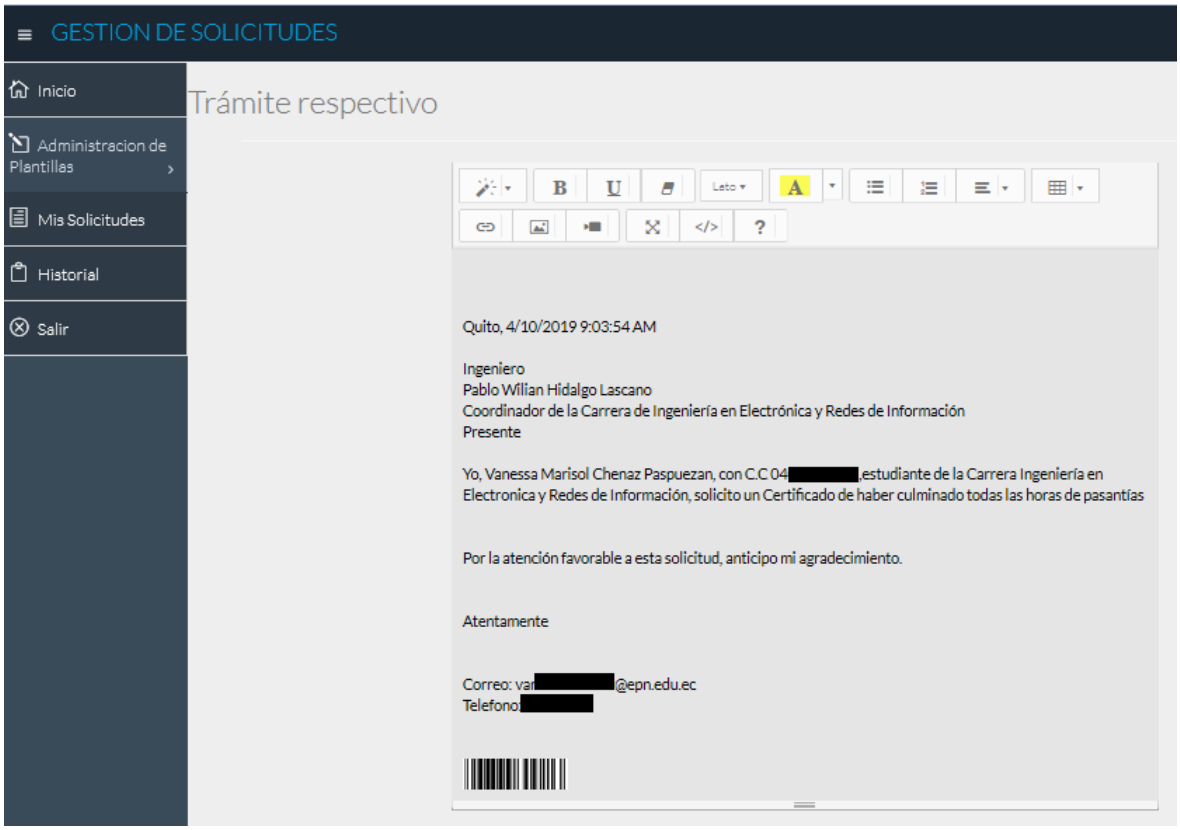


Figura 3.35 Trámite por parte del Secretario estado Aceptado (Parte 1 de 2)

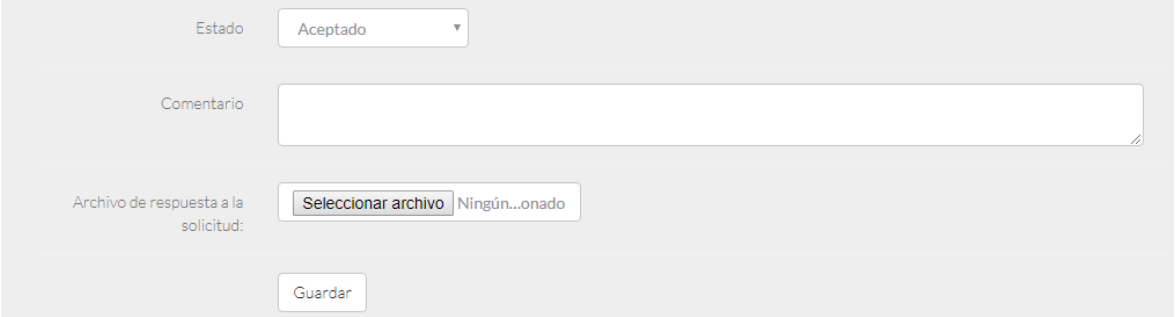


Figura 3.36 Trámite por parte del Secretario estado Aceptado (Parte 2 de 2)

En la Figura 3.37, se muestra cómo se actualizó el estado de la solicitud tramitada, el correo electrónico será el mismo que el de la Figura 3.28.

```
select Usuario.Nombre, Usuario.Apellido, Solicitud.Estado from Usuario
join Solicitud on Solicitud.IDUsuario = Usuario.IDUsuario
```

100 %

Results | Messages

	Nombre	Apellido	Estado
17	Vanessa Marisol	Chenaz Paspuezan	Aceptado
18	Alexander	Acero	Enviado
19	Alexander	Acero	Enviado
20	Marcos Andres	Alcocer Bahamonde	Enviado

Figura 3.37 Consulta para comprobar el cambio de estado

3.2.7.2 TRÁMITE DADO POR PARTE DEL COORDINADOR

El trámite dado por el Coordinador para el rechazo de una solicitud será igual a como tramita un rechazo el Secretario, por lo que en esta parte se verá una validación en la parte de subir archivos de respuesta al estudiante cuando su solicitud ha sido aceptada.

En la Figura 3.38, se muestra en la página del Coordinador las solicitudes que están con el estado de Aceptado para ser tramitadas.

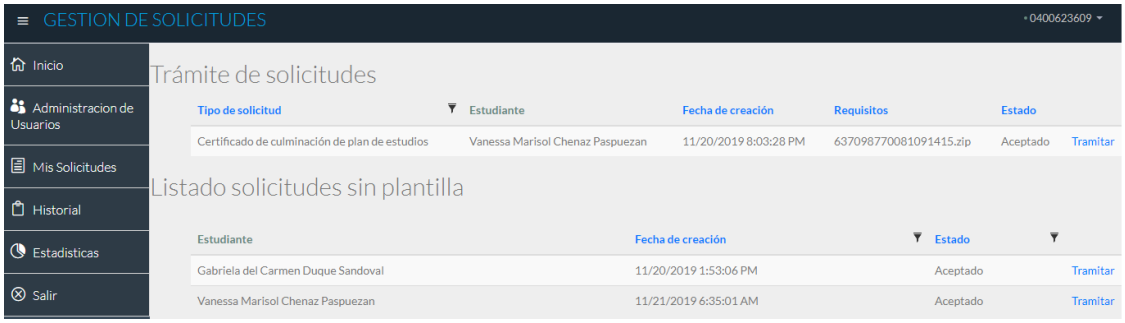


Figura 3.38 Trámite a una solicitud sin plantilla

Ahora se tramita una solicitud sin plantilla y como se muestra en la Figura 3.39, en la cual se aprueba el pedido y se debe subir el archivo de respuesta, para validar los campos se sube un archivo en formato .pdf en la Figura 3.40 y se ve que muestra el mensaje de error ya que se permite solo la subida de archivos en formato .zip o .rar.

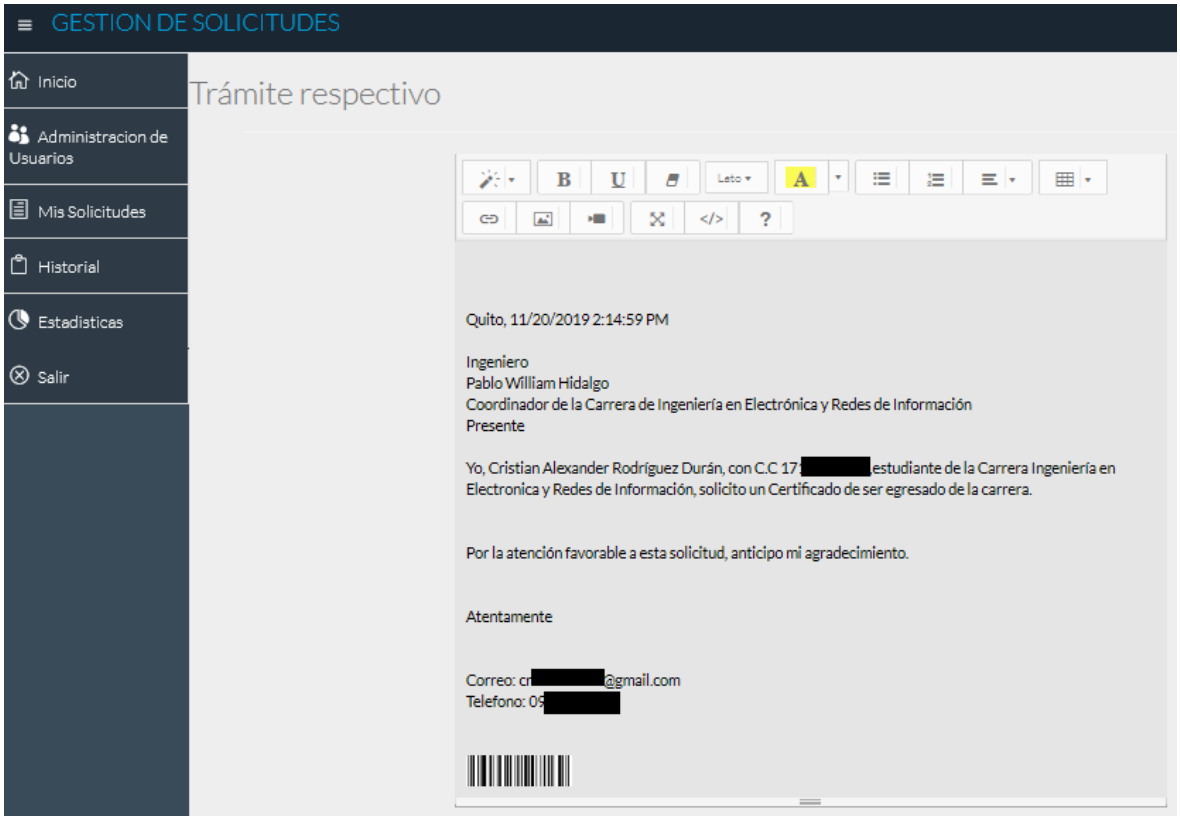


Figura 3.39 Detalle de la solicitud sin plantilla

Figura 3.40 Carga incorrecta de archivo

En la figura Figura 3.41, se muestra la manera correcta del formato subido como respuesta, y entonces se procede a guardar la información, este cambio se comprueba en la Figura 3.42, y el correo electrónico sigue llegando de igual manera que el de la Figura 3.28.

Figura 3.41 Carga correcta de archivo

```
select Usuario.Nombre, Usuario.Apellido, Solicitud.Estado, Solicitud.Comentario from Usuario
join Solicitud on Solicitud.IDUsuario = Usuario.IDUsuario
where Solicitud.Estado = 'Aprobado'
```

esults | Messages

Nombre	Apellido	Estado	Comentario
Vanessa Marisol	Chenaz Paspuezan	Aprobado	Por favor acérquese a la secretaría para que su certificado tenga un sello de validación.

Figura 3.42 Respuesta de aprobación por el Coordinador

3.2.8 RESPUESTA AL ESTUDIANTE EN LA APLICACIÓN

El estudiante ingresa a la aplicación como se muestra en la Figura 3.43 y revisa el estado de su solicitud, para verificar la información que ha pedido selecciona la opción Detalles y se le desplegara la información de la Figura 3.44, se muestran dos opciones de descarga, la primera es para descargar la respuesta dada por el Coordinador y la segunda es en caso de que el Estudiante quiera descargar el detalle de su solicitud en formato .pdf.

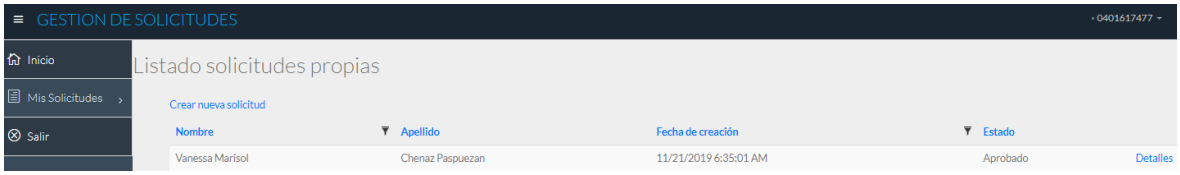


Figura 3.43 Solicitud aprobada al estudiante

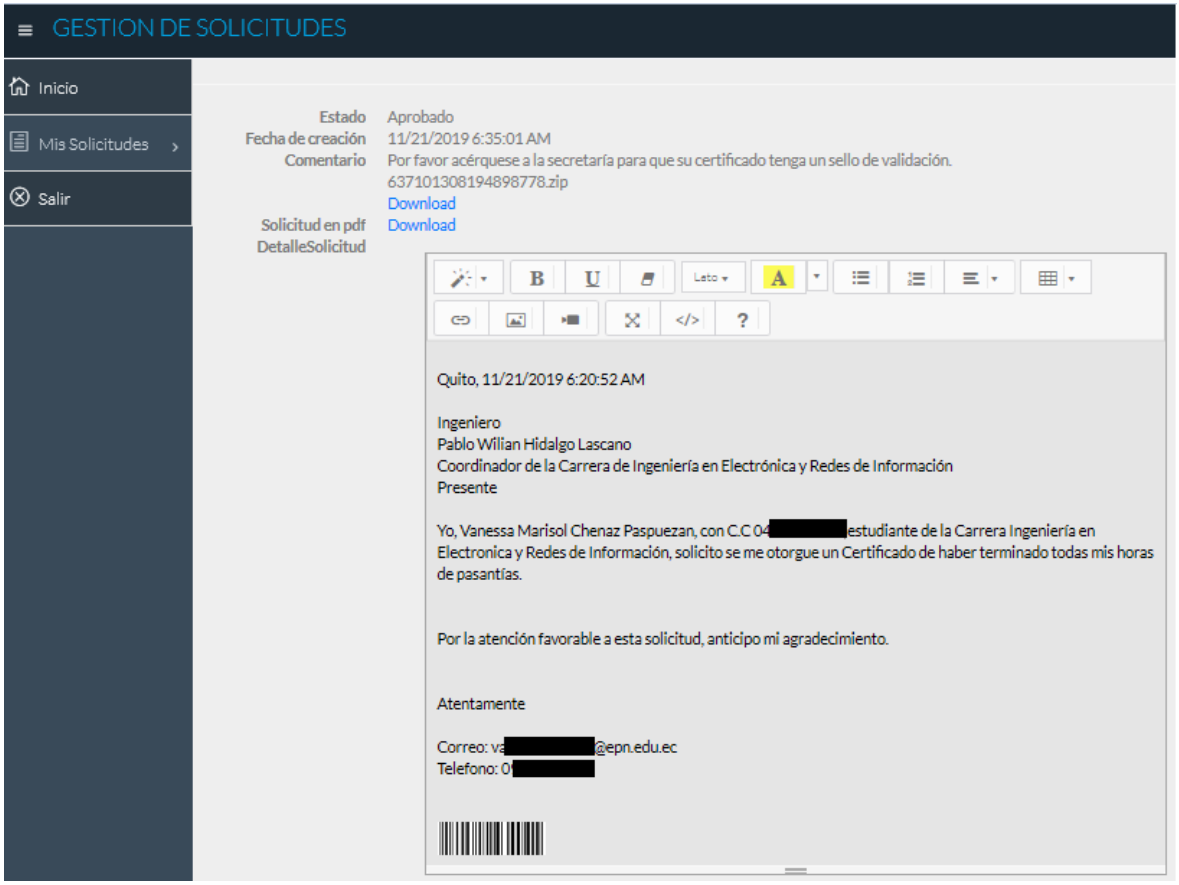


Figura 3.44 Detalles de la solicitud aprobada

3.2.9 VISUALIZACIÓN DE ESTADÍSTICAS DE LAS SOLICITUDES

Esta opción está disponible para el usuario con el rol de Coordinador, el cual puede ver estadísticas de cuántas solicitudes han sido aceptadas, es decir haber cumplido con los requisitos que verifica el Secretario, cuántas han sido rechazadas y cuántas aprobadas.

Las estadísticas tienen la opción de ser escogidas en función del tiempo como en la Figura 3.45.

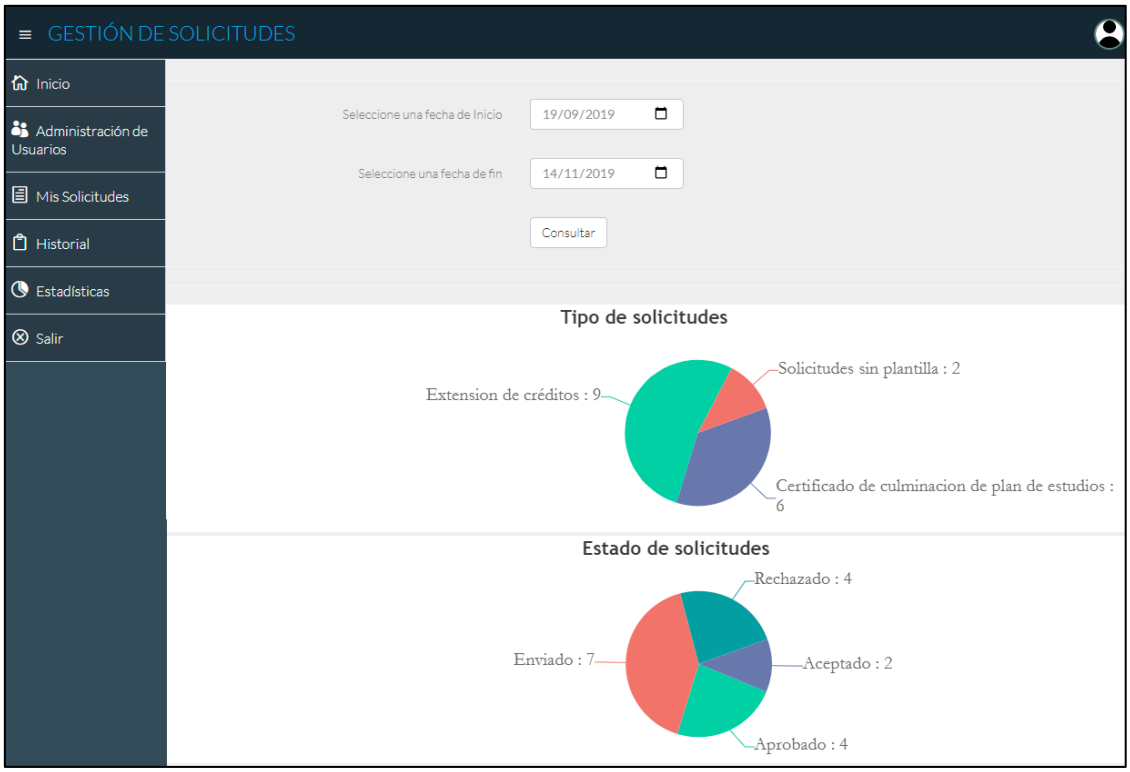


Figura 3.45 Estadísticas de las solicitudes

```
select Usuario.Nombre, Usuario.Apellido, Solicitud.Estado, Solicitud.Comentario from Usuario
join Solicitud on Solicitud.IDUsuario = Usuario.IDUsuario
ORDER BY Solicitud.Estado;
```

100 %

Results | Messages

	Nombre	Apellido	Estado	Comentario
1	Gabriela del Carmen	Duque Sandoval	Aceptado	NULL
2	Vanessa Marisol	Chenaz Paspuezan	Aceptado	NULL
3	Vanessa Marisol	Chenaz Paspuezan	Aprobado	Por favor acérquese a la secretaría para que su ...
4	Alexander	Acero	Enviado	NULL
5	Alexander	Acero	Enviado	NULL
6	Marcos Andres	Alcocer Bahamonde	Enviado	NULL
7	Marcos Andres	Alcocer Bahamonde	Enviado	NULL
8	Byron Gonzalo	Arias Melendres	Enviado	NULL
9	Ernesto Santiago	Gangotena Torres	Enviado	NULL
10	Gabriela del Carmen	Duque Sandoval	Enviado	NULL
11	Alex Fernando	Quilachamin Morocho	Enviado	NULL
12	Willian Javier	Viracocha Verdezoto	Enviado	NULL
13	Cristian Alexander	Rodríguez Durán	Enviado	NULL
14	Maria Belen	Cuesta Plua	Enviado	NULL
15	Alex Alejandro	Montenegro Changotasi	Enviado	NULL
16	Ana Esthela	Sanguil Llangari	Enviado	NULL
17	Fernando Jose	Herrera	Enviado	NULL
18	Ronald Andres	Lema Andrango	Enviado	NULL
19	Pedro Esteban	Proaño Orquera	Enviado	NULL
20	Pedro Esteban	Proaño Orquera	Enviado	NULL
21	Byron Gonzalo	Arias Melendres	Rechazado	Esta solicitud ya cuenta con una plantilla definida...

Figura 3.46 Consulta para comprobar el gráfico de estadísticas

3.3 PRUEBAS DE VALIDACIÓN

Se aplicaron 3 tipos de entrevistas: Para el Coordinador, el Secretario y a 15 estudiantes de la carrera de Ingeniería en Electrónica y Redes de Información. Los formatos de encuestas se encuentran en el ANEXO F.

A continuación se muestran los resultados obtenidos de las encuestas al Coordinador en la Tabla 3.2 y al Secretario en la Tabla 3.3.

Tabla 3.2 Respuestas de la encuesta al Coordinador

Pregunta	Respuesta Coordinador
1. ¿Por favor acceda a la aplicación con el usuario = 0400623609 y la contraseña = admin12345, es posible?	SI
2. ¿Ingrese a la opción Mi perfil y revise su información, es ésta correcta?	SI
3. ¿Cuándo accedió a editar su perfil, su nueva información se guardó de manera correcta?	SI
4. ¿La aplicación permite hacer un cambio en la contraseña de acceso a la aplicación?	SI
5. ¿La aplicación permite crear un nuevo usuario con el rol de Secretario?	SI
6. ¿La aplicación permite ver todos los usuarios que tiene la aplicación?	SI
7. ¿La aplicación permite dar trámite a las solicitudes que han llegado a la aplicación?	SI
8. ¿La aplicación permite descargar los requisitos si estos existen, que se han adjuntado a las solicitudes con plantilla definida?	SI
9. ¿La aplicación permite cambiar el estado de una solicitud?	SI
10. ¿La aplicación permite cargar archivos de respuesta en la aplicación?	SI
11. ¿La aplicación permite descargar la solicitud en formato .pdf?	SI
12. ¿La aplicación permite ver un historial de todas las solicitudes que llegaron a la aplicación?	SI
13. ¿La aplicación permite ver el gráfico de estadísticas de las solicitudes aceptadas, aprobadas y rechazadas?	SI

Tabla 3.3 Respuestas de la encuesta al Secretario

Pregunta	Respuesta Secretario
1. ¿Por favor acceda a la aplicación con el usuario = 0602910804 y la contraseña = Secretario1, es posible?	SI
2. ¿Ingrese a la opción Mi perfil y revise su información, es ésta correcta?	SI
3. ¿Cuándo accedió a editar su perfil, su nueva información se guardó de manera correcta?	SI
4. ¿La aplicación le permitió hacer un cambio en la contraseña de acceso a la aplicación?	SI
5. ¿La aplicación crear plantillas nuevas para nuevos tipos de solicitudes?	SI
6. ¿La aplicación le permitió editar la información de plantillas nuevas para nuevos tipos de solicitudes?	SI
7. ¿La aplicación permite dar trámite a las solicitudes que han llegado a la aplicación?	SI
8. ¿La aplicación permite descargar los requisitos si estos existen, que se han adjuntado a las solicitudes con plantilla definida?	SI
9. ¿La aplicación permite cambiar el estado de una solicitud?	SI
10. ¿La aplicación permite cargar archivos de respuesta en la aplicación?	SI
11. ¿La aplicación permite descargar la solicitud en formato .pdf?	SI
12. ¿La aplicación permite ver un historial de todas las solicitudes que llegaron a la aplicación?	SI

A continuación se muestran los resultados obtenidos de las encuestas a los 15 estudiantes.

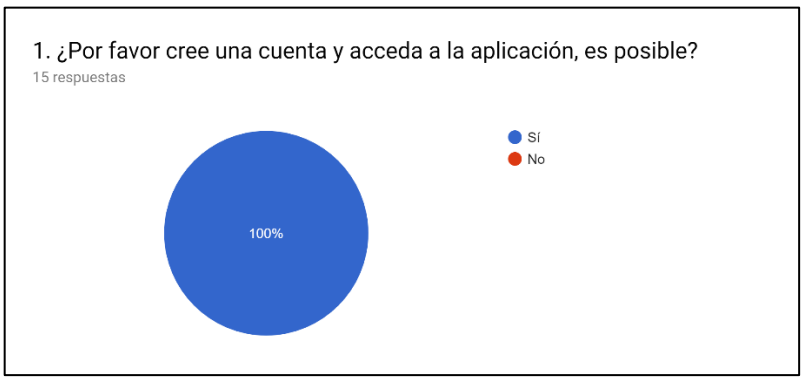


Figura 3.47 Resultados de la primera pregunta

Figura 3.48 Resultados de la segunda pregunta

Figura 3.49 Resultados de la tercera pregunta

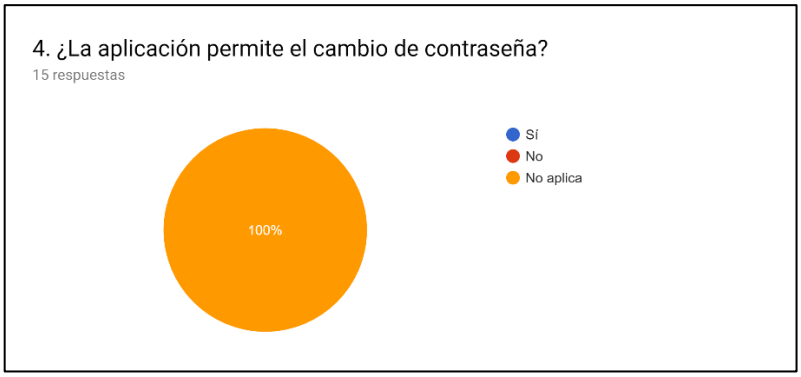


Figura 3.50 Resultados de la cuarta pregunta

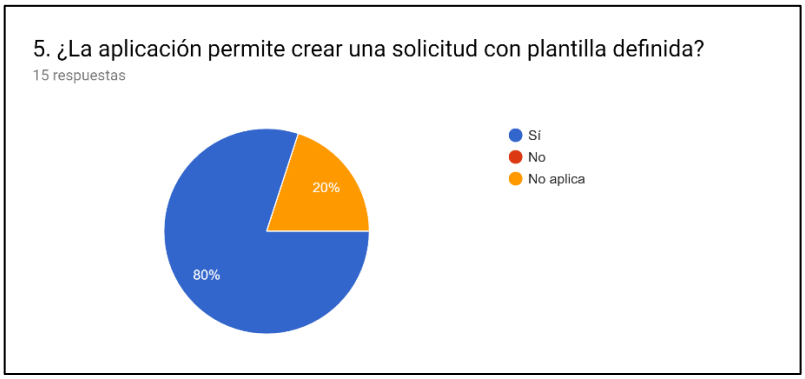


Figura 3.51 Resultados de la quinta pregunta

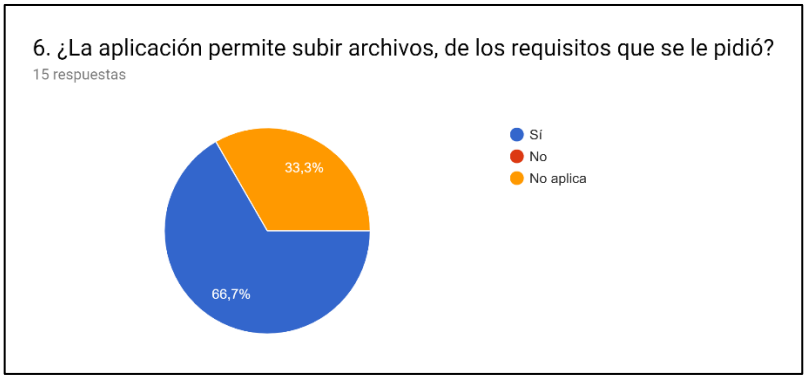


Figura 3.52 Resultados de la sexta pregunta

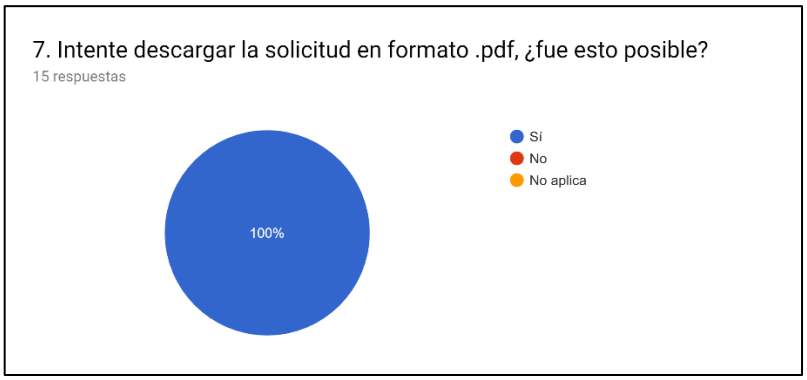


Figura 3.53 Resultados de la séptima pregunta

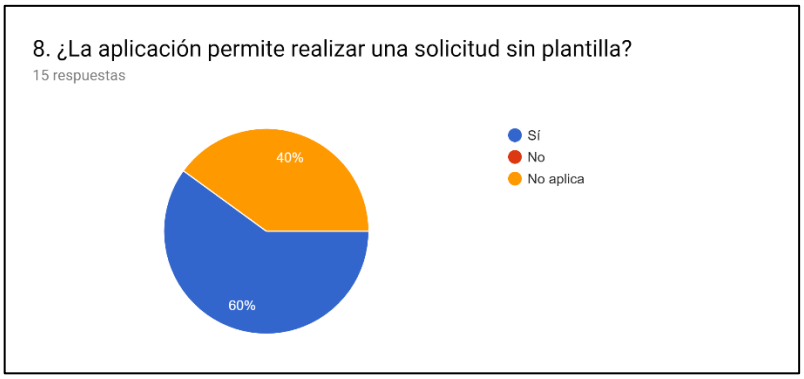


Figura 3.54 Resultados de la octava pregunta

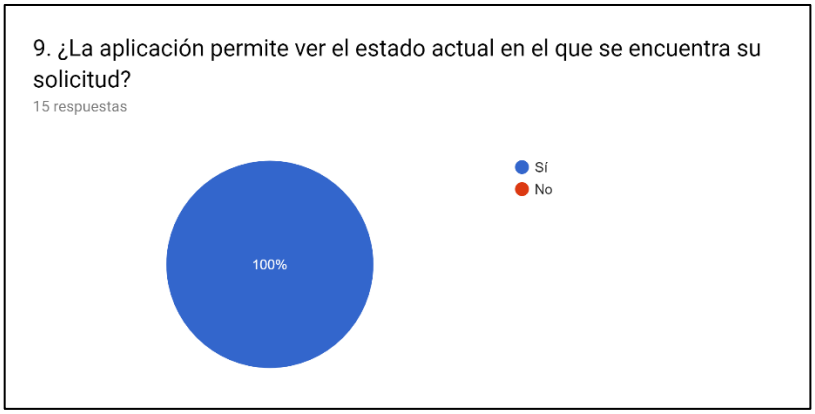


Figura 3.55 Resultados de la novena pregunta

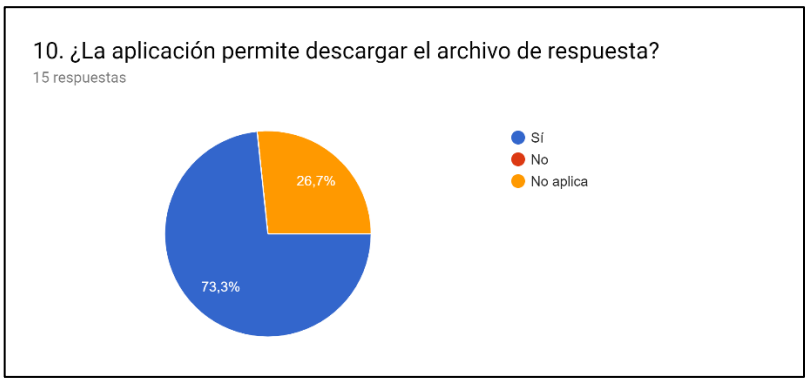


Figura 3.56 Resultados de la décima pregunta

Los casos en los que los estudiantes respondieron con la opción no aplica, significa que no realizaron ese proceso como por ejemplo la pregunta de la Figura 3.50, ninguno realizó el cambio, pero se puede confirmar que el método para cambiar contraseña funciona ya que se le hizo probar al usuario con rol Secretario y al usuario con rol Coordinador, y el cambio fue exitoso.

3.4 CORRECCIÓN DE ERRORES

El primer error que surgió fue cuando se escogió *Code First* como método para crear el modelo de entidad, al momento de que surgía algún cambio en el modelo se tenian que habilitar las migraciones para que estos cambios se realicen en Visual Studio y muchas veces la base de datos no se actualizaba de la mejor manera, provocando que encontrar la fuente del error fuera demasiado complicado, por eso mejor se optó por usar DataBase First ya que generar una base de datos en SQL es mas simple y los cambios se hacen de manera automatica sin afectar el desarrollo del proyecto.

El uso de *Data Annotations* en el Modelo generado por *Entity Framework* provocaba que en cada cambio de la base de datos estos atributos se borraran y se tuvieran que colocar de nuevo en el Modelo que se las requería; esto se pudo resolver haciendo uso de las clases parciales ya que en estas clases se pueden añadir los nuevos cambios que se hayan hecho en la base de datos sin afectar el Modelo generado.

MVC no tiene un control *Grid* en los *Helpers* por lo que se agregó una librería (Grid.Mvc) para poder hacer uso de este control, si bien no afectaba a la funcionalidad de la aplicación, sirve de mucho para tener una mejor presentación de la aplicación.

Se tuvo problemas con el control de texto enriquecido (summernote.editor) al momento de el cambio de los estilos del layout, puesto que la tecla espaciadora se la tenía configurada para desplazar la página; el problema se ocasionaba cuando el Estudiante quería redactar de manera personal su solicitud y no se le permitía dar espacio entre palabras, para este problema la solución fue desactivar esta opción *spacebarenabled:false*

Otro de los problemas relacionados con el control de texto enriquecido era que como se desactivó la opción *spacebarenabled* ésta ya no permitía desplazar la página y no se podria acceder al botón Crear solicitud que estaba en la parte de abajo, para este problema la solución fue deshabilitar la opcion *enablemousewheel:false*

Uno de los problemas que surgió cuando se publicó la aplicación web en Azure fue que Azure no permitia el uso de correo electronico con credenciales de gmail para el envío de correos, la solución fue crear el método para envío de correo con el uso de la cuenta outlook XXX@epn.edu.ec

Al inicio la aplicación estaba guardando la contraseña del usuario en texto plano, en la aplicación esta información no se cargaba, pero si se revisaba en la base de datos con la respectiva consulta se podía obtener esta información, para lo cual por motivos de seguridad se decidió usar encriptación especificamente un *hash*.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- En el presente trabajo de titulación se construyó un sistema de gestión de solicitudes para la coordinación de la carrera de Ingeniería en Electrónica y Redes de información. El sistema es una aplicación web con el uso de una base de datos transaccional y la utilización de un servidor de correo.

- El diseño y la implementación de los módulos fue desarrollado con una arquitectura de 3 capas: capa de datos, capa de negocio y capa de presentación. Para el almacenamiento de la contraseña se utilizó encriptación. Además, se agregaron campos para manejar la auditoría del registro.

- Las pruebas unitarias fueron realizadas utilizando el *framework* de Microsoft y con la ayuda de objetos falsos, la funcionalidad de la aplicación fue protegida con el uso de pruebas unitarias, éstas permiten probar la unidad mínima de código.

- Se realizaron pruebas de validación e integración del sistema con estudiantes, coordinador y secretario de la facultad. Los usuarios identificaron detalles en el sistema que podrían ser mejorados como: mezcla del idioma (español - inglés), falta de mensajes informativos al realizar acciones en el sistema, la recarga de información es forzada por el usuario no es automática.

- El envío de *mail* para la confirmación al estudiante de las acciones que van pasando en la solicitud fue implementado utilizando Outlook 365 con la cuenta de la universidad ya que daba más facilidades al momento de ser desplegado en Azure, contra la implementación en desarrollo en donde se utilizó Gmail como servidor de correo.

- La metodología de desarrollo utilizada en este proyecto fue Scrum para de esta forma avanzar en módulos pequeños y obtener retroalimentación del usuario final en este caso: secretario, coordinador y estudiantes. Esta flexibilidad permitió obtener cambios no identificados al inicio del proyecto como la generación del archivo en formato PDF, mejorar la visualización de la aplicación, manejo de

campos de auditoria que registran el usuario que creó y el último usuario que modificó el registro.

- Se utilizó un repositorio de código en la nube. Esto facilitó el uso de computadores en casa o la universidad para el desarrollo de la aplicación además ayudó a proteger el código y mantener las versiones del código que se iban realizando. Cada vez que el código era almacenado en el repositorio de código se agregaba un comentario el cual describía la tarea realizada, se realizaron alrededor de quinientas subidas de código para esta aplicación.

- La creación de pruebas unitarias y de integración se realizaron para probar cada componente individualmente y la integración de éstos. En el caso de pruebas unitarias se generaron unas pruebas unitarias para evaluar el Controlador y evaluar la utilidad de estos. Para la creación de pruebas unitarias se necesitó conocer temas como inyección de dependencias y un *framework* de pruebas unitarias para crear objetos falsos.

4.2 RECOMENDACIONES

- Para un trabajo futuro se propone mejorar la auditoria de la aplicación agregando tablas históricas que almacenen los datos cuando son modificados o eliminados. Se podría utilizar disparadores en las tablas para identificar cuando un registro se modifica o elimina.

- La aplicación fue publicada en la nube para ser evaluada por los usuarios del sistema, pero se podría tomar ventaja de los servidores web de la universidad para ser publicada y de esta forma reduciría costos de espacio, disponibilidad y velocidad.

- No se realizaron pruebas de seguridad en el sistema, así que unas pruebas de penetración sobre el sistema, configuración mínima necesaria por usuario y reducción de la superficie de ataque contribuiría a mejorar el sistema no solo en calidad sino también en temas de seguridad.

- Un último detalle identificado en el desarrollo del sistema fue que las pruebas de regresión fueron necesarias durante la ejecución del proyecto. Cada mejora o nueva funcionalidad al producto desarrollado tenía que verificar la funcionalidad ya entregada. La automatización de estas pruebas ayudaría a reducir el tiempo de

pruebas y desgaste del desarrollador o probador de la aplicación ya que es una tarea repetitiva que puede ser automatizada.

- El uso del método Code First se recomienda para personas que tiene gran conocimiento en el área de programación, pero si se tiene mayores conocimientos en bases de datos se recomienda usar Database First, en este proyecto al inicio se usó Code First como una manera para aprender a manejar Base de Datos con MVC y una vez que se obtuvo la experiencia necesaria se decidió usar Database First ya que la implementación resulto ser más fácil y los cambios más rápidos.

- Se recomienda el uso de TFS (Team Fundación Server) , el TFS sirve para proteger el código que el desarrollador cree que ya está listo de esta forma no se pierde el código que ya se había avanzado, una vez protegido se puede proceder a desarrollar nuevo código, en caso de que el nuevo código que se está desarrollando se dañara se puede usar la opción deshacer cambios y empezar otra vez desde donde la aplicación funcionaba correctamente así el trabajo realizado no se ve afectado como cuando no se tiene este tipo de respaldo, en este proyecto cada avance era protegido en el TFS, hubo cambios que dañaban el funcionamiento del proyecto pero con la opción deshacer se mantenía la última versión útil del mismo.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] D. Nations, «lifewire,» 14 november 2019. [En línea]. Available: https://www.lifewire.com/what-is-a-web-application-3486637. [Último acceso: 5 noviembre 2019].

[2] A. Yaskevich, «DZone,» 17 Aug 2018. [En línea]. Available: https://dzone.com/articles/types-of-web-applications-from-a-static-web-page-t. [Último acceso: 6 noviembre 2019].

[3] «keycdn,» 4 October 2018. [En línea]. Available: https://www.keycdn.com/support/difference-between-static-and-dynamic. [Último acceso: 2019].

[4] S. Spence, «stevespence,» 30 January 2019. [En línea]. Available: https://www.stevespence.net/courses/writing-for-digital-media/modules/dynamic-sites/dynamic-sites-slides.php. [Último acceso: 7 noviembre 2019].

[5] «WebReunidos,» 4 May 2012. [En línea]. Available: https://www.webreunidos.es/estandares-web/. [Último acceso: 2019].

[6] «TechTerms,» 1 June 2011. [En línea]. Available: https://techterms.com/definition/markup_language. [Último acceso: 2019].

[7] «Exes,» [En línea]. Available: https://www.mundolinux.info/que-es-xml.htm. [Último acceso: 21 octubre 2019].

[8] M. contributors, «MDN web docs,» 8 Nov 2019. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS#targetText=Cascading%20Style%20Sheets%20(CSS)%20is,speech%2C%20or%20on%20other%20media.. [Último acceso: Nov 16 2019].

[9] M. K. Patel, «HTML, CSS, Bootstrap, Javascript and jQuery,» 2018.

[10] M. contributors, «MDN web docs,» 8 Nov 2019. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Último acceso: 15 Nov 2019].

[11] A. Freeman, Pro jQuery, Apress, 2012.

[12] G. B., «Tutorial Hostinger,» 11 abril 2019. [En línea]. Available: https://www.hostinger.com.ar/tutoriales/insertar-javascript-en-html/. [Último acceso: 10 noviembre 2019].

[13] G. B., «Tutorial Hostinger,» 13 mayo 2019. [En línea]. Available: https://www.hostinger.com.ar/tutoriales/que-es-jquery/. [Último acceso: 10 noviembre 2019].

[14] A. Freeman, Pro ASP.NET MVC 5, Apress, 2013.

[15] «Microsoft,» 20 Abril 2011. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6. [Último acceso: 17 Noviembre 2019].

[16] J. Atwood, «Coding Horror,» 5 mayo 2008. [En línea]. Available: https://blog.codinghorror.com/understanding-model-view-controller/. [Último acceso: 13 octubre 2019].

[17] «w3schools.com,» [En línea]. Available: https://www.w3schools.com/asp/razor_intro.asp. [Último acceso: 17 octubre 2019].

[18] J. Ciliberti, ASP.NET MVC 4 RECIPES, Apress, 2013.

[19] «Microsoft,» 10 07 2008. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/views/creating-custom-html-helpers-cs. [Último acceso: 01 2019].

[20] J. R. Guay Paz, Beginning ASP.NET MVC 4, Apress, 2013.

[21] «TutorialsTeacher,» [En línea]. Available: https://www.tutorialsteacher.com/mvc/viewbag-in-asp.net-mvc. [Último acceso: 17 octubre 2019].

[22] «EcuRed,» [En línea]. Available: https://www.ecured.cu/Arquitectura_en_Capas. [Último acceso: 23 octubre 2019].

[23] L. E. Asanza Alvia, Desarrollo de un sistema distribuido basado en SOA y MVC para despliegue de información médica de pacientes, Quito: Escuela Politécnica Nacional, 2017.

[24] S. S. Blog, «Steve Sanderson,» 11 junio 2009. [En línea]. Available: https://blog.stevensanderson.com/2009/06/11/integration-testing-your-aspnet-mvc-application/##targetText=Integration%20tests%20test%20your%20entire,working%20together%20as%20you%20expected.. [Último acceso: 19 octubre 2019].

[25] M. Fowler, «martinFowler.com,» 16 January 2018. [En línea]. Available: https://martinfowler.com/bliki/IntegrationTest.html. [Último acceso: Enero 2020].

[26] A. Augustyn, «Encyclopedia britannica,» [En línea]. Available: https://www.britannica.com/technology/database.

[27] «TechTarget,» enero 2015. [En línea]. Available: https://searchdatacenter.techtarget.com/es/definicion/SQL-o-lenguaje-de-consultas-estructuradas. [Último acceso: 12 mayo 2019].

[28] «Microsoft,» [En línea]. Available: https://docs.microsoft.com/en-us/ef/. [Último acceso: 18 agosto 2019].

[29] J. M. Alarcón, «campus MVP,» 12 marzo 2018. [En línea]. Available: https://www.campusmvp.es/recursos/post/entity-framework-code-first-database-first-y-model-first-en-que-consiste-cada-uno.aspx.

[30] J. A. Armas Navarrete, *Desarrollo de un sistema de gestión de datos para el monitoreo integral de glaciares.,* Quito: Escuela Politecnica Nacional, 2019.

[31] P. Hundermark, «Un mejor Scrum,» Ciudad del Cabo, 2009.

[32] A. Frechina, «WiinRed.es,» 18 Junio 2018. [En línea]. Available: https://winred.es/management/metodologia-scrum-que-es/gmx-niv116-con24594.htm. [Último acceso: 3 noviembre 2019].

[33] A. R. Mesa, «OpenWebinars,» 19 Diciembre 2018. [En línea]. Available: https://openwebinars.net/blog/que-es-un-sprint-scrum/. [Último acceso: 10 noviembre 2019].

[34] K. Schwaber y J. Sutherlad, « SCRUM GUIDES,» november 2017. [En línea]. Available: https://scrumguides.org/scrum-guide.html#events-sprint. [Último acceso: 20 noviembre 2019].

[35] . K. Schwaber y J. Sutherland, «La guía de Scrum,» julio 2013. [En línea]. Available: https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf. [Último acceso: 5 noviembre 2019].

[36] J. Quijano, «GENBETA,» 31 Marzo 2012. [En línea]. Available: https://www.genbeta.com/desarrollo/cuando-todos-son-ventajas-sprint-backlog-hablando-de-scrum.

[37] «VBF,» [En línea]. Available: https://www.verbindungszentrum.com/scrum-en/. [Último acceso: 11 Noviembre 2019].

[38] M. Brind, «Mikesdotnetting,» 24 March 2015. [En línea]. Available: https://www.mikesdotnetting.com/article/268/how-to-send-email-in-asp-net-mvc.

[39] M. Rouse, «SearchWindows Server,» [En línea]. Available: https://searchwindowsserver.techtarget.com/definition/IIS. [Último acceso: 01 2020].

[40] T. Ardal, «elmah.io,» 21 mayo 2019. [En línea]. Available: https://blog.elmah.io/the-ultimate-guide-to-connection-strings-in-web-config/. [Último acceso: 20 noviembre 2019].

ANEXOS

ANEXO A. Reglamentación de la Escuela Politécnica Nacional

ANEXO B. Entrevistas

ANEXO C. Script Base de Datos

ANEXO D. Código de la aplicación web

ANEXO E. Pruebas unitarias

ANEXO F. Encuestas

ORDEN DE EMPASTADO