ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

AUTOMATIZACIÓN DEL PROCESO DE APROBACIÓN DE PLANES DE TRABAJOS DE TITULACIÓN EN LA CARRERA DE INGENIERIA ELECTRÓNICA Y REDES DE INFORMACIÓN

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

JOSÉ ALEJANDRO BRIONES ROMERO

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, MSc.

Quito, mayo 2017

DECLARACIÓN

Yo, José Alejandro Briones Romero, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

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

_____

José Alejandro Briones Romero

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por José Alejandro Briones Romero bajo mi supervisión.

_____

Ing. David Mejía, MSc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

AGRADECIMIENTOS

A Dios, por la vida, el regalo más grande.

A mi madre por su infinito apoyo y amor, por ser mi ejemplo más grande a través de su trabajo y liderazgo. Por enseñarme que ser feliz es hacer felices a los demás.

A mis abuelos por todos sus consejos llenos de tanta sabiduría y cariño, y por todo su sacrificio y tenacidad al formar esta hermosa familia a la que me ha tocado pertenecer.

A mis tíos, tías, primos y primas, desde los más pequeños hasta los más grandes, por su pasión tan grande, a los libros, a la música, al fútbol, a la vida. Por su humor tan único, por su locura, por sus sonrisas y por su solidaridad, porque con sus logros y fracasos han sido mi mejor escuela y son las fuerzas que me permiten levantarme cuando me caigo.

A mis ángeles Saúl, Lucho, Bertha y Edgar, porque su amor y compañía trascienden la existencia.

A mis amigos que forman y formaron parte de Perimetral y Capital Klank, gracias por llenar de música mi vida.

A los RFC por seguir compartiendo grandes amistades a través del fútbol.

A los panas del CSG y de la Poli que aún siguen brindándome su amistad sin importar el paso del tiempo, y a todos las personas que han estado siempre apoyándome, en especial a Cris, Pris, Diana, Will y Jeff.

A Lichi por todo su amor, comprensión -y ternura- y apoyo en este último paso, lleno de tantas caídas.

Al Ing. David Mejía por su guía y paciencia al dirigir este proyecto.

A David Núñez y Vinicio Ochoa por su ayuda en el desarrollo de la aplicación.

DEDICATORIA

A mi madre, quien me ha dedicado su vida.

A mis abuelos: Bertha, Marianita (Mamín) y Arcesio.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

En este Trabajo de Titulación se presenta el desarrollo de una aplicación web que permite automatizar el proceso de aprobación de planes de Trabajos de Titulación presentados en la Carrera de Ingeniería en Electrónica y Redes de Información.

En el primer capítulo se presenta la teoría en la que se sustentó el desarrollo de la aplicación web, la cual fue realizada usando el *framework* ASP.NET MVC y siguiendo la metodología de desarrollo ágil Kanban. En primer lugar se presentan conceptos sobre una aplicación web y el patrón de diseño Modelo-Vista-Controlador (MVC), más adelante se exponen las herramientas que permitieron implementar el software a través del *framework* ASP.NET MVC y se definen los fundamentos de la metodología Kanban. Por último se presentan conceptos de herramientas utilizadas en el desarrollo de la aplicación, tales como historias de usuario, diagramas y pruebas.

En el segundo capítulo se presenta el proceso actual de aprobación de planes de Trabajos de Titulación y los requerimientos de la aplicación. Se inicia describiendo el proceso actual para la aprobación de los planes de Trabajo de Titulación en la Carrera, posteriormente se definen los requerimientos generados por los usuarios que interactuarán con la aplicación, y a partir de estos requerimientos se divide a la aplicación en cuatro módulos y se establecen historias de usuario para cada módulo.

En el tercer capítulo se presentan diagramas generados en la etapa de diseño y más adelante se muestra la implementación de cada módulo a través de códigos que ejemplifican el trabajo realizado. Finalmente se presentan los resultados de las pruebas realizadas sobre la aplicación, las cuales permiten comprobar el cumplimiento de los requerimientos inicialmente definidos.

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

Finalmente en los anexos se muestran el *backlog* y las historias de usuario que se han definido para el desarrollo de la aplicación.

PRESENTACIÓN

Actualmente el proceso de aprobación de planes de Trabajos de Titulación en la Carrera de Ingeniería en Electrónica y Redes de Información muestra una falta de seguimiento que provoca ocasionalmente la prolongación del tiempo de aprobación de un plan, ya que en algunos casos existe una comunicación ineficiente entre estudiantes y la Comisión Permanente de Trabajos de Titulación; la cual se encarga de evaluar los planes y de informar de dicha evaluación a través de la Secretaría de la Carrera.

Para agilizar este proceso se plantea a través de este Trabajo de Titulación, el desarrollo de una aplicación web que permita automatizar la aprobación de planes de Trabajos de Titulación. Esta aplicación ha sido implementada a través de cuatro módulos: Módulo de Autenticación y Roles de Usuario, Módulo de Gestión de la Información, Módulo de Envío de Correos Electrónicos y Módulo de Gestión de Rúbricas.

El primer módulo implementa un mecanismo de autenticación y permite gestionar roles y usuarios en la aplicación. Se han definido para la aplicación los roles: Administrador, Miembro de Comisión y Secretaria, estos roles serán asignados a cada usuario por parte del administrador de la aplicación.

En el segundo módulo se maneja información correspondiente a estudiantes, docentes y planes de Trabajos de Titulación. Para la creación de un plan se deberá cargar un archivo PDF que contendrá el plan escaneado, este archivo será enviado posteriormente vía correo electrónico a los miembros de la Comisión para su eventual revisión. Además en este módulo se permitirá visualizar registros de estudiantes y los planes que estos han realizado, registros de docentes y los planes que estos han dirigido o codirigido y registros de evaluaciones de planes.

En el tercer módulo se gestiona la creación y envío de correos electrónicos. Estos correos permitirán notificar el ingreso de un plan al sistema a estudiantes, directores/codirectores y miembros de la Comisión, y además permitirán informar a estudiantes y docentes acerca del resultado de la evaluación de un determinado

plan, definiendo si un plan ha sido aprobado, negado o continúa pendiente por aprobar.

El cuarto módulo permite la creación y el uso de rúbricas para evaluar un determinado plan de Trabajo de Titulación, así como la generación de un archivo que contenga la rúbrica utilizada en la evaluación para posteriormente informar al estudiante de las observaciones dadas en dicha evaluación vía correo electrónico. Estas observaciones evidenciarán el resultado de la evaluación y permitirán al estudiante conocer dicha evaluación y de ser el caso realizar las modificaciones necesarias para la posterior presentación de su plan.

CAPÍTULO 1

MARCO TEÓRICO

El presente Trabajo de Titulación consiste en desarrollar una aplicación web para automatizar y gestionar de mejor manera la aprobación de planes de Trabajos de Titulación en la Carrera de Ingeniería en Electrónica y Redes de Información. En este capítulo se presenta un resumen de la teoría en la que se fundamenta el presente Trabajo de Titulación. Primero se brinda una descripción sobre aplicaciones web y se mencionan conceptos sobre patrones de diseño, centrándose específicamente en el patrón de diseño Modelo-Vista-Controlador (MVC). Más adelante se exponen las herramientas utilizadas para el desarrollo de software, las mismas que forman parte del *framework*[1] ASP.NET MVC. También se explican conceptos relacionados a la metodología ágil Kanban, la cual ha sido usada para desarrollar la aplicación web. Finalmente se revisan conceptos acerca de las herramientas usadas para desarrollar la aplicación web.

1.1 APLICACIÓN WEB

Una aplicación web es una herramienta de software basada en el modelo cliente-servidor a la cual se puede acceder dentro de una red mediante una ruta específica que apunta a un servidor web. El protocolo que se usa para poder comunicarse entre cliente y servidor es HTTP[2].La red en cuestión usa la arquitectura TCP/IP[3]. Una aplicación web actúa de la siguiente manera:

- **Cliente:** La aplicación web se ejecuta a través de un navegador web. El navegador web es el cliente y se ayuda de varios *scripts* que permiten tener una mayor interacción entre el usuario y el navegador.

---

[1] *Framework*: Es un entorno de trabajo cuya función es ayudar al desarrollador a programar su aplicación de software. Normalmente cuenta con componentes de software tales como librerías, plantillas, compiladores, máquinas virtuales, entre otros.

[2] HTTP (*Hypertext Transfer Protocol)*: Protocolo que permite la comunicación en la web y corre sobre la capa aplicación de la arquitectura TCP/IP.

[3] Arquitectura TCP/IP. Basada en los protocolos TCP (*Transfer Control Protocol*) e IP (*Internet Protocol*) que permite realizar la comunicación entre dispositivos.

- **Servidor:** El servidor web responderá cada una de las peticiones que haga el usuario, pudiendo llegar a interactuar con la información alojada en las bases de datos.

En la Figura 1.1 se muestra la interacción existente entre cliente y servidor, la misma que consiste en responder las solicitudes hechas por el cliente a través del servidor.

1.1.1 VENTAJAS Y DESVENTAJAS

A continuación se expondrán las ventajas y desventajas de utilizar una aplicación web [1].

Entre las ventajas se pueden mencionar las siguientes:

- Una aplicación web puede correr en diferentes navegadores web de cualquier sistema operativo.
- Existe una mayor facilidad para actualizar y dar mantenimiento a la aplicación web con respecto a una aplicación de escritorio.
- No necesita la instalación y distribución del software en cada computadora cliente.
- Todos estos aspectos derivan en una reducción de costos.

Entre las desventajas se consideran:

- La experiencia de usuario generalmente será mejor en una aplicación de escritorio.
- Pueden ser blancos de ataques que atenten contra la privacidad y seguridad.

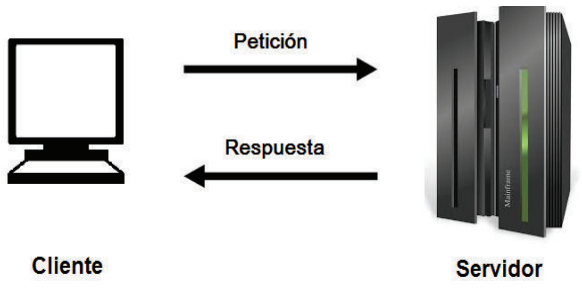

Figura 0.1. Modelo Cliente-Servidor

1.2 PATRÓN DE DISEÑO

Usar un patrón de diseño para el desarrollo y mantenimiento de una aplicación puede simplificar el trabajo del programador ya que resulta importante manejar abstracciones para poder lidiar con la complejidad que se presente.

Un patrón de diseño es una descripción de como objetos y clases pueden interactuar con la finalidad de resolver un problema de diseño en una situación particular. También puede ser visto como una plantilla abstracta que puede ser usada una y otra vez. El uso de abstracciones resulta muy útil sobre todo para ambientes que usen paradigmas de programación orientada a objetos [1].

A continuación se expondrán conceptos sobre el patrón de diseño basado en la arquitectura de 3 capas que representa la estructura de una aplicación web y el patrón de diseño MVC el cual será usado para desarrollar el presente trabajo.

1.2.1 PATRONES DE DISEÑO BASADOS EN LA ARQUITECTURA DE 3 CAPAS [2]

Los patrones de diseño basados en capas son muy usados en el desarrollo de aplicaciones web ya que permiten visualizar como están organizadas las diferentes estructuras de software dentro del modelo cliente-servidor.

Estos patrones de diseño se basan en separar al sistema en diferentes piezas que cumplan con funcionalidades específicas de manera que cada capa interactúe con sus capas adyacentes a través de interfaces definidas.

Dentro de las arquitecturas basadas en capas uno de los patrones más usados es el basado en 3 capas. Para una aplicación web, tal como se puede observar en la Figura 1.2, la arquitectura está formada por las siguientes capas:

- **Capa de Presentación:** Representa la interfaz con la que interactúa el usuario, es decir, el navegador web.
- **Capa de Aplicación (o Lógica):** Crea, recupera, modifica o elimina datos en la capa de datos y envía los resultados a la capa de presentación. Aquí se encuentra el servidor web y la lógica asociada con la generación de contenido.

- **Capa de Datos:** Representa a la fuente de datos asociada a la aplicación, es decir, la base de datos.

La Capa de Presentación suele dividirse en dos subcapas:

- **Subcapa del Cliente:** Contiene los componentes de la interfaz de usuario.
- **Subcapa de Lógica de Presentación:** Contiene los *scripts* usados para interactuar con el contenido de las páginas web.

La Capa de Aplicación o Lógica puede dividirse en dos subcapas:

- **Subcapa de Lógica de Negocios:** Asocia los objetos con la aplicación y determina la forma en que este proceso se realiza.
- **Subcapa de Acceso de Datos:** Accede a la información y la envía a la capa de lógica de negocios.

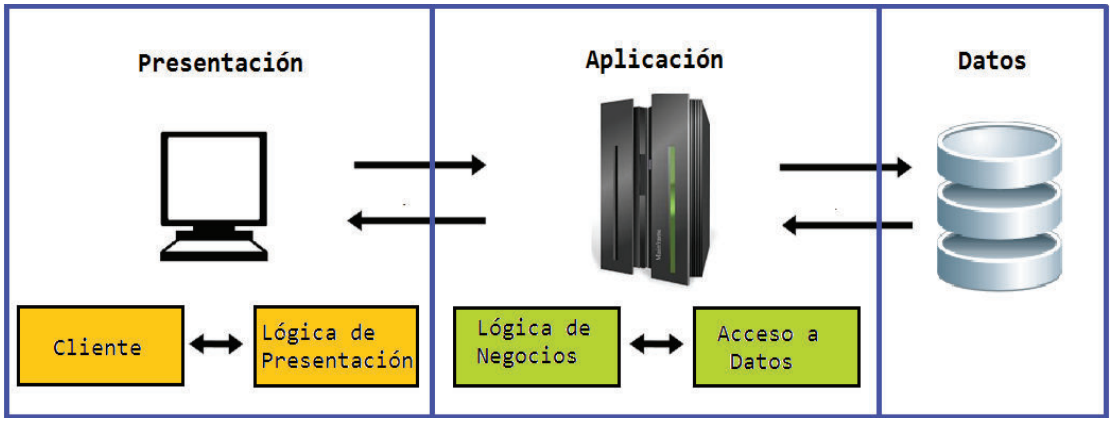

Figura 0.2. Arquitectura de 3 capas sobre la que se ejecuta una Aplicación Web

1.2.2 PATRÓN DE DISEÑO MODELO-VISTA-CONTROLADOR

Este patrón de diseño se encuentra sobre la capa intermedia de la arquitectura web y es usado por muchos *frameworks* para crear aplicaciones más robustas y fáciles de mantener logrando separar la presentación (vista) de los datos (modelo) respondiendo a las peticiones que se generen a través de un controlador. A continuación se describe cada elemento del patrón de diseño.

- **Modelo:** Es la abstracción que representa a los datos. Toda la información que se solicite en alguna consulta se envía desde el modelo.

- **Vista:** Presenta el modelo al usuario con una interfaz que represente toda la información pedida en un formato entendible e interactivo.
- **Controlador:** Es el encargado de responder a las peticiones del usuario de tal manera que presente a través de la vista la solicitud realizada utilizando el modelo correspondiente.

En la Figura 1.3 se representa la interacción entre el usuario y la aplicación usando el patrón MVC. Esta interacción se inicia a través de una petición del usuario por medio de la interfaz web, el controlador recibe la notificación de esta acción y gestiona un evento que accede al modelo correspondiente. Una vez completada la petición, la vista es actualizada con la información solicitada por el usuario.

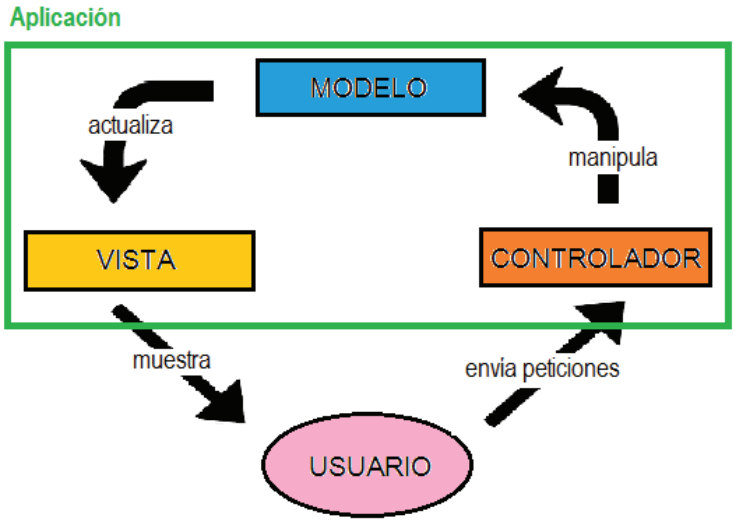

Figura 0.3. Diagrama del Funcionamiento del Patrón Modelo-Vista-Controlador entre un usuario y la aplicación web

1.3 HERRAMIENTAS DE ASP.NET MVC

Para el desarrollo del presente Trabajo de Titulación se ha elegido trabajar con el *framework* ASP.NET MVC, el cual forma parte del IDE[4] Microsoft Visual Studio. ASP.NET MVC es la tecnología con mayor soporte en los últimos años por parte

[4] IDE (*Integrated Development Environment*): Es una aplicación que facilita al programador el desarrollo del software. Un IDE está formado por un editor de código, un depurador, un compilador y/o un intérprete. Entre los IDE más usados están Visual Studio, Eclipse y NetBeans.

de Microsoft brindando altos niveles de escalabilidad y mantenimiento para aplicaciones web de todo tipo. El uso del patrón MVC permite a la aplicación tener una estructura más sólida y un desarrollo más organizado aunque también aumenta la complejidad de su entendimiento en un inicio por la falta de interacción directa con controles y formularios [3].

ASP.NET MVC cuenta con varias herramientas tecnológicas que facilitan el desarrollo de software. Por lo general estas herramientas crean plantillas de código genéricas, facilitan la interacción entre distintos lenguajes de programación o a su vez optimizan el uso de código facilitando el trabajo del programador de tal forma que este se centre específicamente en crear la lógica que manipule la información. ASP.NET MVC no es la única forma de desarrollar aplicaciones web con Visual Studio. Existen también otras tecnologías que pueden ser usadas, estas son *Web Forms*[5] y *Web Pages*[6].

A continuación se revisarán las herramientas más importantes del *framework* ASP.NET MVC, las cuales facilitan el trabajo del desarrollador al realizar la aplicación web.

1.3.1 SCAFFOLDING

Es una herramienta de ASP.NET MVC que genera código mediante el uso de plantillas estándar. *Scaffolding* básicamente permite que se genere de forma automática, a partir de un modelo, tanto controladores como vistas.

Una vez seleccionada la opción de añadir un elemento con *scaffolding* se tienen diferentes opciones como: crear controladores, crear vistas, o crear ambas estructuras usando Entity Framework[7]. Las vistas se crearán generando código HTML[8] y CSS[9] dentro de una carpeta asignada con el nombre del modelo teniendo una vista para cada opción de manipulación (visualizar, crear, actualizar,

[5] *Web Forms:* Este modelo de programación está basado en eventos y controles. Los controles son los encargados de abstraer el código HTML, CSS y JavaScript. Es la tecnología estándar para desarrollo con ASP.NET gracias al uso de controles tipo *drag-and-drop* para la creación de formularios web.

[6] *Web Pages:* Es una tecnología recomendada para aplicaciones simples o sitios web de pocas páginas por lo que es la más simple de entender y usar. Tiene cierta similitud al desarrollo de PHP y al de ASP en sus inicios. Al igual que las otras tecnologías tiene control total sobre HTML, CSS y JavaScript.

y eliminar). El controlador por su parte se creará añadiendo métodos para cada opción de manipulación de datos. Por convención el controlador conservará el nombre del modelo en plural más un distintivo que lo identifique como controlador tal como se puede apreciar en la Figura 1.4 [4].

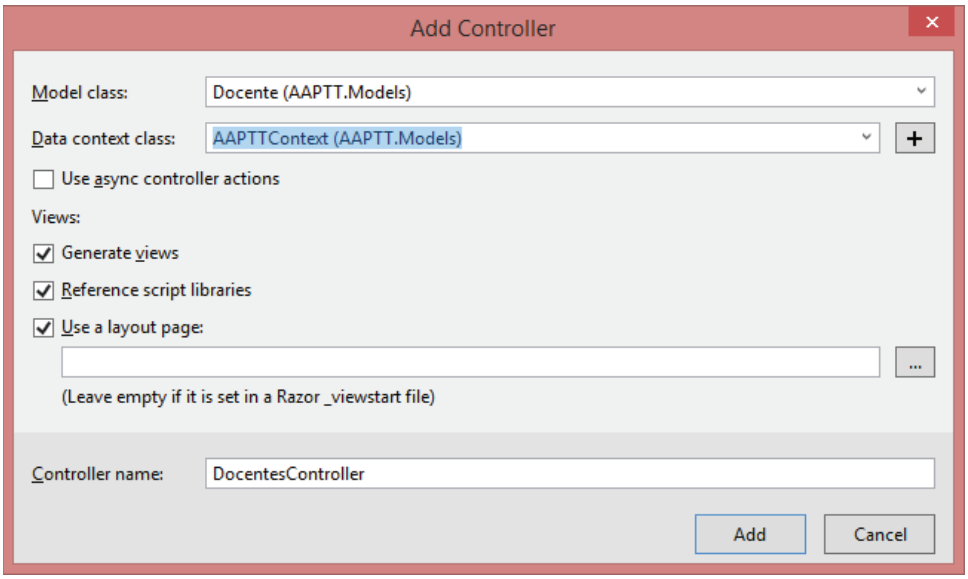

Figura 0.4. Formulario para añadir un controlador con sus respectivas vistas en el *framework* ASP.NET MVC

1.3.2 ENTITY FRAMEWORK [4]

Es un componente de software creado por desarrolladores de ASP.NET mediante código *open source* que permite realizar una asociación entre objetos y tablas de tal forma que la base de datos que funciona con un modelo relacional pueda traducirse en código que use el paradigma orientado a objetos y viceversa. Este *framework* también es conocido como motor de persistencia ya que forma una capa intermedia de persistencia que permite que tanto la capa de datos como la de aplicación puedan entenderse pese a que ambas funcionen con modelos de programación diferentes [5].

---

[7] Entity Framework: Herramienta que permite asociar objetos y clases con tablas y bases de datos.
[8] HTML (*HyperText Markup Language)*: Es el lenguaje que permite describir páginas web a través de un sistema de etiquetas que serán interpretadas por el navegador web.
[9] CSS (*Cascading Style Sheets):* Es el lenguaje que se usa para definir la presentación de un documento HTML o XML. El desarrollador puede a través de CSS usar declaraciones que definan el estilo y formato de elementos web.

La principal ventaja de Entity Framework es que elimina la necesidad de escribir código para acceder a los datos. La conexión a la base de datos se especifica en un archivo de configuración y se la hace de forma automática sin necesidad de que el desarrollador tenga que definirla.

Existen tres maneras en las cuales se puede desarrollar una aplicación usando Entity Framework: *Database First*, *Model First* y *Code First*.

- ***Database First:*** Si ya se tiene una base de datos existente Entity Framework puede crear automáticamente un modelo a partir de las tablas y columnas de dicha base de datos, las cuales serán transformadas en clases y atributos según corresponda.
- ***Model First:*** Si no se tiene una base de datos existente se puede crear un modelo usando un diseñador gráfico, el modelo se almacenará en un archivo XML con extensión .edmx[10]. Una vez creado el modelo, Entity Framework creará la base de datos a partir de sentencias DDL[11].
- ***Code First:*** Este método consiste en crear clases y atributos de manera que Entity Framework permita persistir dicha información en una base de datos con sus respectivas tablas y columnas. *Code First* es válido para proyectos que tengan o no una base de datos existente; si es que se tiene una base de datos creada, Entity Framework generará las clases y atributos correspondientes a dicha base de datos.

Para el presente trabajo se ha elegido trabajar con *Code First* ya que se pretende desarrollar una aplicación totalmente nueva sin existencia de datos y además, porque para cualquier cambio que existiese en la estructura de datos la herramienta *Code First Migrations* permite brindar persistencia en el desarrollo de la aplicación, inclusive en etapas de producción [6].

El método *Code First* de Entity Framework cuenta con varias herramientas que permiten realizar la asociación entre objetos y tablas de manera óptima entre las cuales están: la Clase `Context`, *Code First Migrations* y el Método `Seed`.

---

[10] .edmx: Es la extensión de un archivo XML que define un modelo conceptual, un modelo de almacenamiento y la relación entre ambos.

[11] DDL (*Data Definition Language*): Es un lenguaje que permite definir estructuras de datos y sus relaciones.

1.3.2.1 Clase Context

A través de esta clase, Entity Framework puede relacionar las entidades existentes con sus respectivas tablas en la base de datos de tal manera que se permita realizar tareas de: actualización, eliminación, obtención y almacenamiento de información. Si bien esta clase es comúnmente conocida como clase Context, el usuario puede renombrarla; sin embargo, se recomienda mantener la palabra Context como identificativo de la misma. La creación de esta clase se genera de forma automática al usar la herramienta Scaffolding por primera vez.

En el Código 1.1 se presenta un ejemplo de uso de la Clase `Context`. La funcionalidad de esta clase es heredada de `DbContext` (línea 5), clase que forma parte del espacio de nombres `System.Data.Entity`. En la línea 7 se define el constructor de esta clase, el cual tiene como parámetro una cadena de caracteres que representa el nombre de la base de datos.

Posteriormente se crean dos propiedades `DbSet`, las cuales representan una colección de datos asociados a un determinado modelo; el modelo `Estudiante` en la línea 11 y el modelo `Docente` en la línea 12. Una propiedad `DbSet` permite asociar un modelo con una tabla de la base de datos, permitiendo la interacción entre ambos a través de sus propiedades y métodos; un ejemplo de interacción de la propiedad `DbSet` es el manejo de operaciones CRUD[12] solicitadas por un usuario. Estas propiedades `DbSet` por convención pasan a definirse con el mismo nombre del modelo pero en plural.

```
1   using System.Data.Entity;
2
3   namespace AAPTT.Models
4   {
5       public class AAPTTContext : DbContext
6       {
7           public AAPTTContext() : base("name=AAPTTContext")
8           {
9           }
10
11          public DbSet<Estudiante> Estudiantes { get; set; }
12          public DbSet<Docente> Docentes { get; set; }
13
14      }
```

Código 0.1. Ejemplo de uso de la Clase *Context*.

[12] CRUD: Acrónimo de *Create*, *Read*, *Update* y *Delete*; representa las funciones básicas que se pueden realizar en el manejo de datos.

1.3.2.2 Code First Migrations

Esta herramienta brinda persistencia entre cambios hechos en el código de la aplicación con respecto a la base de datos y además permite automatizar el despliegue de la base de datos en fase de producción [6]. Si es que el desarrollador no realiza las migraciones correspondientes, la aplicación no podrá ejecutarse y se notificará que hay información desactualizada en la base de datos. Una vez realizada una migración se debe proceder a actualizar la base de datos, para esto se ejecutan los comandos `Add-Migration` y `Update-Database` en la *Package Manager Console*[13]. La utilización de estos comandos se muestra en la Figura 1.5 y la Figura 1.6, respectivamente.

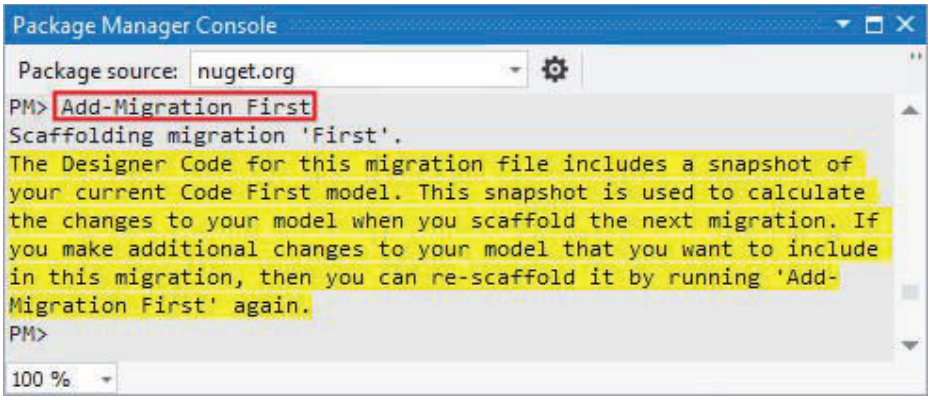

Figura 0.5. Adición de una migración [7]

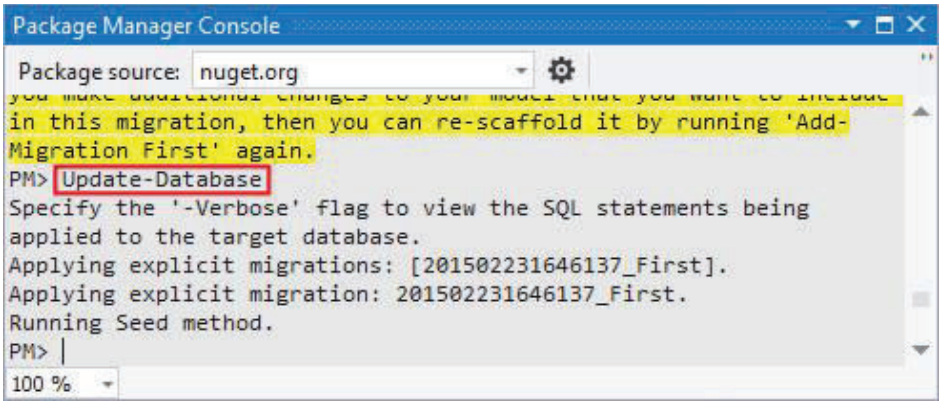

Figura 0.6. Actualización de la base de datos luego de realizar la migración [7]

[13] *Package Manager Console*: Consola de Gestión de Paquetes, es una consola usada en Visual Studio para interactuar con paquetes que pueden venir previamente instalados o pueden ser adquiridos posteriormente.

1.3.2.3 Método Seed

Una aplicación necesita tener una base de datos poblada para poder interactuar con dicha información, esta acción se la puede realizar usando un método que permita tener datos cargados por defecto. Este método es conocido como `Seed`, el cual corre cada vez que existe una actualización en la base de datos a través de las migraciones. El método `Seed` se encuentra dentro de la clase `Configuration`, la cual se crea automáticamente al realizar la primera migración.

En el Código 1.2 se muestra un ejemplo del uso del método `Seed`. Este método recibe como parámetro una instancia de la Clase `Context` (línea 11) para poder guardar los datos definidos más adelante.

```
11          protected override void Seed(AAPTTContext context)
12          {
13
14              var estudiantes = new List<Estudiante>
15              {
16                  new Estudiante
17                  {
18                      EstudianteID = "1775127571",
19                      NumeroUnico = "201017551",
20                      Nombres = "WALTER ANDRES",
21                      Apellidos = "CARGUA HINOJOSA",
22                      Telefono = "0993781634",
23                      CorreoElectronico = "walter.cargua@epn.edu.ec",
24                      CarreraEstudianteID = 3,
25                      EstadoEstudiante = true
26                  },
27                  new Estudiante
28                  {
29                      EstudianteID = "1775128682",
30                      NumeroUnico = "201017552",
31                      Nombres = "MARCOS JULIO",
32                      Apellidos = "MORENO JARAMILLO",
33                      Telefono = "0984230382",
34                      CorreoElectronico = "marcos.moreno@epn.edu.ec",
35                      CarreraEstudianteID = 1,
36                      EstadoEstudiante = true
37                  }
38                      };
39
40              estudiantes.ForEach(s => context.Estudiantes.AddOrUpdate(p => p.EstudianteID, s));
41              context.SaveChanges();
42          }
43
```

Código 0.2. Uso del Método `Seed` para el ingreso de dos registros en la tabla `Estudiantes`

En este caso se va a proceder a crear dos registros de la tabla Estudiantes, para esto se crea la variable `estudiantes`, la cual almacena una lista de objetos de

tipo `Estudiante` (línea14). En la lista se almacena la información necesaria para crear los dos registros a través de dos objetos `Estudiante`, objetos que deberán cargar información dentro de las propiedades correspondientes a cada columna de la tabla, siendo estas: `EstudianteID`, `NumeroUnico`, `Nombres`, `Apellidos`, `Telefono`, `CorreoElectronico`, `CarreraEstudianteID` y `EstadoEstudiante` (líneas 18 a la 24 y líneas 29 a la 36).

Finalmente se procede a guardar los registros previamente definidos utilizando una iteración *foreach*, la misma que a través del método `AddOrUpdate` permite guardar dicha información en la clase *Context*, sin embargo, para que exista persistencia y los datos se almacenen en la base de datos es necesario el uso del método `SaveChanges`.

1.3.3 RAZOR

Razor es una sintaxis de programación usada en ASP.NET para embeber código C# o Visual Basic en el código HTML de una página web, de tal forma que se creen páginas dinámicas.

Algunas características de Razor son [8]:

- Se utiliza el caracter arroba (`@`) para indicar código del lado del servidor.
- Si es que el código a embeber consta de dos o más líneas se utilizan llaves para definir el bloque de código.
- Dentro de un bloque de código se debe finalizar cada línea de código con un punto y coma.
- Se usan comillas para definir *strings*.
- Las variables son sensibles al uso de mayúsculas.

Razor resulta muy útil cuando permite una interacción dinámica con elementos HTML a través de herramientas conocidas como *helpers*.

1.3.3.1 HTML Helpers

Los HTML *Helpers* son métodos que permiten pasar valores de elementos HTML a objetos que posteriormente serán usados por los controladores para definir alguna acción y afectar a un determinado modelo, y de forma inversa también se

pueden asignar valores de un modelo a elementos HTML una vez que se cargue una determinada página web.

Dentro de los HTML *Helpers* más usados se tiene:

- `Html.Label`: Genera una etiqueta.
- `Html.Display`: Genera texto HTML.
- `Html.TextBox`: Genera un cuadro de texto.
- `Html.CheckBox`: Genera una casilla de selección.
- `Html.RadioButton`: Genera un botón de selección exclusiva.
- `Html.DropDownList`: Genera una lista de opciones a seleccionarse.
- `Html.Editor`: Genera un control HTML muy parecido a un cuadro de texto pero que de acuerdo al tipo de datos podrá variar su presentación.

Estos elementos permiten tener una relación directa con los elementos HTML que existan, de forma que se puedan agregar, editar, mostrar o validar valores según la interacción del usuario y la información predefinida en cada vista.

El Código 1.3 presenta un ejemplo de una vista usando la sintaxis Razor y los HTML *Helpers*. Esta vista presenta un formulario de *login* en donde el usuario deberá ingresar su correo electrónico y su contraseña.

```
19              <div class="form-group">
20                  @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
21                  <div class="col-md-10">
22                      @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
23                      @Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })
24                  </div>
25              </div>
26
27              <div class="form-group">
28                  @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
29                  <div class="col-md-10">
30                      @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
31                      @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
32                  </div>
33              </div>
```

Código 0.3. Código Razor embebido en HTML

En primera instancia se especifica el dato a llenar mediante un `Html.Label` (línea 20), luego mediante un `Html.TextBox` se especifica el cuadro de texto donde el usuario ingresará su correo electrónico (línea 22), y por último se

especifica por medio de un `Html.ValidationMessageFor` el mensaje de alerta que saldrá en caso de no validarse los datos ingresados en el cuadro de texto. Para el siguiente campo se sigue la misma lógica (líneas 28, 30 y 31); en donde el usuario deberá ingresar su contraseña.

1.3.4 BOOTSTRAP [4]

Bootstrap es el *framework* más popular para desarrollo de *front-ends*[14] del tipo *responsive*[15]. Esta herramienta hace uso de plantillas de diseño que pueden contener elementos como botones, cuadros, menús de navegación, formularios entre otros; que se construyen a partir de HTML y CSS. Además Bootstrap utiliza jQuery y JavaScript para añadir acciones adicionales para elementos web. El implementar Bootstrap en una aplicación no es obligatorio, sin embargo el uso de CSS puro puede ser muy tedioso y complejo representando una pérdida de tiempo para el programador.

1.3.5 JAVASCRIPT [9]

Es un lenguaje de programación utilizado casi siempre del lado del cliente a través de un navegador web con el propósito de crear páginas dinámicas y mejorar la experiencia del usuario. La intención de JavaScript es poder realizar acciones dentro de una página web sin necesidad de una interacción con el servidor. Entre sus características principales se puede mencionar que es un lenguaje interpretado, orientado a objetos y débilmente tipado.

1.3.6 JQUERY [9]

Es la librería más popular de JavaScript debido a su manejo simple y su funcionalidad. jQuery nace como una herramienta orientada al manejo de las funciones más frecuentes al momento de interactuar con elementos HTML de tal

[14] Front-end: En programación web se denomina front-end a las tecnologías de software que interactúan con los usuarios a través de un navegador web. Por lo general estas tecnologías son HTML, CSS y JavaScript.

[15] Responsive: El diseño web responsive busca brindar una visualización adecuada y atractiva sin importar el dispositivo que se utilice, sea este un celular, una tablet o una computadora.

forma que exista una optimización del código usado y permita al desarrollador definir una función JavaScript de una manera más sencilla.

JavaScript y jQuery pueden funcionar sin necesidad de ser parte de algún *framework*, basta con añadir los *scripts* realizados en la página HTML. Además pueden funcionar como parte de otras librerías o herramientas como AJAX.

1.3.7 AJAX

AJAX (*Asynchronus JavaScript And* XML) es un conjunto de técnicas que sirven para desarrollar aplicaciones web interactivas. AJAX permite actualizar una porción de la página web sin tener que recargarla y de igual manera solicitar y recibir información del servidor aún después de que la página ha sido cargada.

El uso de AJAX para el desarrollo de aplicaciones web cuenta con las siguientes ventajas y desventajas [10]:

- Al recargar solo ciertos segmentos de la página web se reduce el uso del ancho de banda ya que pueden darse casos en las que no se deba recargar imágenes o videos si se está llenando campos con texto.
- La experiencia del usuario mejora al aumentar la velocidad de respuesta de la aplicación web y al brindar dinamismo a la página con una interacción directa entre el usuario y los elementos existentes en la aplicación web.
- El uso de jQuery garantiza que el código escrito para ejecutar determinada acción funcione en cualquier navegador web evitando al desarrollador tener que escribir código dependiente del navegador que se utilice para correr la aplicación web.
- Pese a los avances por estandarizar el uso de AJAX en cualquier navegador todavía existen casos de usuarios con navegadores muy antiguos o que no permiten el uso de JavaScript en su navegador de tal manera que en estos casos AJAX no funcionará.
- La implementación de AJAX en una aplicación web aumenta el grado de complejidad para el desarrollador ya que debe poder integrar el uso de diferentes tecnologías y lenguajes adicionales como HTML, CSS, Razor, entre otros.

1.3.8 JSON

JSON (*JavaScript Object Notation*) es un formato de escritura destinado para el almacenamiento e intercambio de datos. A pesar de que JSON usa una sintaxis JavaScript para definir sus estructuras, el formato JSON está basado netamente en texto, el cual puede ser leído y usado por cualquier lenguaje de programación.

En el desarrollo de aplicaciones web, JSON es muy usado ya que puede ayudar a crear objetos (o un conjunto de objetos a través de arreglos) que contengan información que debe ser enviada al servidor. Estos objetos son creados a través de un conjunto de pares clave/valor que se escriben dentro de llaves, la clave y el valor son separados por dos puntos y cada par es separado por una coma. En el Código 1.4 se puede apreciar está notación a través de un ejemplo, el cual está conformado por 3 pares clave/valor, los cuales especifican el nombre, edad y carro de una persona.

```
{ "name":"John", "age":30, "car":null }
```

Código 0.4. Ejemplo de objeto JSON [11]

1.4 METODOLOGÍA DE DESARROLLO

Para el desarrollo de una aplicación web es importante seguir una metodología que defina los procesos necesarios en la creación de la solución de software. El uso de una metodología es de vital importancia ya que muestra los pasos a seguir para obtener un producto de buena calidad y usando óptimamente los recursos disponibles.

Actualmente el desarrollo de aplicaciones web se realiza usando metodologías ágiles como XP, Scrum o Kanban [12]. Las metodologías tradicionales han pasado a un segundo plano por la exigencia que existe para entregar proyectos de desarrollo en tiempos cada vez más cortos. El uso de metodologías ágiles permite disponer de productos entregables que ayudan a definir el resultado ideal para el cliente, brindando la capacidad de poder realizar cambios durante el proceso de desarrollo sin necesidad de afectar a todo el sistema debido al trabajo modular que existe.

A continuación se expondrán los objetivos de las metodologías ágiles para luego revisar la metodología Kanban, la cual ha sido elegida para desarrollar la aplicación web de este Trabajo de Titulación.

1.4.1 METODOLOGÍAS ÁGILES

Las metodologías ágiles pueden resumir su comportamiento en el Manifiesto por el Desarrollo Ágil de Software, el cual especifica:

> "Individuos e interacciones sobre procesos y herramientas.
> Software funcionando sobre documentación extensiva.
> Colaboración con el cliente sobre negociación contractual.
> Respuesta ante el cambio sobre seguir un plan." [13]

Esto significa que aunque cada elemento es importante se valoran más los establecidos en el lado izquierdo de cada sentencia presentada.

Además dentro de este Manifiesto se especifican los siguientes Principios del Manifiesto Ágil:

- "Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia." [14]

1.4.2 METODOLOGÍA KANBAN [15]

Esta metodología nace en 1940 en Japón y fue creada por la empresa automotriz Toyota. Kanban es una metodología ágil que usa principios JIT (*Just In Time*)[16] y está enfocada en organizar el trabajo sin depender de iteraciones de tiempo fijo; la condición para trabajar en la siguiente tarea es completar el trabajo que se estaba realizando previamente.

Entre las principales características de la metodología Kanban se pueden citar:

- Visualizar el flujo manejando un tablero (*Kanban Board*). Un ejemplo del Tablero Kanban se muestra en la Figura 1.7.

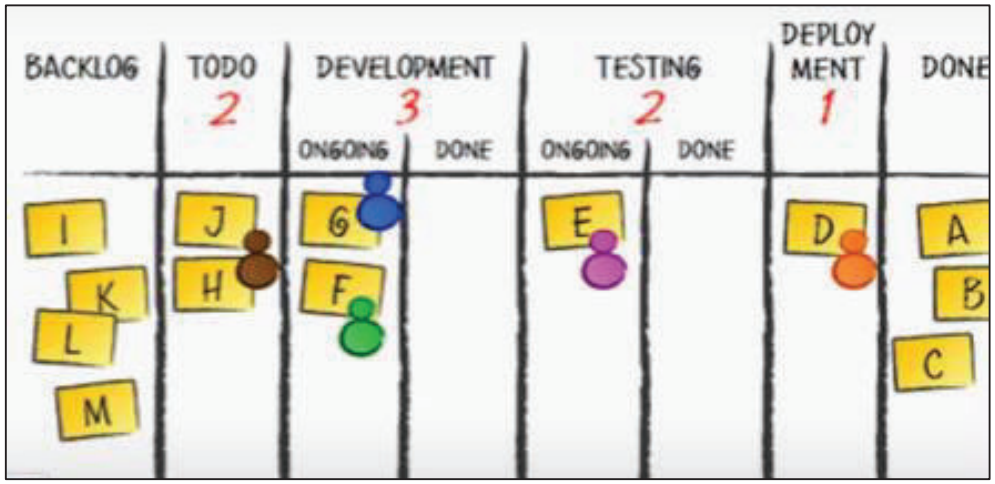

Figura 0.7. Tablero Kanban [15]

- Evitar trabajo prematuro: Las tareas se las hace el momento que necesitan ser hechas, no antes. De esta manera se evita la sobrecarga de trabajo y el desperdicio de recursos.
- Reducir el *multitasking*: Cada persona debe estar encargada de una tarea específica, ya sea en la etapa de desarrollo, pruebas o despliegue. El momento que una persona quiere hacer dos tareas a la vez es cuando la producción baja.

[16] *Just In Time*: Método que se basa en ejecutar acciones que se requieran para ese momento exacto de forma que no se desperdicien recursos y se garantice la calidad del trabajo hecho. El trabajo se basa en pedidos reales y no en suposiciones de lo que puede suceder.

- Impulsar y mejorar el trabajo en equipo: Una vez que una persona acaba su tarea debe ayudar en otras tareas que no han sido finalizadas y producen cuellos de botellas en una determinada etapa.
- Establecer políticas claras: Conseguir que cada etapa tenga sus reglas marcadas para asegurar la calidad del trabajo.
- *Daily Kanban Meeting*: Pueden hacerse reuniones de evaluación del trabajo diarias para hacer un balance de cómo se está avanzando en el proyecto haciendo un análisis de los tiempos ocupados en cada etapa.

La metodología Kanban busca tener un flujo continuo de trabajo para poder avanzar eficientemente. La idea es poder ver en el tablero donde están los cuellos de botella para tomar acciones y balancear la demanda del trabajo.

1.5 HERRAMIENTAS PARA EL DESARROLLO DE SOFTWARE

El proceso de desarrollo de software cumple con etapas definidas como el análisis de los requerimientos, el diseño del software, la implementación del código y las pruebas. Estas etapas según la metodología y los *frameworks* a usarse pueden verse alteradas y en algunos casos hasta omitidas. Sin embargo, existen representaciones que ayudan en estos procesos. A continuación se expondrán conceptos sobre historias de usuario, diagramas y pruebas que se usarán en el presente desarrollo de software.

1.5.1 HISTORIAS DE USUARIO

La metodología Kanban al ser una metodología ágil divide el trabajo de desarrollo en pequeñas tareas. Estas tareas son el producto de realizar un análisis de lo requerido por el usuario y se las puede definir a través de historias de usuario o casos de uso. Para la metodología Kanban se especifica trabajar con historias de usuario debido a que su creación requiere menos tiempo y se logra agilizar el proceso de desarrollo. Además las historias de usuario permiten una comunicación directa entre los desarrolladores de software y el cliente, y muestran una descripción concisa de lo que se quiere implementar en la aplicación..

Las historias de usuario son una especificación de los requisitos presentes, comúnmente escritas en una o dos frases con un lenguaje entendible por el usuario y el desarrollador. Una historia de usuario siempre debe especificar qué se quiere lograr, cuál es el beneficio que se obtiene y quién resulta beneficiado con esa acción.

Por tal razón el momento de enunciar una historia de usuario se puede definir los siguientes elementos [16]:

- **Identificador (ID):** Código que identifica a la historia de usuario y la diferencia de las demás. El formato del identificador es elegido por el grupo de desarrollo.
- **Rol:** El rol que tiene el usuario en dicha acción. Este elemento también se lo puede describir como el actor que participa en la historia de usuario respectiva.
- **Funcionalidad (Característica):** Representa lo que se quiere lograr en una parte específica del sistema. Es la función que debe ser ejecutada por el usuario con el rol correspondiente.
- **Razón (Resultado):** Es el fin o resultado que el usuario busca conseguir una vez ejecutada la acción. Este elemento puede ser opcional, se puede definir la historia de usuario solo con el rol y la funcionalidad.

En la Figura 1.8 se muestra un ejemplo de una historia de usuario para el requerimiento de autenticación.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU01_03_AUTENTICACION	Como usuario del sistema	necesito digitar mi correo electrónico y mi contraseña	para poder autenticarme e ingresar al sistema.

Figura 0.8. Ejemplo de una Historia de Usuario

1.5.2 DIAGRAMAS

Los diagramas forman parte importante dentro del desarrollo de software ya que permiten representar acciones e interacciones entre distintas abstracciones en el

sistema. El uso del método *Code First* de Entity Framework permite obviar ciertos diagramas para la implementación de código, aun así se pueden utilizar diagramas entidad-relación y diagramas de secuencia para describir la funcionalidad de la aplicación y generar documentación acerca de la misma. De esta manera se tiene una representación de la estructura de la base de datos y de las interacciones entre objetos en el sistema.

1.5.2.1 Diagrama Entidad-Relación

Es una herramienta que permite representar la información en entidades y sus respectivas relaciones y propiedades. Este diagrama permite modelar los datos para una futura implementación de la base de datos y representa las entidades con rectángulos, los atributos o propiedades con círculos y las relaciones con rombos.

En la Figura 1.10 se presenta un ejemplo de un diagrama entidad-relación, el mismo que cuenta con dos entidades, cuatro atributos y una relación.

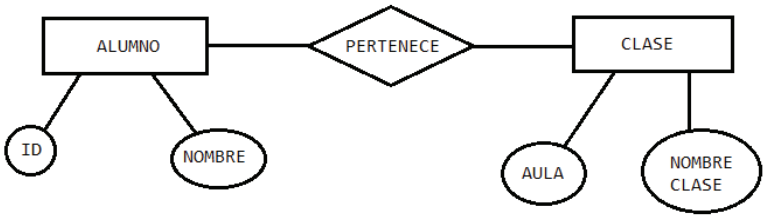

Figura 0.9. Ejemplo de Diagrama Entidad-Relación

1.5.2.2 Diagrama de Secuencia

Es una representación de las interacciones entre objetos de un sistema a lo largo de un determinado lapso de tiempo. En la Figura 1.11 se muestra un ejemplo de un diagrama de secuencia en donde las líneas de vida representan a eventos que suceden durante una interacción entre diferentes componentes. Una línea de vida puede representar a un componente del sistema, una instancia de una clase o un actor. A lo largo de las líneas de vida por medio de rectángulos verticales se representan los distintos eventos que pueden suceder a lo largo del tiempo y los mensajes que se intercambian entre componentes a través de flechas.

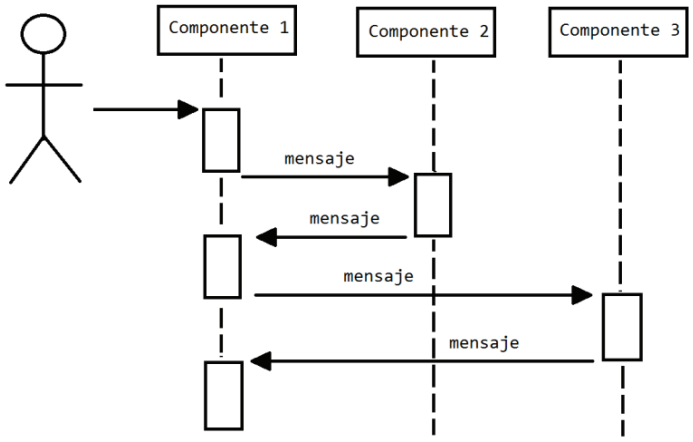

Figura 0.10. Ejemplo de Diagrama de Secuencia

1.5.3 PRUEBAS [17]

A continuación se exponen conceptos de las pruebas que se han realizado en la presente aplicación.

1.5.3.1 Pruebas Funcionales

Su objetivo es evaluar que el sistema cumpla con la función encomendada. Como ejemplos de pruebas funcionales se tiene:

- Pruebas de integración: Estas pruebas permiten evaluar que varios componentes funcionen correctamente en una aplicación.
- Pruebas de sistema: Sirven para detectar fallas al momento de interactuar con el sistema completo e integrado.
- Pruebas de aceptación: Se realizan conjuntamente con el cliente para comprobar que el software cumple con los requerimientos definidos en un inicio.
- Pruebas de compatibilidad: Estas pruebas permiten evaluar la funcionalidad de la aplicación en diferentes entornos como en navegadores web, sistemas operativos o dispositivos.

CAPÍTULO 2

ANÁLISIS DE REQUERIMIENTOS Y LEVANTAMIENTO DEL PROCESO

En este capítulo se presenta el levantamiento del proceso de aprobación de planes de Trabajos de Titulación en la Carrera y los requerimientos que permitirán desarrollar la aplicación. Con base en estos requerimientos se ha dividido a la aplicación en módulos y se han generado historias de usuario para cada módulo. Previo a esto se presenta el alcance de la aplicación y la metodología seguida para desarrollar la misma.

2.1 ALCANCE

La aplicación web descrita en el presente Trabajo de Titulación cuenta con cuatros módulos, estos son: Autenticación y Roles, Gestión de la Información, Envío de Correos Electrónicos y Gestión de Rúbricas.

El módulo de Autenticación y Roles gestiona el ingreso de los tres tipos de usuarios que existen en la aplicación: Administrador, Miembro de Comisión y Secretaria. Estos tipos de usuarios, conocidos como roles, son asignados por el Administrador del sistema a cada usuario existente.

El módulo de Gestión de la Información trabaja con información correspondiente a Estudiantes, Docentes y Planes de Trabajos de Titulación. Este módulo se encarga de la creación, edición, búsqueda y desactivación de estos registros. La eliminación de información está desactivada ya que para búsqueda de históricos se debe seguir contando con estudiantes ya graduados o docentes que han dejado de trabajar en la universidad.

Existe información propia de estos tres tipos de registros que debe ser configurable ya que puede variar con el paso del tiempo, es el caso de la Carrera a la que pertenece un estudiante, los Tipos de Trabajos de Titulación y las Líneas de Investigación de un Plan de Trabajo de Titulación, todos estos campos pueden

cambiar su denominación, incrementarse o disminuirse con el paso del tiempo. Se han manejado cuatro estados posibles para los Planes de Trabajos de Titulación, estos son: Aprobado, Negado, Pendiente por Aprobar y Anulado. Además en este módulo se maneja la búsqueda de los planes que han realizado determinados estudiantes, la búsqueda de planes que han dirigido o codirigido determinados docentes, y además la búsqueda de planes evaluados permitiendo observar los estados por los que los planes han pasado luego de la respectiva evaluación realizada por parte de la Comisión Permanente de Trabajos de Titulación.

El módulo de Envío de Correos Electrónicos gestiona la creación y envío de correos electrónicos a estudiantes en situaciones como: ingreso de un plan al sistema, aprobación de un plan, negación de un plan, o también al realizarse observaciones a un plan luego de la revisión de la Comisión. También para el caso en que se ingrese un plan en el sistema se notificará a los miembros de la Comisión mediante un correo electrónico que tenga como archivo adjunto el plan de Trabajo de Titulación en formato PDF. El cuerpo de los correos electrónicos podrá configurarse para cada caso mencionado anteriormente.

El módulo de Gestión de Rúbricas manejará dos instancias, la creación de una rúbrica y el uso de esta para evaluar un plan. En la primera instancia el administrador de la aplicación crea una rúbrica y asigna porcentajes para definir si el plan está aprobado, negado o pendiente por aprobar, posteriormente se definen los criterios a ser evaluados en la rúbrica y la valoración máxima asignada a cada uno de estos criterios.

Para el uso de rúbricas, se elige la rúbrica con la que se evaluará un plan y el plan a ser evaluado, luego se da una valoración y se presentan las observaciones necesarias para cada criterio. Al finalizar la evaluación de cada criterio se procede a guardar esta información y se define el estado del plan de Trabajo de Titulación según porcentajes asignados a la rúbrica en la instancia de creación.

2.2 METODOLOGÍA

La aplicación web en la que se basa el presente Trabajo de Titulación ha sido desarrollada bajo la metodología Kanban, la cual forma parte de las metodologías

ágiles dentro del desarrollo de software. A continuación se describe el proceso de desarrollo de la aplicación web.

En primera instancia se realizaron entrevistas con el personal administrativo para conocer el proceso actual de la aprobación de planes de Trabajos de Titulación. Una vez que se entendió el proceso y los actores que interactúan en el mismo, se procedió a definir los requisitos del sistema a implementarse, para esto se realizaron entrevistas con los miembros de la Comisión Permanente de Trabajos de Titulación, esta comisión es la encargada de evaluar los planes para su posterior aprobación o negación.

Luego de recoger los requisitos presentados por los miembros de la Comisión, se construyeron historias de usuario, las cuales pasaron a formar parte de la lista de tareas conocida como *backlog*, y se procedió a realizar cada tarea de forma secuencial.

Para el diseño de la aplicación web se definieron cuatro módulos: Módulo de Autenticación y Roles, Módulo de Gestión de la Información, Módulo de Envío de Correos Electrónicos y Módulo de Gestión de Rúbricas. Además se establecieron diagramas entidad – relación y diagramas relacionales para la base de datos, diagramas de clases para el código y un diagrama de secuencia para entender la interacción MVC en el sistema. La implementación del código fue realizada utilizando el método *Code First* de Entity Framework, en el cual se definen primero los modelos y posteriormente los controladores y vistas usando en algunos casos la herramienta *Scaffolding*, esta herramienta permite crear de forma automática el controlador y las vistas asociadas a un determinado modelo, y además la clase `Context`, clase que permite a los modelos interactuar con la base de datos.

Para la implementación de la base de datos se utilizó una versión ligera de SQL Server Express conocida como LocalDB, la cual viene preinstalada en el IDE Visual Studio.

Al finalizar la implementación de una historia de usuario se realizaron pruebas de funcionalidad sobre dicho requerimiento hasta tener el resultado esperado. Luego de terminar las historias de usuario asociadas a un módulo se realizaron pruebas

sobre todo el módulo, y finalmente se realizaron pruebas de integración para corroborar el correcto funcionamiento del sistema y pruebas de compatibilidad para verificar el funcionamiento de la aplicación en varios navegadores web. El Director del presente Trabajo de Titulación también ha evaluado las funcionalidades del software, cumpliendo las veces de un cliente y realizando pruebas de aceptación. A lo largo de estas evaluaciones se definieron nuevos requerimientos, los cuales fueron agregados como nuevas historias de usuario; estas nuevas historias de usuario fueron desarrolladas siguiendo la metodología mencionada anteriormente.

2.3 LEVANTAMIENTO DEL PROCESO

Para entender el proceso actual de aprobación de un plan de Trabajo de Titulación se realizaron entrevistas con las secretarias de la Carrera de Ingeniería en Electrónica y Control y de la Carrera de Ingeniería en Electrónica y Redes de Información, las cuales sirven de vínculo entre los estudiantes y la Comisión Permanente de Trabajos de Titulación, ente encargado de revisar los planes. Esta Comisión está conformada por el Coordinador de la Carrera y dos docentes más.

El proceso empieza cuando el estudiante presenta su plan de Trabajo de Titulación en secretaría para que este pueda ser ingresado al SAEw[17] y forme parte de la lista de planes por aprobar. Si el estudiante tiene un plan presentado previamente, primero deberá anular dicho plan para poder presentar el plan actual y que este pueda ser ingresado al SAEw. Posteriormente la Comisión se reúne, analiza los planes y presenta las observaciones a cada uno de ellos. Luego de la reunión, el Coordinador de la Carrera ingresa las observaciones a cada plan en el SAEw. El estudiante es notificado vía SAEw del estado de su plan, el cual puede ser: Aprobado, Pendiente por Aprobar o Negado.

En caso de que el plan haya sido negado, se debe presentar otro plan. En caso de que el plan siga pendiente por aprobar se deben hacer modificaciones según las observaciones hechas en la sesión por parte de la Comisión. El nuevo plan

[17] SAEw (Sistema de Administración Estudiantil Web). Sistema Web que permite gestionar información entre estudiantes, docentes, administrativos y demás miembros de la Escuela Politécnica Nacional.

modificado será entregado en secretaría nuevamente. La Comisión volverá a tratar el plan hasta que no exista ninguna observación y el plan se pueda aprobar. Una vez aprobado el plan, el Coordinador genera el acta correspondiente a través del SAEw y el Subdecanato envía una notificación vía Quipux[18] al Director del Trabajo de Titulación mencionando que el plan ha sido aprobado.

En la Figura 2.1 se expone un diagrama de flujo del proceso actual al que se somete un plan de Trabajo de Titulación para poder ser aprobado por parte de la Comisión Permanente de Trabajos de Titulación. En la primera parte de la Figura 2.1 se presenta un diagrama de flujo para el caso en el que un estudiante haya presentado un plan previamente. Para este caso el estudiante deberá anular el plan antes presentado para poder presentar un plan de Trabajo de Titulación en la Secretaría de la Carrera. En la segunda parte de la Figura 2.1 se expone el proceso de actual de aprobación de un plan de Trabajo de Titulación considerando los estados a los que puede llegar un plan luego de la evaluación por parte de la Comisión.

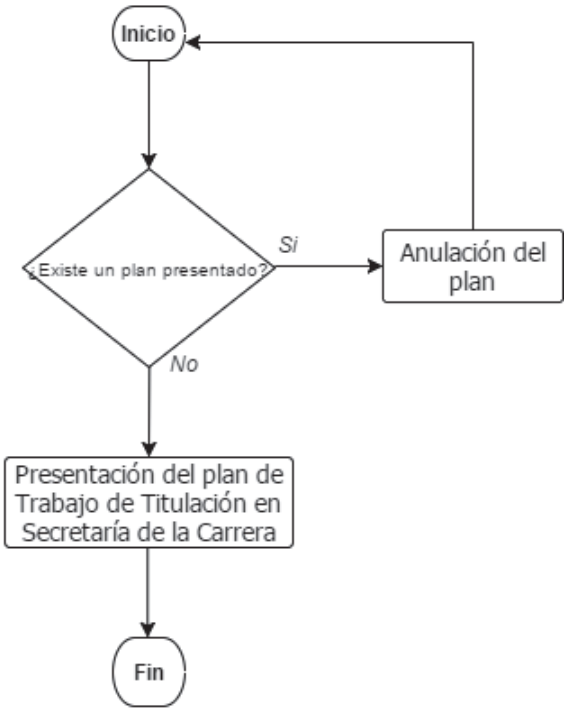

Figura 2.1. Proceso Actual de Aprobación de un Plan de Trabajo de Titulación

[18] Quipux: Es el sistema de gestión documental utilizado dentro de instituciones públicas para el manejo de oficios, circulares, memorandos, etc.

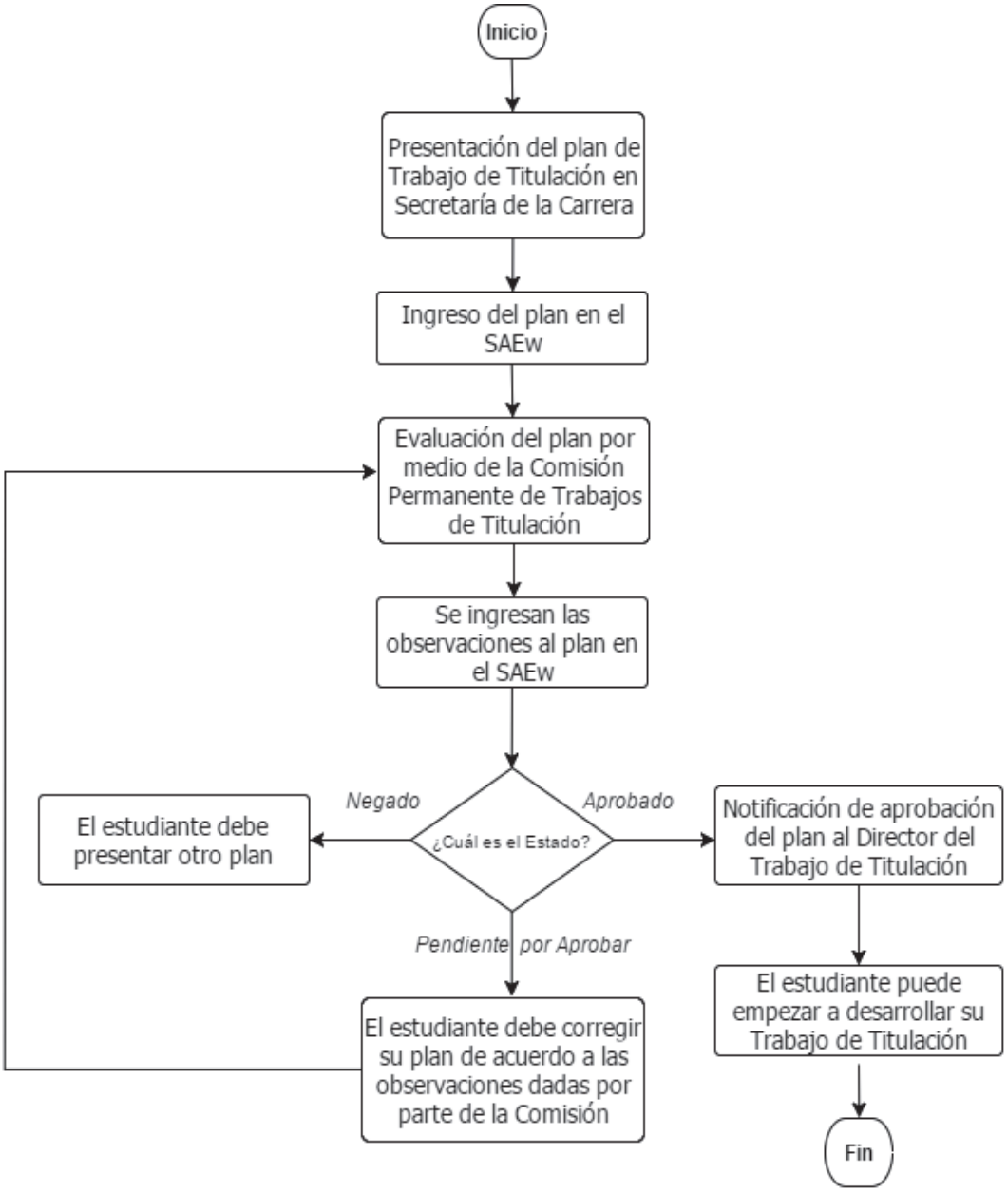

Figura 2.2. Proceso Actual de Aprobación de un Plan de Trabajo de Titulación (Continuación)

2.4 REQUERIMIENTOS DEL SISTEMA

Los requerimientos del sistema fueron planteados a partir de entrevistas realizadas en el mes de febrero del 2016 a los miembros de la Comisión Permanente de Trabajos de Titulación y a la Secretaria de la Coordinación de la

Carrera de Ingeniería Electrónica y Redes de Información. Entre los requisitos recogidos se tiene:

- Implementar un mecanismo de autenticación para acceder al sistema.
- Definir roles de usuarios para discernir lo que cada usuario puede realizar en la aplicación. Los roles son:
  - **Administrador:** Este rol corresponderá al Coordinador de la Carrera y Presidente de la Comisión Permanente de Trabajos de Titulación. Este usuario podrá manipular toda la información del sistema.
  - **Secretaria:** Este rol corresponderá a la secretaria de la Carrera y su rol será principalmente la gestión de información de la aplicación, como por ejemplo, el ingreso de nuevos planes de Trabajos de Titulación en el sistema, la actualización de información de docentes y estudiantes, entre otros.
  - **Miembro de Comisión:** Este rol corresponderá a los otros dos miembros de la Comisión Permanente de Trabajos de Titulación, los cuales podrán visualizar información del sistema.
- Definir un mecanismo de gestión y manipulación de información de estudiantes, docentes y planes de Trabajos de Titulación.
- Permitir buscar información correspondiente a planes dirigidos o codirigidos por docentes, planes realizados por estudiantes, planes evaluados, entre otros.
- Permitir notificar al estudiante vía correo electrónico cada resolución respecto al plan luego de que este haya sido evaluado por la Comisión y también al ingresar un plan al sistema.
- Permitir notificar a los miembros de la comisión a través de un correo electrónico el ingreso de un nuevo plan o la evaluación de un plan por medio de la Comisión Permanente de Trabajos de Titulación.
- Permitir evaluar un plan de Trabajo de Titulación definiendo la valoración de un conjunto de criterios establecidos en una rúbrica.

2.5 MÓDULOS DEL SISTEMA

En base a los requisitos recogidos se han definido cuatro módulos en el sistema:

2.5.1 MÓDULO DE AUTENTICACIÓN Y ROLES

Este módulo define el mecanismo de autenticación de la aplicación así como la gestión de los roles y usuarios del sistema.

2.5.2 MÓDULO DE GESTIÓN DE LA INFORMACIÓN

En este módulo se definen las funciones CRUD para gestionar la información correspondiente a estudiantes, docentes y planes de Trabajos de Titulación. Además se permite realizar la búsqueda de información según los parámetros adecuados.

2.5.3 MÓDULO DE ENVÍO DE CORREOS ELECTRÓNICOS

En este módulo se define el envío de correos electrónicos para cada instancia que se requiera según el proceso de aprobación de cada plan de Trabajo de Titulación, así como la manipulación de la información a enviar en cada correo electrónico.

2.5.4 MÓDULO DE GESTIÓN DE RÚBRICAS

Este módulo permite la creación y uso de rúbricas para la evaluación de un determinado plan de Trabajo de Titulación.

2.6 HISTORIAS DE USUARIO

Una vez organizados los requerimientos se ha procedido a crear las historias de usuario que permitirán empezar a desarrollar la aplicación web. Se han especificado 32 historias de usuario para cumplir cada funcionalidad de la aplicación.

Una vez realizadas las historias de usuario, cada una de estas se transforma en una tarea por hacer dentro del *backlog*, el cual se muestra en el Anexo A. En la Figura 2.2 se presentan todas las historias de usuario de acuerdo a su identificador. Este identificador tiene como estructura las iniciales “HU”

correspondientes a Historias de Usuario, un primer número que representa al módulo correspondiente y un segundo número que identifica a cada historia de usuario. Para el segundo módulo se añade un número intermedio, el cual se define según el tipo de información: ya sea esta correspondiente a estudiantes, docentes o planes de Trabajos de Titulación.

MÓDULO DE AUTENTICACIÓN Y ROLES

1	HU01_01_USUARIOS
2	HU01_02_ROLES
3	HU01_03_AUTENTICACION
4	HU01_04_OLVIDO_CONTRASEÑA

MÓDULO DE GESTIÓN DE LA INFORMACIÓN

5	HU02_01_01_CREAR_ESTUDIANTE
6	HU02_01_02_EDITAR_ESTUDIANTE
7	HU02_01_03_BUSCAR_ESTUDIANTE
8	HU02_02_01_CREAR_DOCENTE
9	HU02_02_02_EDITAR_DOCENTE
10	HU02_02_03_BUSCAR_DOCENTE
11	HU02_03_01_CREAR_PTT
12	HU02_03_02_PDF_PTT
13	HU02_03_03_ESTUDIANTE_PTT
14	HU02_03_04_DOCENTE_PTT
15	HU02_03_05_EDITAR_PTT
16	HU02_03_06_BUSCAR_PTT
17	HU02_03_07_BUSCAR_PLAN_ESTUDIANTE
18	HU02_03_08_BUSCAR_PLAN_DOCENTE
19	HU02_03_09_BUSCAR_HISTORIAL_PLAN
20	HU02_04_DESACTIVAR
21	HU02_05_CAMPOS_CONFIGURABLES
22	HU02_06_ANULACION_PTT

MÓDULO DE ENVÍO DE CORREOS ELECTRÓNICOS

23	HU03_01_CUERPO_CORREO
24	HU03_02_CORREO_INGRESO
25	HU03_03_CORREO_MIEMBRO
26	HU03_04_CORREO_OBSERVACIONES
27	HU03_05_CORREO_NEGACION_PTT
28	HU03_06_CORREO_APROBACION_PTT

MÓDULO DE GESTIÓN DE RÚBRICAS

29	HU04_01_RUBRICA_CREACION
30	HU04_02_RUBRICA_PORCENTAJES
31	HU04_03_RUBRICA_EVALUACION
32	HU04_04_RUBRICA_PDF

Figura 2.3. Historias de Usuario

En la Tabla 2.1, la Tabla 2.2, la Tabla 2.3 y la Tabla 2.4 se exponen los enunciados de las historias de usuario generadas; estos enunciados se han definido a través del rol que las cumple, la funcionalidad a desarrollar y el resultado a cumplir, esta información se muestra en el Anexo B.

En la Tabla 2.1 se muestran los enunciados de las historias de usuario correspondientes al Módulo de Autenticación y Roles, estas historias de usuario definen la gestión de roles y la gestión de usuarios de la aplicación, además se especifica que el sistema de autenticación genere una nueva contraseña en caso de que el usuario no recuerde cuál era su contraseña.

En la Tabla 2.2 se exponen los enunciados de las historias de usuario generadas para el Módulo de Gestión de la Información, en este módulo se observa una subdivisión entre historias de usuario correspondientes a la gestión de estudiantes, docentes y planes de Trabajos de Titulación.

Tabla 2.1. Enunciados de las Historias de Usuario del Módulo de Autenticación y Roles

Identificador (ID) de la Historia	Enunciado de la Historia
HU01_01_USUARIOS	Como administrador del sistema necesito gestionar usuarios (crear, editar, mostrar y eliminar) para permitir el ingreso de los mismos al sistema.
HU01_02_ROLES	Como administrador del sistema necesito gestionar roles para asignar a los usuarios un determinado rol y discriminar el acceso a cierta información en la aplicación según el rol establecido.
HU01_03_AUTENTICACION	Como usuario del sistema necesito digitar mi correo electrónico y mi contraseña para poder autenticarme e ingresar al sistema.
HU01_04_OLVIDO_CONTRASEÑA	Como usuario del sistema necesito poder generar una nueva contraseña a través de un enlace enviado vía correo electrónico para poder ingresar al sistema con la nueva contraseña generada.

Tabla 2.2. Enunciados de las Historias de Usuario del Módulo de Gestión de la Información

Identificador (ID) de la Historia	Enunciado de la Historia
HU02_01_01_CREAR_ESTUDIANTE	Como secretaria necesito ingresar los datos: Cédula, Número Único (opcional), Nombres, Apellidos, Carrera, Número de Teléfono y Correo Electrónico para poder crear un nuevo registro de estudiante.
HU02_01_02_EDITAR_ESTUDIANTE	Como secretaria necesito poder editar los campos del registro estudiante para actualizar y corregir información errónea.

HU02_01_03_BUSCAR_ESTUDIANTE	Como secretaria necesito poder buscar un registro estudiante con el apellido como parámetro de búsqueda para visualizar la información de dicho registro.
HU02_02_01_CREAR_DOCENTE	Como secretaria necesito ingresar los datos: Cédula, Nombres, Apellidos y Correo Electrónico para poder crear un nuevo registro de docente.
HU02_02_02_EDITAR_DOCENTE	Como secretaria necesito poder editar los campos del registro docente para actualizar y corregir información errónea.
HU02_02_03_BUSCAR_DOCENTE	Como secretaria necesito poder buscar un registro docente con el apellido como parámetro de búsqueda para visualizar la información de dicho registro.
HU02_03_01_CREAR_PTT	Como secretaria necesito ingresar los datos: Título, Tipo de Trabajo de Titulación, Línea de Investigación para poder crear un nuevo registro de Plan de Trabajo de Titulación (PTT).
HU02_03_02_PDF_PTT	Como secretaria necesito cargar un archivo PDF que contenga el plan escaneado para registrar un nuevo PTT y poder enviar el plan a los miembros de la Comisión posteriormente.
HU02_03_03_ESTUDIANTE_PTT	Como secretaria necesito buscar y elegir la información de uno a cuatro estudiantes para incluir esta información durante el registro de un PTT.
HU02_03_04_DOCENTE_PTT	Como secretaria necesito elegir la información de un docente para asignar un director, y opcionalmente de otro más para asignar un codirector, para incluir esta información durante el registro de un PTT.
HU02_03_05_EDITAR_PTT	Como secretaria necesito poder editar los campos del registro de un PTT para actualizar y corregir información errónea.
HU02_03_06_BUSCAR_PTT	Como secretaria necesito poder buscar uno o más registros de un PTT usando diferentes filtros para visualizar la información de dichos registros.
HU02_03_07_BUSCAR_PLAN_ESTUDIANTE	Como secretaria necesito poder buscar los PTT que ha presentado un estudiante usando el apellido del mismo como parámetro de búsqueda para visualizar dichos registros.
HU02_03_08_BUSCAR_PLAN_DOCENTE	Como secretaria necesito poder buscar los PTT que ha dirigido o codirigido un docente, usando el apellido del mismo como parámetro de búsqueda para visualizar dichos registros.
HU02_03_09_BUSCAR_HISTORIAL_PLAN	Como secretaria necesito poder buscar el historial de evaluación de uno o más PTT para poder visualizar dichos registros.
HU02_04_DESACTIVAR	Como secretaria necesito poder especificar el estado de los registros estudiante, docente o PTT,

	ya sea este inactivo o activo para evitar eliminar registros y solo mantener la información desactivada en caso de ya no ser necesaria.
HU02_05_CAMPOS_CONFIGURABLES	Como administrador necesito poder crear, actualizar y desactivar los campos: Tipo de Trabajo de Titulación, Línea de Investigación y Carrera (Estudiante) para poder gestionar estos datos en caso de que cambie en un futuro dicha información.
HU02_06_ANULACION_PTT	Como secretaria necesito poder registrar la anulación de un PTT por parte de un estudiante para poder cambiar el estado de un PTT a: Anulado.

En la Tabla 2.3 se presentan los enunciados de las historias de usuario del Módulo de Envío de Correos Electrónicos, en este módulo se definen las situaciones en las que se enviará un correo electrónico y la personalización del texto del cuerpo del correo electrónico.

Tabla 2.3. Enunciados de las Historias de Usuario del Módulo de Envío de Correos Electrónicos

Identificador (ID) de la Historia	Enunciado de la Historia
HU03_01_CUERPO_CORREO	Como administrador necesito poder cambiar el texto de los cuerpos de los correos electrónicos para personalizar dicha información.
HU03_02_CORREO_INGRESO	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar a estos del ingreso del PTT en el sistema.
HU03_03_CORREO_MIEMBRO	Como secretaria necesito enviar un correo electrónico con el PTT adjunto en formato PDF a los miembros de la comisión para que estos cuenten con información del plan previamente a la evaluación del mismo.
HU03_04_CORREO_OBSERVACIONES	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar los cambios que debe hacer al PTT para poder aprobarlo.
HU03_05_CORREO_NEGACION_PTT	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar que el PTT ha sido negado por la comisión.
HU03_06_CORREO_APROBACION_PTT	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar que el PTT ha sido aprobado por la comisión.

En la Tabla 2.4 se presentan los enunciados de las historias de usuario del Módulo de Gestión de Rúbricas. En este módulo se define la creación y el uso de rúbricas para evaluar un determinado plan de Trabajo de Titulación.

Tabla 2.4. Enunciados de las Historias de Usuario del Módulo de Gestión de Rúbricas

Identificador (ID) de la Historia	Enunciado de la Historia
HU04_01_RUBRICA_CREACION	Como administrador necesito crear una rúbrica en donde se definan criterios de evaluación y el valor máximo que se le puede asignar a cada criterio para poder evaluar un plan.
HU04_02_RUBRICA_PORCENTAJES	Como administrador debo asignar en la rúbrica creada un porcentaje de valoración de criterios para los estados de evaluación: negado, pendiente por aprobar y aprobado; para poder determinar el estado del plan luego de la evaluación.
HU04_03_RUBRICA_EVALUACION	Como administrador necesito usar una rúbrica en donde se dé una valoración a cada criterio y una observación (de ser necesario) para evaluar un determinado plan de Trabajo de Titulación.
HU04_04_RUBRICA_PDF	Como secretaria necesito generar un archivo PDF que contenga la rúbrica del plan evaluado para poder notificar al estudiante vía correo electrónico de las observaciones a su plan.

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN

En este capítulo se presentan los diagramas que han sido parte de la etapa de diseño y además segmentos de código que ejemplifican la implementación de la aplicación.

3.1 DISEÑO

A continuación se presentan los diagramas realizados para este proyecto.

3.1.1 DIAGRAMA DE SECUENCIA

En la Figura 3.1 se presenta un diagrama de secuencia del proceso MVC que ocurre cada vez que un usuario hace una petición en la aplicación web.

Las peticiones realizadas por un usuario se pueden evidenciar con interacciones tales como: hacer clic en un enlace, especificar una dirección URL[19] en la barra de direcciones del navegador, enviar la información de formularios al hacer clic en un botón, entre otras.

Estas peticiones son enviadas desde el cliente a través de métodos HTTP, los métodos HTTP comúnmente usados son GET y POST. Por lo general GET es usado para obtener información del servidor, mientras que POST es usado para enviar información hacia el servidor.

Tal como se observa en la Figura 3.1 la petición realizada por el usuario provoca la ejecución de una acción dentro del controlador, la localización de dicha acción y dicho controlador es especificada por medio de una URL.

En el controlador se ejecutará la lógica necesaria para responder a la solicitud del usuario a través de un método; este método tendrá una relación unívoca con la acción que se requiere cumplir, acción que puede asociarse a una petición GET o

[19] URL (*Uniform Resource Locator)*: Conjunto de caracteres definidos por un estándar que permiten localizar recursos dentro de una red.

a una petición POST. Para poder ejecutar la acción solicitada es necesario interactuar con los datos del sistema, es aquí donde el modelo permite manipular la información necesaria e interactuar con la base de datos. El modelo también se encargará de la validación de datos ingresados en casos como el de envío de información de formularios.

Una vez realizada la acción que el usuario solicitó, se procede a retornar la información necesaria a través de la presentación de la vista elegida en el método asociado a dicha acción. Esta vista será cargada por el navegador web a través de un documento HTML y permitirá al usuario seguir interactuando con la aplicación web.

El diagrama de secuencia mostrado en la Figura 3.1 representa el proceso MVC correspondiente a ambas peticiones GET y POST, ya que para estos dos tipos de peticiones el proceso es el mismo; la diferencia radica en que para el caso de GET, la base de datos carga los datos pedidos y devuelve esta información, mientras que para el caso de POST, en la base de datos se crean o actualizan registros con la información generada por el usuario.

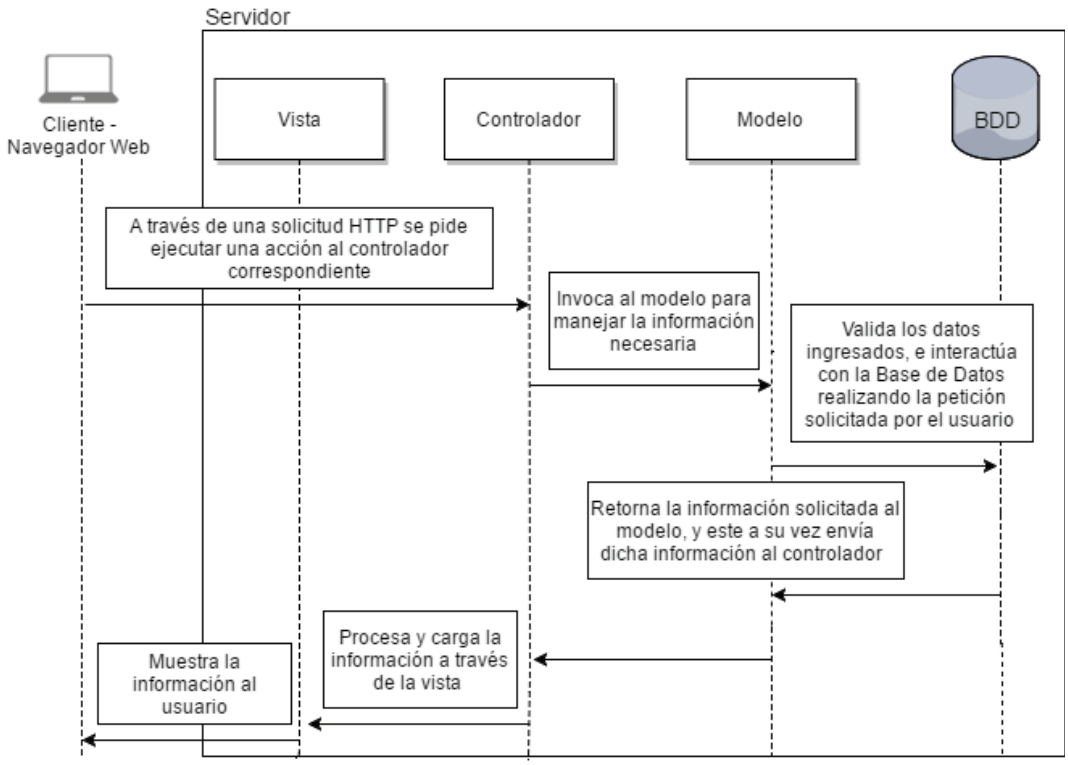

Figura 3.1. Diagrama de Secuencia del proceso MVC

3.1.2 DIAGRAMA ENTIDAD – RELACIÓN

En la Figura 3.2 se presenta el diagrama entidad-relación de la base de datos del sistema. La base de datos corresponde a los datos manipulados por los módulos: Gestión de Información, Envío de Correos Electrónicos y Gestión de Rúbricas. La información del módulo Autenticación y Roles no se muestra debido a que estos datos forman parte de otra base de datos implementada automáticamente al hacer uso del *framework* ASP.NET Identity, el cual permite manejar un sistema de autenticación para proyectos de Visual Studio.

Para el Módulo de Gestión de Información se observa la definición de tres entidades principales: Estudiante, Docente y Plan de TT. La entidad Estudiante se asocia con la entidad Carrera de tal forma que un estudiante solo pueda estar en una Carrera al momento de presentar un determinado plan, por otro lado se entiende que una Carrera consta de varios estudiantes. La entidad Plan de TT se asocia con las entidades Tipo de TT y Línea de Investigación de tal forma que un plan es realizado para un tipo de Trabajo de Titulación y permite ser desarrollado bajo una sola línea de investigación. Para la relación entre Plan de TT y Estudiante, y entre Plan de TT y Docente, se establece una cardinalidad de muchos a muchos ya que un plan puede ser desarrollado por varios estudiantes y puede contar con dos docentes asociados en las figuras de Director y Codirector, y además un estudiante puede realizar varios planes, en caso de no aprobar un plan en primera instancia, y un docente puede dirigir o codirigir varios planes.

Para el módulo de Envío de Correos Electrónicos se ha definido solamente la entidad Cuerpo Correo Electrónico, la cual permite personalizar el texto del cuerpo de los correos a enviar.

Para el último módulo se tiene a dos entidades principales: Rúbrica y Rúbrica Evaluación La entidad Rúbrica se asocia con la entidad Estados Calificación a través de una cardinalidad de muchos a muchos ya que cada rúbrica debe definir un porcentaje para los posibles estados de un plan luego de que este ha sido evaluado; Rúbrica también se asocia con Criterios Nuevos a través de la cardinalidad uno a muchos ya que se deben definir varios criterios de evaluación al crear la rúbrica, sin embargo, estos criterios formarán parte de una sola rúbrica. Para la entidad Rúbrica Evaluación se han definido tres asociaciones. Para la

asociación entre Rúbrica y Evaluación y Rúbrica se ha utilizado una cardinalidad de muchos a uno ya que una rúbrica debe permitir evaluar varios planes o evaluar el mismo plan varias veces, es decir, debe convertirse en varias rúbricas de evaluación, pero cada rúbrica de evaluación se crea a partir de una sola rúbrica previamente definida. Para la asociación entre Rúbrica Evaluación y Plan TT también se define una cardinalidad de muchos a uno ya que la rúbrica de evaluación debe calificar un determinado plan, pero un plan puede ser evaluado con varias rúbricas de evaluación en el caso de no ser aprobado en una primera instancia.

La última asociación definida entre Rúbrica Evaluación y Criterios Evaluación cumple el mismo comportamiento de la asociación entre Rúbrica y Criterios Nuevos ya que para ambos casos las rúbricas deben manipular la información de los criterios utilizados, pero estos criterios solo corresponden a una sola rúbrica, de ahí la cardinalidad uno a muchos.

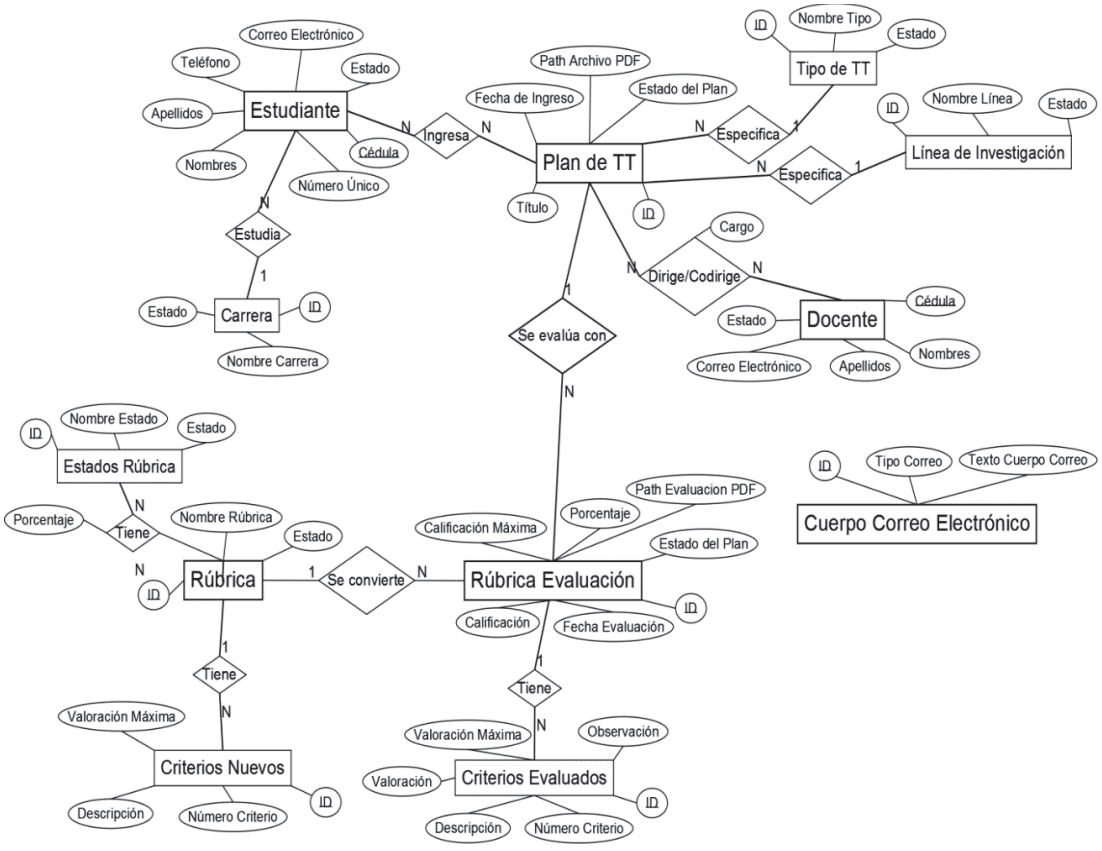

Figura 3.2. Diagrama Entidad-Relación

3.1.3 DIAGRAMA RELACIONAL

Se muestra a continuación un diagrama relacional que representa a la base de datos generada a través de Entity Framework. Como ya se mencionó anteriormente, el uso de Entity Framework *Code First* permite al desarrollador crear clases que representen a cada entidad previamente definida. Estas clases pasarán a ser denominadas como clases modelos o simplemente modelos.

Cada modelo estará relacionado con una tabla, y cada propiedad de la clase con una columna. El uso de objetos `DbSet` a través de la clase `Context` permitirá tener una persistencia entre las operaciones realizadas en el controlador y la base de datos, de esta manera las tablas serán creadas automáticamente vía *Scaffolding* siguiendo la convención de usar el nombre en plural del modelo para la denominación de cada tabla, así por ejemplo, el modelo `Estudiante` mediante la propiedad `DbSet` correspondiente creará la tabla `Estudiantes`.

En la Figura 3.3 y la Figura 3.4 se puede observar que un modelo cuenta con dos tipos de propiedades, las propiedades propiamente dichas, que representan a una columna dentro de la tabla correspondiente y las propiedades de navegación, que permiten la asociación entre varias entidades a través de la cardinalidad especificada.

En la Figura 3.3 se muestran los modelos creados para la manipulación de información dentro del Módulo de Gestión de Información. Se observa cómo se han creado dos entidades auxiliares a través de dos modelos denominados `PlanDocente` y `PlanEstudiante`, los cuales permiten cumplir la cardinalidad de muchos a muchos existente entre `Docente` y `PlanTT` y entre `Estudiante` y `PlanTT`, respectivamente.

Tomando como ejemplo al modelo `PlanEstudiante` se puede observar la existencia en este modelo de tres propiedades llamadas `ID`, `PlanTTID` y `EstudianteID`. La propiedad `ID` define la clave primaria de la tabla correspondiente, el denominar a una propiedad con la palabra “ID” o con el nombre del modelo concatenado a la palabra “ID” permite a Entity Framework interpretar a esta propiedad como la clave primaria. Las otras dos propiedades mencionadas permiten asociar uno o más registros de `Estudiante` con uno o

más registros de `PlanTT` usando las claves primarias de los registros correspondientes, es por esto que `PlanTTID` y `EstudianteID` pasan a ser claves foráneas en la tabla y se definen con el nombre del modelo al que asocian concatenado con la palabra "ID". Las propiedades de navegación `PlanTT` y `Estudiante` permitirán asociar los registros entre un plan y un estudiante.

Para el caso del modelo `PlanDocente` se observa la adición de una nueva propiedad denominada `Cargo`, la cual permite distinguir si el docente asociado a un determinado plan tiene el cargo de Director o de Codirector.

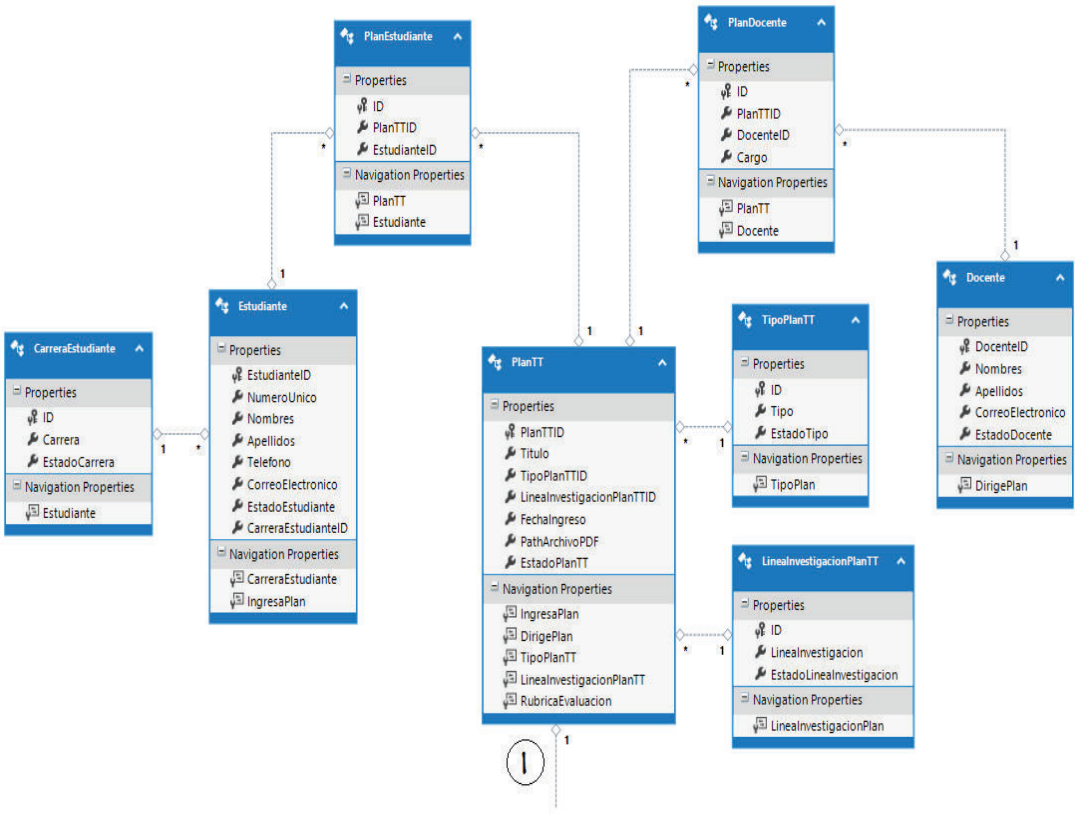

Figura 3.3. Diagrama Relacional

En la Figura 3.4 se muestran los modelos y sus asociaciones para el Módulo de Gestión de Rúbricas y también se observa el modelo `EmailCuerpo` que forma parte del Módulo de Envío de Correos Electrónicos. Se observa en este caso la adición de un nuevo modelo llamado `RubricaEstado`, el cual permite asociar a los modelos `Rubrica` y `EstadoCalificacion` a través de la cardinalidad

muchos a muchos; para este caso se añade la propiedad `Porcentaje` para establecer los porcentajes de aprobación para cada estado y se definen las claves foráneas `RubricaID` y `EstadoCalificacionID` y la clave primaria `ID`. Cabe mencionar que el modelo `RubricaEvaluacion` permite interactuar a los Módulos Gestión de Información y Gestión de Rúbricas al definir la relación entre este modelo y el modelo `PlanTT` mediante una relación de muchos a uno entre ambos, esta relación se puede evidenciar mediante el uso de la clave foránea `PlanTTID`, propiedad que permite asociar a un registro de `RubricaEvaluación` el ID del plan evaluado.

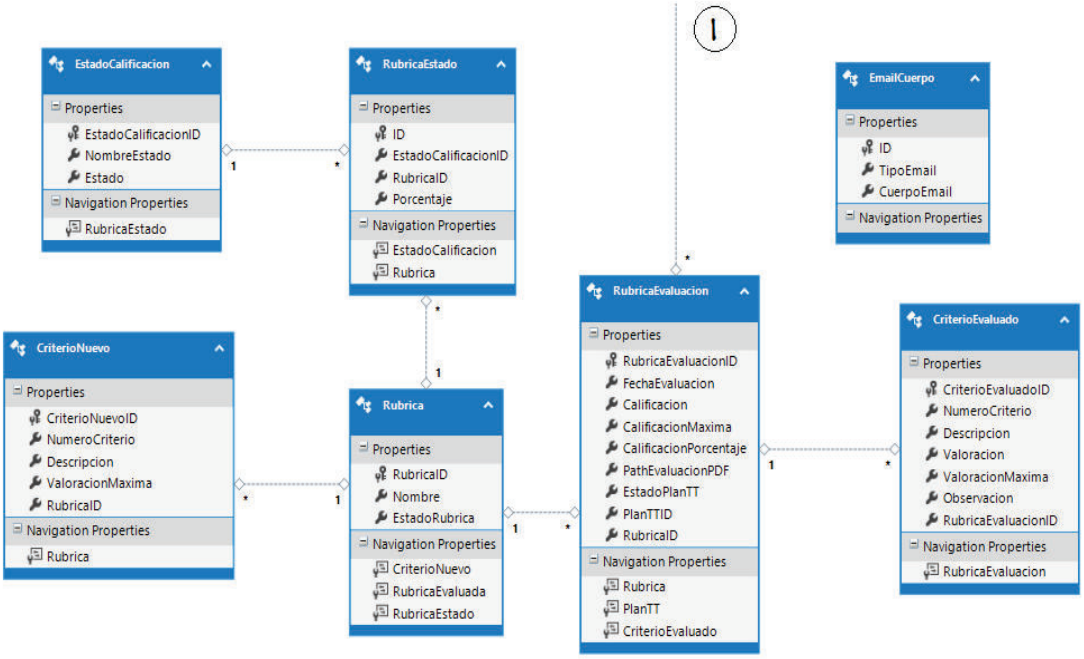

Figura 3.4. Diagrama Relacional (continuación)

3.1.4 DIAGRAMA DE CLASES

A continuación se presentan los diagramas de clases correspondientes al código de la aplicación. En la Figura 3.5 se presentan los diagramas de clases de los modelos `CarreraEstudiante`, `EmailCuerpo`, `PlanEstudiante`, `Estudiante`, `PlanDocente`, `Docente`, `TipoPlanTT`, `LineaInvestigacionPlanTT`, y `PlanTT`; para estas clases se muestran las propiedades definidas en la Figura 3.4.

Además se presenta el diagrama de la clase `EmailService`, la cual se usa para poder enviar un correo electrónico a través del *framework* ASP.NET Identity en el Módulo de Autenticación y Roles, y el diagrama de la clase `Context`, la cual hereda de `DbContext`, y define 15 propiedades `DbSet`, propiedades que permiten asociar a los modelos creados con las tablas de la base de datos. También se observa el diagrama de la clase `Configuration`, en la cual se define el método `Seed` que permite cargar datos por defecto en la aplicación, y el diagrama de clase del ViewModel[20] `VMPlanTT` que permite manejar información de diferentes modelos dentro de una vista.

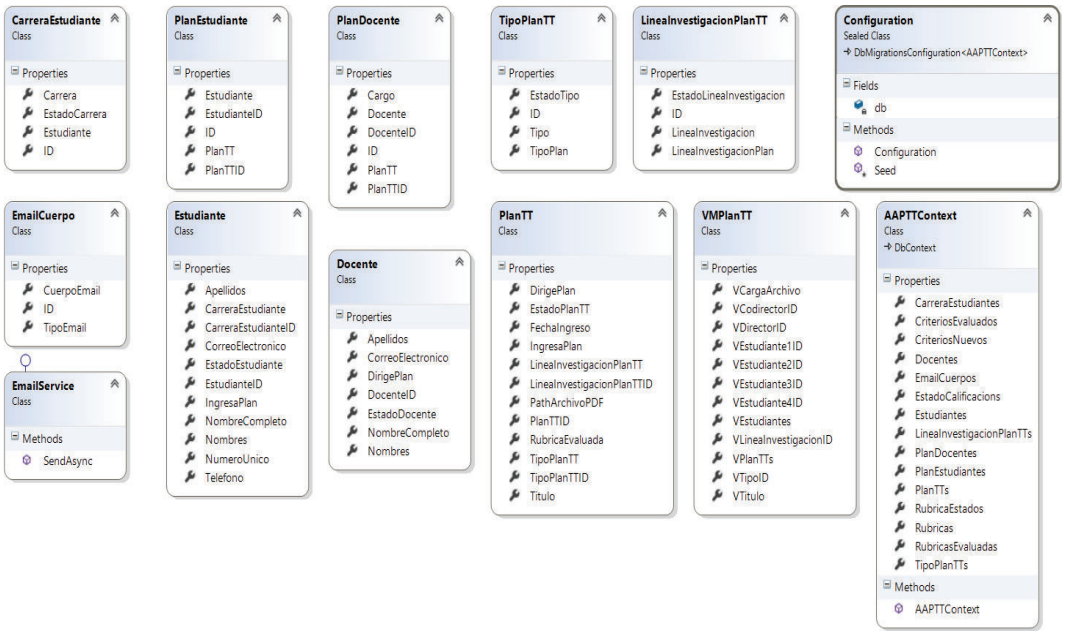

Figura 3.5. Diagramas de Clases de la aplicación

En la Figura 3.6 se observan los diagramas de clases de los modelos `RubricaEstado`, `Rubrica`, `CriterioNuevo`, `CriterioEvaluado`, `EstadoCalificacion` y `RubricaEvaluacion`. También se observa el diagrama de clase del ViewModel `VMRubrica` y de las clases `DatosCriterioEvaluado`, `DatosRubricaEvaluacion`, `DatosJSON` y `DatosJSONRubrica`, las cuales permiten manejar la recepción de datos JSON

[20] ViewModel: Representa la información necesaria a usarse en una vista. Difiere de un modelo tradicional ya que sus propiedades no representan una columna de una tabla.

enviados desde el lado del cliente al momento de evaluar un plan con una determinada rúbrica de evaluación.

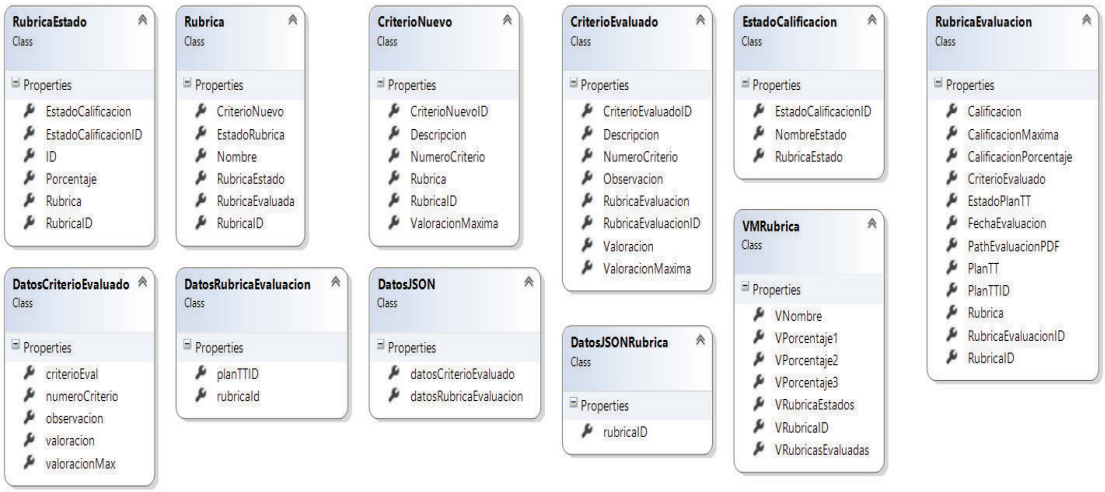

Figura 3.6. Diagramas de Clases de la aplicación (continuación)

En la Figura 3.7 se presentan los diagramas de clases de los controladores usados en el Módulo de Gestión de la Información, los cuales han sido generados vía *Scaffolding* a partir de los modelos correspondientes. Todos estos controladores definen una propiedad `db`, que representa una instancia de la clase `Context` y mediante la cual se puede interactuar con la base de datos. Se puede ver que la mayoría de estos controladores implementan los métodos `Create`, `Details`, `Edit` e `Index`, los cuales permiten manejar las acciones solicitadas por el usuario, y el método `Dispose` que permite liberar recursos utilizados al finalizar la solicitud. Para los métodos `Create` y `Edit` se especifica adicionalmente un método sobrecargado, el método sin parámetros permitirá manejar la solicitud HTTP GET y cargar el formulario correspondiente, mientras que el método sobrecargado permitirá a través de una solicitud HTTP POST enviar la información del formulario al servidor para crear o editar registros.

Para el controlador `PlanTTsController` se evidencia el uso de otros métodos como `Anulacion`, que permite cambiar el estado de un plan a “Anulado”, situación que puede ocurrir si un estudiante decide anular su plan presentando la documentación necesaria en la Secretaría de la Carrera. También se observa el

uso de métodos como `EmailIngreso` o `EmailIngresoMiembro`, los cuales permiten enviar las notificaciones correspondientes vía correo electrónico a los estudiantes y directores/codirectores del plan. Además se muestran métodos como `_BuscaryAgregar`, el mismo que permite asociar un estudiante a un plan al momento de crear un nuevo registro. Los controladores `PlanEstudiantesController` y `PlanDocentesController` son usados solo para mostrar la información de un estudiante y los planes que ha realizado o de un docente y los planes que ha dirigido o codirigido, por tal razón no se implementan los métodos `Create` y `Edit`.

También se muestra el controlador `HomeController`, el cual permite cargar a través de sus métodos `About`, `Contact` e `Index` vistas que contengan información acerca del desarrollador de la aplicación, información de contacto y la información inicial que se muestra cuando se ingresa por primera vez a la aplicación, respectivamente.

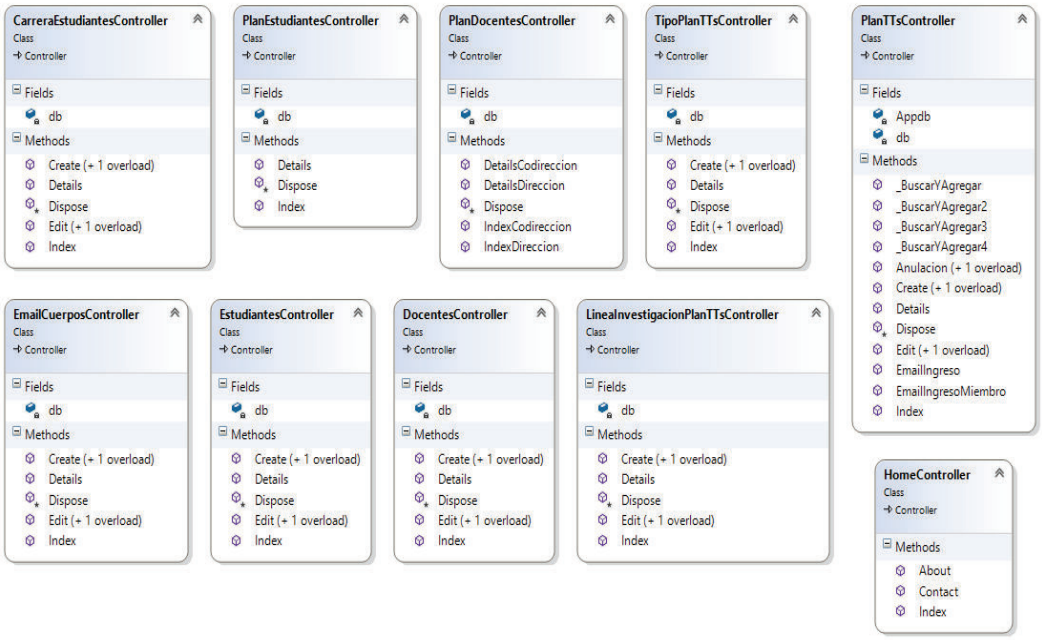

Figura 3.7. Diagramas de clases de los controladores de la aplicación

En la Figura 3.8 se muestran los controladores generados para el Módulo de Gestión de Rúbricas. Se observa que los modelos `CriterioNuevo` y

`CriterioEvaluado` no generan controladores, ya que la información de dichos modelos se utiliza al momento de crear una rúbrica y de usar la misma para evaluar un plan a través de `RubricasController`.

En `RubricasController` se muestra el método `_ElegirPTT` que permite elegir el plan a evaluar con una determinada rúbrica, y además los métodos `Criterios`, `CriteriosEvaluacion`, `ObtenerCriterios` y `GuardarEvaluacionRubrica` que permiten manipular la información de las rúbricas al momento de crear y usar una rúbrica.

Para el controlador `RubricaEvaluacionsController` se muestra el método `GenerarPDF`, el cual permite generar un archivo PDF que muestre la rúbrica usada para la evaluación de un determinado plan. Este archivo será enviado vía correo electrónico al estudiante y director/codirector del plan indicando las observaciones hechas al plan por parte de la Comisión; el envío de este correo electrónico será gestionado a través del método `EmailEvaluacion`.

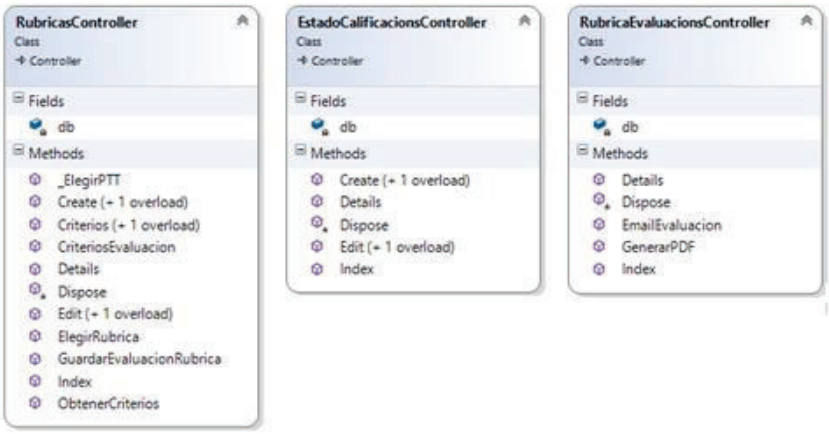

Figura 3.8. Diagramas de clases de los controladores de la aplicación (continuación)

3.2 IMPLEMENTACIÓN

3.2.1 MÓDULO 1: AUTENTICACIÓN Y ROLES DE USUARIO

La primera funcionalidad a desarrollar para la aplicación web fue el módulo de autenticación, el mismo que definió la gestión de usuarios y roles de usuarios. El IDE Visual Studio Express 2015 for Web permite crear aplicaciones web ASP.NET

que implementen por defecto un sistema de autenticación denominado *Individual User Accounts* tal como se observa en la Figura 3.9. Sin embargo existen otras opciones que se muestran en la Figura 3.10, estas opciones son: *Windows Authentication*, que está dirigido para aplicaciones de intranets, *Work and School Accounts,* que permite a los usuarios autenticarse con sistemas como Office 365 o Microsoft Azure Active Directory, y *No Authentication*, la cual no implementa ningún sistema de autenticación.

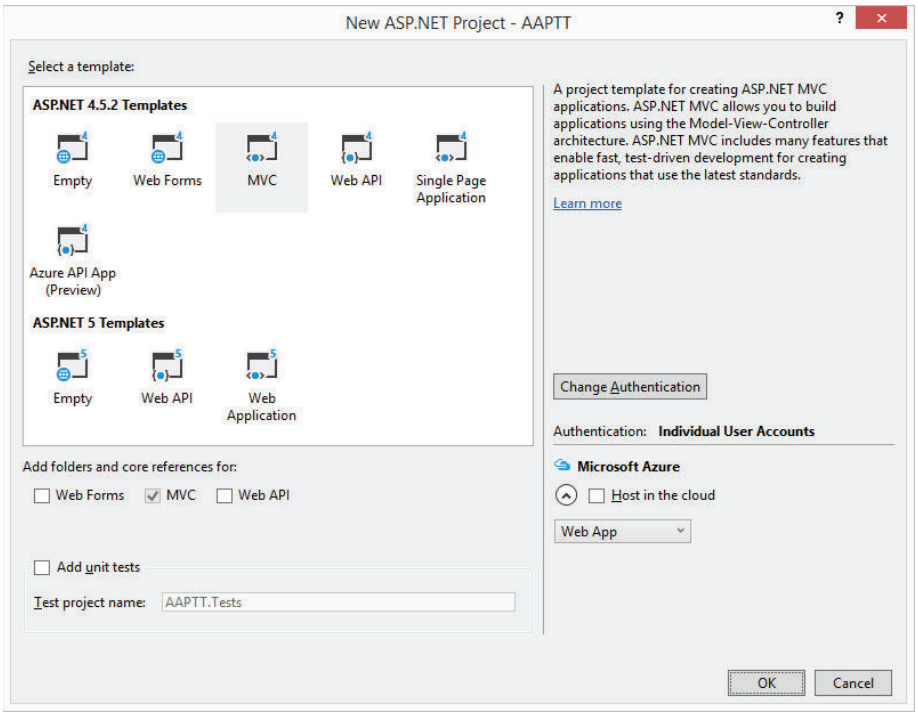

Figura 3.9. Creación de un Nuevo Proyecto en Visual Studio

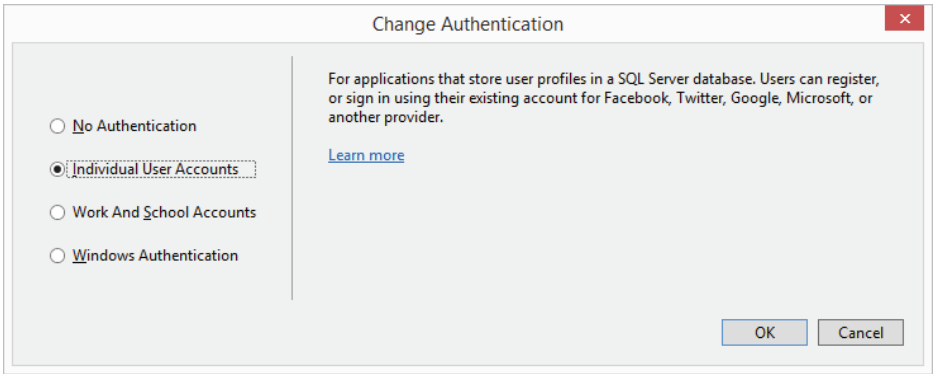

Figura 3.10. Sistemas de Autenticación para la Aplicación Web

La característica principal de *Individual User Accounts* es la de almacenar perfiles de usuarios dentro de una base de datos SQL Server a través del *framework* denominado ASP.NET Identity.

En marzo de 2014 fue lanzada la versión 2.0 de este *framework*, la cual presentó algunas nuevas características como: autenticación de doble factor o verificación de dos pasos, bloqueo de autenticación luego de varios intentos fallidos, soporte para el ingreso a través de perfiles de redes sociales, confirmación de ingreso al sistema vía correo electrónico, gestión de usuarios y roles, entre otras.

El uso de ASP.NET Identity permitió cumplir con los requerimientos solicitados a través de las historias de usuario correspondientes a este módulo, entre los cuales se encuentran la autenticación de usuarios, la gestión de usuarios y roles, y la posibilidad de restablecer una contraseña en caso de que el usuario la haya olvidado a través de un enlace enviado a su correo electrónico.

El *framework* ASP.NET Identity cuenta con su propia base de datos, su propia clase `Context` y sus propias estructuras correspondientes a modelos, vistas y controladores que permiten cumplir con las funcionalidades encomendadas. Cabe mencionar que el uso del middleware OWIN[21] facilita la implementación de los componentes que se encargarán de crear la lógica necesaria, por tal razón, las clases implementadas a través de ASP.NET Identity se valdrán de componentes externos que permitan gestionar el sistema de autenticación previamente mencionado.

A continuación se muestran ejemplos del uso de ASP.NET Identity para atender los requerimientos del presente módulo.

3.2.1.1 Creación de un Rol

Una de las funcionalidades cubiertas por parte de ASP.NET Identity es la gestión de roles dentro del sistema. A continuación se muestra el proceso de la creación de un rol a través del controlador, el modelo y la vista correspondiente.

Dentro del contexto MVC, las solicitudes hechas por el usuario se manejan a través de acciones, ya sean estas solicitudes hechas a través del método HTTP

[21] OWIN: (*Open Web Interface for* .NET) Es una especificación de código abierto que describe una capa de abstracción entre servidores web y componentes de la aplicación.

GET o del método HTTP POST. Para cumplir con la acción correspondiente se implementan métodos llamados métodos de acción, los cuales a través de la clase `ActionResult` permiten retornar información a través de la presentación de una vista asociada al método de acción.

En el Código 3.1 se muestra el método `Create`, el cual maneja una solicitud HTTP GET que requiere cargar una vista para la creación de un nuevo rol en el sistema. En la línea 91 se observa la definición del método `Create` de tipo `ActionResult`, método que permitirá manejar la acción solicitada por el usuario, mientras que en la línea 93 se observa el uso del método `View`, el cual permitirá cargar la vista asociada al método `Create`, ya que el método `View` es de tipo `ViewResult`, clase que hereda de `ActionResult`.

```
90          // GET: /Roles/Create
91          public ActionResult Create()
92          {
93              return View();
94          }
```

Código 3.1. Método `Create` (Solicitud GET)

Para poder cargar la vista asociada a un método de acción es necesario definir un modelo que permita manipular la información ingresada por el usuario en la vista. En ciertas ocasiones se crean modelos que no están relacionados con una tabla en específico, sino que más bien sirven para representar la información que se quiere mostrar en la vista, estos modelos son conocidos como ViewModels.

En el Código 3.2 se observa el uso del ViewModel `RoleViewModel` que define las propiedades `Id` (línea 9) y `Name` (línea 12). Para la propiedad `Name` se especifican los atributos `Required` (línea 10) y `Display` (línea 11). El atributo `Required` permite a través de su propiedad `AllowEmptyStrings` validar que la información ingresada no este vacía, mientras que el atributo `Display` a través de su propiedad `Name` muestra en este caso el texto con el que se cargará la propiedad en la vista, mediante los respectivos *helpers* que pueden ser `HTML.Label` o `HTML.Display`.

```
7     public class RoleViewModel
8     {
9         public string Id { get; set; }
10        [Required(AllowEmptyStrings = false)]
11        [Display(Name = "Rol")]
12        public string Name { get; set; }
13    }
```

Código 3.2. ViewModel `RoleViewModel`

En el Código 3.3 se muestra la vista que permitirá la creación de un nuevo rol. A esta vista se la denomina con el mismo nombre del método que está asociada, es decir, `Create`.

En la línea 1 se observa el uso de la directiva `@model`, la cual permite referenciar a `RoleViewModel` dentro de la vista. Más adelante se define el título del documento HTML (línea 3 a la línea 5) y el título presentado en la vista que indica la operación a realizarse (línea 7).

A partir de la línea 10 se define el formulario que permitirá al usuario crear el registro. En la línea 12 a través de `Html.AntiForgeryToken` se busca proteger a la aplicación del ataque *Cross-Site Request Forgery*[22]. A partir de la línea 14 se observa el uso de etiquetas `<div>` que permitirán crear secciones en la página utilizando estilos definidos por Bootstrap.

Más adelante se muestra un subtítulo que define el tipo de registro a crearse (línea 15) y a través de `Html.ValidationSummary` (línea 17) se crean mensajes de validación para los campos que lo especifiquen dentro del formulario.

Este formulario cuenta con un solo campo a llenar, el cual cuenta con una etiqueta que se implementa con `Html.LabelFor` (línea 20) y un cuadro de texto que se define a través de `Html.TextBoxFor`. Este cuadro de texto permite al usuario digitar el nombre del rol a crear (línea 22). Más adelante se define un mensaje de validación a través de `Html.ValidationMessageFor`, el mismo

[22] *Cross-Site Request Forgery*: Es un tipo de ataque en el que se aprovecha el envío de *tokens* de autenticación desde el navegador al servidor web para poder influenciar la interacción entre ambos y falsificar información.

que se mostrará en caso de que exista un error en la validación de los datos (línea 23).

Además se muestra el código de un botón denominado “Crear”, el cual permitirá guardar la información ingresada una vez que se da clic sobre él (línea 29). Finalmente se crea un enlace que llama al método de acción `Index`, método que permitirá cargar la vista donde se muestran los roles creados hasta el momento (línea 36).

```
1   @model AAPTT.Models.RoleViewModel
2
3   @{
4       ViewBag.Title = "Crear";
5   }
6
7   <h2>Crear</h2>
8
9
10  @using (Html.BeginForm())
11  {
12      @Html.AntiForgeryToken()
13
14      <div class="form-horizontal">
15          <h4>Rol</h4>
16          <hr />
17          @Html.ValidationSummary(true)
18
19          <div class="form-group">
20              @Html.LabelFor(model => model.Name, new { @class = "control-label col-md-2" })
21              <div class="col-md-10">
22                  @Html.TextBoxFor(model => model.Name, new { @class = "form-control" })
23                  @Html.ValidationMessageFor(model => model.Name)
24              </div>
25          </div>
26
27          <div class="form-group">
28              <div class="col-md-offset-2 col-md-10">
29                  <input type="submit" value="Crear" class="btn btn-default" />
30              </div>
31          </div>
32      </div>
33  }
34
35  <div>
36      @Html.ActionLink("Regresar a la lista", "Index")
37  </div>
```

Código 3.3. Vista `Create`

En el Código 3.4 se muestra el método de acción `Create` correspondiente a la solicitud POST del usuario, esta solicitud se la realiza al dar clic en el botón "Crear" de la vista `Create` (Código 3.3). La mayoría de métodos de acción de ASP.NET Identity implementan operaciones asíncronas definidas a través de la clase `Task` y los operadores `async` y `await`. En un inicio se observa que `Create` está definido como un método de acción asíncrono de tipo `Task` que debe retornar un objeto `ActionResult`, y que recibe como argumento un objeto de tipo `RoleViewModel` (línea 99). Luego de cerciorarse que el modelo manipulado no contiene errores (línea 101) se obtiene la información definida en el objeto de tipo `RoleViewModel` (línea 103) para luego crear el registro correspondiente a través del método asíncrono `CreateAsync` (línea 104). Si el resultado de la creación del registro es exitoso se llama al método de acción `Index` (línea 110). Además si el modelo pasado al método contiene errores o si la creación del registro no ha sido exitosa (línea 105 a la 109), se procede a cargar de nuevo la vista `Create` (líneas 108 y 112).

```
96          //
97          // POST: /Roles/Create
98          [HttpPost]
99          public async Task<ActionResult> Create(RoleViewModel roleViewModel)
100         {
101             if (ModelState.IsValid)
102             {
103                 var role = new IdentityRole(roleViewModel.Name);
104                 var roleresult = await RoleManager.CreateAsync(role);
105                 if (!roleresult.Succeeded)
106                 {
107                     ModelState.AddModelError("", roleresult.Errors.First());
108                     return View();
109                 }
110                 return RedirectToAction("Index");
111             }
112             return View();
113         }
```

Código 3.4. Método `Create` (Solicitud POST)

3.2.1.2 Generación de una nueva contraseña

ASP.NET Identity permite que un usuario que no pueda ingresar al sistema por haber olvidado su contraseña, genere una nueva a través del envío de un enlace

al correo electrónico, sin embargo, es necesario definir un agente externo que permita el envío de dicho correo electrónico. Para hacer uso de este servicio se ha utilizado SendGrid [18]. SendGrid es una de las plataformas más usadas para envíos de correos electrónicos, su funcionamiento basado en la nube se caracteriza por brindar una entrega confiable y ágil de correos electrónicos [19], además el uso de SendGrid es muy recomendado por desarrolladores que han usado el *framework* ASP.NET Identity.

En el Código 3.5 se presenta el uso de SendGrid a través de la clase `EmailService`, la cual forma parte del sistema de autenticación definido por ASP.NET Identity. En un inicio se observa a la clase `EmailService` que hereda de la interfaz `IIdentityMessageService` (línea 70), esto se realiza para poder utilizar el método `SendAsync`, el cual se define en la interfaz mencionada. El método `SendAsync` es de tipo `Task` y representa una tarea asíncrona que será ejecutada en un hilo diferente al hilo principal de ejecución, este método recibe como parámetro un objeto de tipo `IdentityMessage` (línea 72). El objeto `IdentityMessage` define 3 propiedades: `Body`, `Destination` y `Subject`, las cuales almacenan la información del cuerpo, el destinatario y el asunto de un correo electrónico, respectivamente. Más adelante se configura el cliente SMTP mediante un objeto de la clase `SmtpClient`, la cual permite enviar correos electrónicos a través del protocolo SMTP[23]. Al crearse este objeto se define la dirección del servidor SMTP y el puerto que se usará para el envío de correos electrónicos (línea 75), además se evita usar las credenciales que vienen por defecto (línea 76) para usar como credenciales el nombre de usuario y contraseña de la cuenta SendGrid asociada, estos datos al ser información sensible serán manejados mediante un archivo de configuración (línea 79 a la 82).

Más adelante se define una dirección de envío (línea 85) y el correo a enviar a través de una instancia de la clase `MailMessage`, la cual define el remitente y el destinatario del correo electrónico (línea 86) para después asignar el asunto y el cuerpo del mismo (líneas 87 y 88). Por último a través del método `SendMailAsync` se permite el envío del correo electrónico (línea 91).

[23] SMTP (*Simple Mail Transfer Protocol)*: Protocolo de red utilizado para el intercambio de correos electrónicos.

```
70      public class EmailService : IIdentityMessageService
71      {
72          public Task SendAsync(IdentityMessage message)
73          {
74              // Configuración del cliente
75              var client = new SmtpClient("smtp.sendgrid.net", Convert.ToInt32(587));
76              client.UseDefaultCredentials = false;
77
78              // Creación de las credenciales
79              var sendGridUserName = ConfigurationManager.AppSettings["usernameSendGrid"];
80              var sendGridPassword = ConfigurationManager.AppSettings["passwordSendGrid"];
81              NetworkCredential credentials = new NetworkCredential(sendGridUserName, sendGridPassword);
82              client.Credentials = credentials;
83
84              // Creación del mensaje
85              var sentFrom = "AAPTT_EPN@gmail.com";
86              var mail = new MailMessage(sentFrom, message.Destination);
87              mail.Subject = message.Subject;
88              mail.Body = message.Body;
89
90              // Envío del correo:
91              return client.SendMailAsync(mail);
92          }
93      }
```

Código 3.5. Uso de SendGrid para el envío de correos electrónicos

3.2.2 MÓDULO 2: GESTIÓN DE INFORMACIÓN

El desarrollo de este módulo se basó en la gestión de tres entidades importantes: Estudiante, Docente y PlanTT. Para cada una de estas tres entidades se implementaron métodos que permiten la creación, edición y búsqueda de un registro.

La eliminación no fue implementada ya que puede haber casos de docentes que actualmente no formen parte del personal de la carrera pero que posiblemente puedan volver a ser parte de esta en un futuro, también pueden haber casos de estudiantes que se retiren de una carrera y que en un futuro puedan reingresar.

En lugar de la eliminación de registros se manejaron estados, el estado del registro determina si dicho registro está activo o inactivo. Para los registros de planes de Trabajos de Titulación se manejaron los estados: Pendiente por aprobar, Aprobado, Negado y Anulado.

La gestión de registros en este módulo siguió un proceso común basado en *Code First* de Entity Framework. *Code First* define para cada entidad la creación del modelo correspondiente a través de una clase, y posteriormente mediante *Scaffolding,* la generación del controlador y las vistas que permitan al usuario ejecutar cada acción definida en los métodos de acción del controlador. Al utilizar

Scaffolding también se genera una propiedad `DbSet` dentro de la clase `Context`, esta propiedad permitirá crear e interactuar con la tabla correspondiente al modelo previamente creado.

En la creación de un registro para la entidad PlanTT se necesitó crear registros adicionales que permitan asociar la información de estudiantes y docentes a través de las tablas `PlanEstudiantes` y `PlanDocentes`. Para que todo este proceso sea transparente al usuario se procedió a utilizar ventanas modales, vistas parciales, funciones de JavaScript y AJAX, de forma que sin necesidad de manejar más vistas se envíe la información necesaria al servidor.

A continuación se describirá la creación de un registro para la entidad Estudiante a través del modelo, el controlador y la vista correspondiente, y además se mostrará la asociación dinámica de un estudiante al formulario correspondiente a la creación de un registro de plan de Trabajo de Titulación.

3.2.2.1 Creación de un registro Estudiante

Para representar la creación de un registro de la entidad Estudiante se describe a continuación el proceso realizado a través de *Code First* de Entity Framework, realizando en primer lugar la creación del modelo y luego la personalización del controlador y la vista correspondiente.

3.2.2.1.1 Creación del Modelo

La clase que corresponde al modelo asociado a la entidad Estudiante se denomina `Estudiante`. En el Código 3.6 se muestra de forma detallada las propiedades de esta clase.

La clase `Estudiante` contiene una propiedad por cada columna de la tabla `Estudiantes`, por tal razón se tienen las siguientes propiedades: `EstudianteID` (línea 13), `NumeroUnico` (línea 17), `Nombres` (línea 21), `Apellidos` (línea 25), `Telefono` (línea 30), `CorreoElectronico` (línea 35), `CarreraEstudianteID` (línea 39) y `EstadoEstudiante` (línea 42).

La propiedad `NombreCompleto` (línea 45) no corresponde a una columna, se la implementa para mostrar el nombre completo de un estudiante en caso de ser necesario. Por último se definen dos propiedades de navegación: `IngresaPlan`

(línea 47) y `CarreraEstudiante` (línea 49). Para el primer caso se define una colección de tipo `PlanEstudiante`, esta colección permitirá manejar varias entidades y establecer la cardinalidad de uno a muchos entre `Estudiante` y `PlanEstudiante`; para el segundo caso se define un objeto de tipo `CarreraEstudiante`, este objeto permitirá manejar una sola entidad y establecer la cardinalidad de muchos a uno entre `Estudiante` y `CarreraEstudiante`.

Cada propiedad puede utilizar atributos que permitan validar la información ingresada en los campos del formulario o también especificar variaciones dentro de la tabla asociada al modelo. Entity Framework por defecto implementa la propiedad SQL Identity[24] en la clave primaria, se observa en un inicio el uso del atributo `DatabaseGenerated` que a través de su propiedad `DatabaseGeneratedOption` establecida en `None` evita la asignación automática de la propiedad SQL Identity en la clave primaria (línea 9). Más adelante y mediante el uso del atributo `Required` y su propiedad `ErrorMessage` se especifica que el campo asociado a dicha propiedad no puede quedar vacío, de no cumplir con esto se presenta el mensaje de error correspondiente (líneas 10, 19, 23, 27, 32 y 37), además se establecen restricciones de longitud de caracteres para campos que necesitan tener un número mínimo y máximo de caracteres a través del atributo `StringLength` (líneas 11, 15 y 29). Por último se representa a través del atributo `Display` el texto con el que se cargará cada propiedad en la vista (líneas 12,16, 20, 24, 28, 33, 38, 41 y 44).

Una vez creado el modelo se procede a generar el controlador y las vistas respectivas mediante la herramienta *Scaffolding* para posteriormente poder personalizar tanto el controlador como las vistas para implementar las funcionalidades deseadas.

3.2.2.1.2 Personalización del Controlador y la Vista

El controlador que se genera por defecto a través del *Scaffolding* implementa los métodos de acción `Create`, `Edit`, `Details`, `Index` y `Delete`. Como ya se

[24] Identity: Propiedad que genera para dicho registro un valor secuencial que empieza en uno y se incrementa en pasos de uno a medida que se crean nuevos registros.

mencionó anteriormente la presente aplicación no implementa la acción de eliminar por lo que se prescinde del método `Delete`.

```
7   public class Estudiante
8   {
9       [DatabaseGenerated(DatabaseGeneratedOption.None)]
10      [Required(ErrorMessage = "Este campo es obligatorio.")]
11      [StringLength(10, MinimumLength = 10, ErrorMessage = "Se deben ingresar 10 dígitos.")]
12      [Display(Name = "Cédula")]
13      public string EstudianteID { get; set; }
14
15      [StringLength(9, MinimumLength = 5, ErrorMessage = "Se deben ingresar de 5 a 9 dígitos.")]
16      [Display(Name = "Número Único")]
17      public string NumeroUnico { get; set; }
18
19      [Required(ErrorMessage = "Este campo es obligatorio.")]
20      [Display(Name = "Nombres")]
21      public string Nombres { get; set; }
22
23      [Required(ErrorMessage = "Este campo es obligatorio.")]
24      [Display(Name = "Apellidos")]
25      public string Apellidos { get; set; }
26
27      [Required(ErrorMessage = "Este campo es obligatorio.")]
28      [Display(Name = "Teléfono")]
29      [StringLength(10, MinimumLength = 7, ErrorMessage = "Se deben ingresar de 7 a 10 dígitos.")]
30      public string Telefono { get; set; }
31
32      [Required(ErrorMessage = "Este campo es obligatorio.")]
33      [Display(Name = "Correo Electrónico")]
34      [EmailAddress(ErrorMessage ="Ingrese una dirección de correo electrónico válida.")]
35      public string CorreoElectronico { get; set; }
36
37      [Required]
38      [Display(Name = "Carrera")]
39      public int CarreraEstudianteID { get; set; }
40
41      [Display(Name = "Estado")]
42      public bool EstadoEstudiante { get; set; }
43
44      [Display(Name = "Estudiante")]
45      public string NombreCompleto { get {return Apellidos + " " + Nombres;} }
46
47      public virtual ICollection<PlanEstudiante> IngresaPlan { get; set; }
48
49      public virtual CarreraEstudiante CarreraEstudiante { get; set; }
50  }
```

Código 3.6. Modelo `Estudiante`

A continuación se describe el código implementado para la creación de un registro de estudiante en el controlador y la vista correspondiente.

Para crear un registro de estudiante, el usuario debe dirigirse hacia el menú de Estudiantes, el cual cargará la vista `Index`, una vez allí al hacer clic en el enlace "Crear Nuevo" se cargará la vista con el formulario correspondiente, una vez ingresada la información necesaria en el formulario, el usuario hará clic en el botón que permita guardar esta información y volverá a la vista `Index`.

Para implementar este proceso el controlador hace uso de dos métodos de acción denominados `Create`, el primer método permitirá cargar la vista con el formulario a través de una solicitud HTTP GET, mientras que el segundo método guardará la información del formulario a través de una solicitud HTTP POST.

En el Código 3.7 se presenta el método `Create` que se usa cuando se realiza una solicitud HTTP GET por parte del usuario, este método permitirá cargar el formulario necesario para crear un nuevo registro a través de la vista asociada a este método.

Se observa inicialmente el uso del atributo `Authorize` (línea 101), el cual evita que un usuario que no esté autenticado acceda al formulario mediante su URL. Más adelante a través de una consulta LINQ se procede a cargar en el objeto `carreras` todos los registros de la tabla `CarreraEstudiantes` (línea 104) y luego se filtra la información que contenga registros con un estado activo (línea 107).

Esta información se cargará en la vista a través de una lista desplegable, para esto se define una lista de tipo `SelectList`, la cual especifica a través de sus argumentos: los registros a cargar en la lista desplegable, la propiedad del registro que se manipulará y la propiedad del registro que se mostrará en la lista desplegable. La información definida en `SelectList` será almacenada en una propiedad `ViewBag`[25] denominada `ViewBag.Carrera` (línea 109) que permitirá cargar la lista desplegable en la vista.

Finalmente se genera la vista correspondiente a través del método `View` (línea 110), este método carga la vista asociada al método, la cual también lleva el nombre `Create`.

[25] ViewBag: Es una propiedad dinámica que permite compartir información entre el controlador y la vista.

```
100         // GET: Estudiantes/Create
101         [Authorize]
102         public ActionResult Create()
103         {
104             var carreras = from c in db.CarreraEstudiantes
105                            select c;
106
107             carreras = carreras.Where(c => c.EstadoCarrera.Equals(true));
108
109             ViewBag.Carrera = new SelectList(carreras, "ID", "Carrera");
110             return View();
111         }
```

Código 3.7. Método `Create` de `EstudiantesController` (Solicitud HTTP GET)

La vista generada maneja la misma estructura que se presentó en el Código 3.3 en donde se define un título y subtítulo que permitan definir la operación que se realiza y la entidad que se manipula, posteriormente se presenta el formulario en donde cada propiedad del modelo generará un campo que contenga una etiqueta y un cuadro de texto, y que además mostrará un mensaje en caso de darse algún error en el proceso de validación, estos tres elementos se definen a través de los HTML *Helpers* correspondientes. Finalmente se implementa un botón que permite guardar la información ingresada en el servidor y un enlace que permita volver a la vista `Index`. Esta estructura es generada automáticamente gracias al uso de *Scaffolding*.

En el Código 3.8 se muestra un segmento de código de la vista `Create`, en donde se observan ciertas modificaciones a las propiedades de los campos del formulario.

Se observa por ejemplo para el primer campo el uso del atributo HTML `type` para permitir exclusivamente información numérica dentro del cuadro de texto (línea 19). Más adelante para los campos Nombres y Apellidos se especifica que la información ingresada en el cuadro de texto sea transformada a mayúsculas a través del atributo HTML `style` (línea 35 y 43), por último se presenta la lista desplegable a través del *helper* `Html.DropDownListFor` (línea 51), el cual carga la información definida en la propiedad `ViewBag.Carrera` del método `Create` (Código 3.7). Luego de ingresar la información necesaria en los campos

del formulario, se procede a enviar estos datos al servidor para poder almacenar dicho registro en la tabla correspondiente de la base de datos una vez que se da clic en el botón “Crear” (línea 74).

```
16      <div class="form-group">
17          @Html.LabelFor(model => model.EstudianteID, htmlAttributes: new { @class = "control-label col-md-2" })
18          <div class="col-md-10">
19              @Html.EditorFor(model => model.EstudianteID, new { htmlAttributes = new { @class = "form-control", @type = "number" } })
20              @Html.ValidationMessageFor(model => model.EstudianteID, "", new { @class = "text-danger" })
21          </div>
22      </div>
(...)
32      <div class="form-group">
33          @Html.LabelFor(model => model.Nombres, htmlAttributes: new { @class = "control-label col-md-2" })
34          <div class="col-md-10">
35              @Html.EditorFor(model => model.Nombres, new { htmlAttributes = new { @class = "form-control", style = "text-transform:uppercase" } })
36              @Html.ValidationMessageFor(model => model.Nombres, "", new { @class = "text-danger" })
37          </div>
38      </div>
39
40      <div class="form-group">
41          @Html.LabelFor(model => model.Apellidos, htmlAttributes: new { @class = "control-label col-md-2" })
42          <div class="col-md-10">
43              @Html.EditorFor(model => model.Apellidos, new { htmlAttributes = new { @class = "form-control", style = "text-transform:uppercase" } })
44              @Html.ValidationMessageFor(model => model.Apellidos, "", new { @class = "text-danger" })
45          </div>
46      </div>
47
48      <div class="form-group">
49          @Html.LabelFor(model => model.CarreraEstudianteID,"Carrera", htmlAttributes: new { @class = "control-label col-md-2" })
50          <div class="col-md-10">
51          @Html.DropDownListFor(model => model.CarreraEstudianteID, (SelectList)ViewBag.Carrera, htmlAttributes: new { @class = "form-control" })
52          @Html.ValidationMessageFor(model => model.CarreraEstudianteID, "", new { @class = "text-danger" })
(...)
72      <div class="form-group">
73          <div class="col-md-offset-2 col-md-10">
74              <input type="submit" value="Crear" class="btn btn-default" />
75          </div>
76      </div>
77  </div>
```

Código 3.8. Vista `Create` para un nuevo registro de estudiante

En el Código 3.9 se presenta el método `Create` empleado en una solicitud HTTP POST, este método permitirá guardar la información ingresada por el usuario en el formulario y regresar a la vista `Index`, en la cual se muestran los registros de estudiantes existentes.

Inicialmente se observa el uso del atributo `HttpPost` (línea 114), el mismo que asocia a las solicitudes HTTP POST con este método `Create`, además se usa el atributo `ValidateAntiForgeryToken` (línea 115) para evitar el ataque `Cross-`

`Site Request Forgery` al enviar el formulario al servidor. En las líneas 116 y 117 se define el método `Create` teniendo como argumento a un objeto de tipo `Estudiante` y especificando las propiedades que se manejarán desde la vista a través de la propiedad `Include` del atributo `Bind`.

Antes de guardar los datos presentes en el formulario se procede a verificar que esta información sea válida (línea 121); de ser este el caso se procede a guardar la información y se ejecutan acciones adicionales como las de asignar un estado activo al estudiante (línea 123) y guardar los nombres y apellidos en mayúsculas (líneas 124 y 125). Finalmente se agrega el registro a la tabla `Estudiantes` (líneas 126 y 127) y se procede a llamar al método de acción `Index`, el cual posteriormente cargará la vista `Index` (línea 128).

Si existe algún error al guardar la información se cargará la página nuevamente a través del método `View` (línea 140) mostrando un mensaje de error definido en la propiedad `ViewBag.ErrorMesage` (líneas 133 y 134), para esto es necesario cargar nuevamente la propiedad `ViewBag.Carrera` con la información necesaria (línea 136 a la 139).

```
113         // POST: Estudiantes/Create
114         [HttpPost]
115         [ValidateAntiForgeryToken]
116         public ActionResult Create([Bind(Include = @"EstudianteID,NumeroUnico,Nombres,Apellidos,
117             Telefono,CorreoElectronico,CarreraEstudianteID")] Estudiante estudiante)
118         {
119             try
120             {
121                 if (ModelState.IsValid)
122                 {
123                     estudiante.EstadoEstudiante = true;
124                     estudiante.Nombres = estudiante.Nombres.ToUpper();
125                     estudiante.Apellidos = estudiante.Apellidos.ToUpper();
126                     db.Estudiantes.Add(estudiante);
127                     db.SaveChanges();
128                     return RedirectToAction("Index");
129                 }
130             }
131             catch (DataException)
132             {
133                 ViewBag.ErrorMessage = @"La información no ha podido ser guardada, por favor inténtelo nuevamente.
134                         Si el error persiste contacte al administrador del sistema";
135             }
136             var carreras = from c in db.CarreraEstudiantes
137                            select c;
138             carreras = carreras.Where(c => c.EstadoCarrera.Equals(true));
139             ViewBag.Carrera = new SelectList(carreras, "ID", "Carrera");
140             return View();
141         }
```

Código 3.9. Método `Create` de `EstudiantesController` (Solicitud HTTP POST)

3.2.2.2 Asignación de un estudiante al formulario de creación de un plan

Para la creación de un plan es necesario añadir información de estudiantes y docentes en el formulario correspondiente. Para la elección de los docentes que ocuparán los cargos de Director y Codirector se implementarán listas desplegables que permitan la inclusión de esta información, pero para definir los estudiantes que realizarán el plan se hacen uso de otras herramientas que permitan buscar y elegir al estudiante.

A continuación se describe el uso de una ventana modal, una vista parcial, un *script* de JavaScript y la técnica AJAX; herramientas que permitirán asignar la información de un estudiante en el formulario correspondiente.

3.2.2.2.1 Ventana Modal

Una ventana modal es una ventana que se despliega sobre la vista principal y permite ejecutar acciones adicionales, evitando de esta manera salir de la vista con la que interactúa el usuario. Para el presente caso la ventana modal se mostrará cuando el usuario presione el botón que se encontrará junto al campo que presentará la información del estudiante, este botón tendrá un ícono de una lupa, tal como se observa en la Figura 3.11.

Figura 3.11. Campo Estudiante del formulario de creación de un plan

En el Código 3.10 se muestra el código del botón que permitirá cargar la ventana modal. A través de los atributos `data-toggle` y `data-target` se especifica el tipo y el identificador del elemento que se desplegará (línea 49), también se muestra el uso de un ícono de lupa en el botón (línea 50).

```
49    <button type="button" class="btn btn-default btn-s col-md-offset-4" data-toggle="modal" data-target="#myModalEstudiante">
50        <span class="glyphicon glyphicon-search" aria-hidden="true"></span>
51    </button>
```

Código 3.10. Botón que permite cargar la ventana modal

En el Código 3.11 se observa la estructura del código de la ventana modal denominada `myModalEstudiante`, definida en el atributo HTML `id` (línea 145). Todos los estilos definidos a través de los atributos HTML `class` dentro de los contenedores `<div>` forman parte del *framework* Bootstrap y permiten implementar la ventana modal. Primero se define una cabecera (línea 148 a la 151) en la que se define un botón para cerrar la ventana (línea 149) y un título (línea 150), más adelante se define un contenedor para el cuerpo de la ventana (línea 152 a la 190) y dentro de este contenedor se define un pie de página en el que se implementa un botón para volver a la vista principal (línea 188 a la 190).

```
145  <div class="modal fade myModalEstudiante" id="myModalEstudiante" role="dialog">
146      <div class="modal-dialog">
147          <div class="modal-content">
148              <div class="modal-header">
149                  <button type="button" class="close" data-dismiss="modal">&times;</button>
150                  <h4 class="modal-title">ESTUDIANTE</h4>
151              </div>
152              <div class="modal-body">
(...)
188                  <div class="modal-footer">
189                      <button type="button" class="btn btn-default" data-dismiss="modal">Volver</button>
190                  </div>
191              </div>
192          </div>
193      </div>
194  </div>
```

Código 3.11. Ventana Modal `myModalEstudiante`

En el Código 3.12 se muestra el código definido para el cuerpo de la ventana modal, el mismo que permitirá buscar registros de estudiantes. La búsqueda de los registros se realizará a través de AJAX de tal forma que no se necesite recargar la página para ejecutar la acción solicitada. En un inicio se observa la declaración de un formulario que utilizará AJAX a través del método de acción `_BuscarYAgregar` y se configuran las opciones que permitirán ejecutar la operación AJAX (línea 154 a la 159), estas opciones se configuran a través de propiedades como: `HttpMethod`, que define el uso del método HTTP GET para realizar la petición (línea 156), `UpdateTargetId`, que especifica el elemento en el cual se cargará la información solicitada (línea 157) e `InsertionMode`, que define la forma de inserción de los datos en el elemento, para este caso se especifica el modo de reemplazo (línea 158).

Posteriormente se muestra un cuadro de texto que permitirá ingresar el apellido del estudiante a buscar (línea 163) una vez que se presione el botón “Buscar” (línea 164). Más adelante se implementa una tabla (línea 167 a 187), la cual cuenta en su primera fila con una cabecera con los campos Cédula, Apellidos, Nombres y Carrera (línea 168 a la 181), esta cabecera se mantendrá estática en la ventana modal, sin embargo, el resultado de la búsqueda ira variando de acuerdo a la información ingresada en el cuadro de texto.

El resultado de la búsqueda se presenta haciendo uso de la etiqueta HTML `<tbody>` (líneas 183, 184 y 185), se observa que este elemento HTML cuenta con el identificador definido en la propiedad `UpdateTargetId` (línea 157). Dentro de este elemento se procede a llamar al método de acción `_BuscarYAgregar` (línea 184), con el parámetro de entrada correspondiente a la información ingresada en el cuadro de texto (línea 163), este método permitirá realizar la búsqueda solicitada y devolver dicha información en la vista parcial que se describe a continuación.

```
152             <div class="modal-body">
153
154                 @using (Ajax.BeginForm("_BuscarYAgregar", new AjaxOptions
155                 {
156                     HttpMethod = "GET",
157                     UpdateTargetId = "searchResultEstudiante",
158                     InsertionMode = InsertionMode.Replace,
159                 }))
160
161                 {
162                     <p>
163                         @Html.TextBox("apellidos")
164                         <button type="submit">Buscar</button>
165                     </p>
166                 }
167                 <table class="table">
168                     <tr>
169                         <th>
170                             Cédula
171                         </th>
172                         <th>
173                             Apellidos
174                         </th>
175                         <th>
176                             Nombres
177                         </th>
178                         <th>
179                             Carrera
180                         </th>
181                     </tr>
182
183                     <tbody id="searchResultEstudiante">
184                         @Html.Action("_BuscarYAgregar", new { apellidos = ""})
185                     </tbody>
186
187                 </table>
```

Código 3.12. Ventana Modal `myModalEstudiante` (continuación)

3.2.2.2.2 Vista Parcial

Se define como vista parcial a la vista que permite cargar información dentro de otra vista. En el Código 3.13 se muestra el método `_BuscarYAgregar` que permitirá realizar la búsqueda solicitada y cargar el resultado de la búsqueda en una vista parcial.

Se observa que este método es de tipo `PartialViewResult`, lo cual indica que la información solicitada se cargará a través de una vista parcial, este método recibe como parámetro una cadena de caracteres que contiene la información a buscar (línea 137). Posteriormente luego de validar que la cadena contiene información (línea 141), se realiza la búsqueda de los estudiantes activos que contengan en su apellido la cadena pasada desde la vista (líneas 143 a 146). Para manejar esta información se define un objeto de tipo `VMPlanTT`, `VMPlanTT` es un ViewModel que permite a través de su propiedad `VEstudiantes` almacenar registros de tipo `Estudiante` que coincidan con la búsqueda (líneas 148 y 149), cada uno de los registros será agregado a una lista de tipo `VMPlanTT` (línea 150), y esta lista será presentada en la vista parcial `_BuscarYAgregar` (línea 152).

```
137         public PartialViewResult _BuscarYAgregar(string apellidos)
138         {
139             List<VMPlanTT> listaEstudiantes = new List<VMPlanTT>();
140
141             if (!String.IsNullOrEmpty(apellidos))
142             {
143                 var estudiantes = from e in db.Estudiantes.ToList()
144                                   select e;
145                 estudiantes = estudiantes.Where(e => e.Apellidos.Contains(apellidos.ToUpper())
146                 &e.EstadoEstudiante.Equals(true));
147
148                 VMPlanTT vmPlanTT = new VMPlanTT();
149                 vmPlanTT.VEstudiantes = estudiantes.ToList();
150                 listaEstudiantes.Add(vmPlanTT);
151             }
152             return PartialView(listaEstudiantes);
153         }
```

Código 3.13. Método `_BuscaryAgregar`

En el Código 3.14 se muestra el código de la vista parcial `_BuscaryAgregar`. Esta vista permite presentar información basada en una lista de modelos

`VMPlanTT` (línea 1) en vez de un modelo, ya que como se verá más adelante es necesario realizar iteraciones para mostrar los resultados de la búsqueda, y estas iteraciones solo se pueden realizar a través del uso de listas.

En la primera iteración se manejan los registros que coinciden con los criterios de la búsqueda, mientras que en la segunda iteración se carga, para cada registro, las propiedades correspondientes a la cédula, apellidos, nombres y carrera (líneas 9, 12, 15, 18) en una nueva fila de la tabla que se presentará en la ventana modal.

Esta nueva fila creada define una celda adicional en donde se muestra un enlace con el nombre “Agregar” (línea 21), al hacer clic en este enlace se ejecutará otra acción que permitirá, a través de AJAX, enviar la información del estudiante elegido a la vista `Create` del controlador `PlanTTsController` , esto se realiza gracias al uso de la propiedad `OnComplete` (línea 23), la cual permite definir la función de JavaScript `agrega`, esta función permite obtener el nombre completo y la cédula del estudiante elegido antes de cargar la vista `Create`.

```
1   @model List<AAPTT.Models.VMPlanTT>
2
3           @foreach (var item in Model)
4           {
5              foreach (var estudiante in item.VEstudiantes)
6              {
7               <tr>
8                   <td>
9                       @Html.DisplayFor(modelItem => estudiante.EstudianteID)
10                  </td>
11                  <td>
12                      @Html.DisplayFor(modelItem => estudiante.Apellidos)
13                  </td>
14                  <td>
15                      @Html.DisplayFor(modelItem => estudiante.Nombres)
16                  </td>
17                  <td>
18                      @Html.DisplayFor(modelItem => estudiante.CarreraEstudiante.Carrera)
19                  </td>
20                  <td>
21                      @Ajax.ActionLink("Agregar", "Create", "PlanTTs", new { id = estudiante.EstudianteID }, new AjaxOptions
22                  {
23                     OnComplete = "agrega('"+estudiante.EstudianteID+"','"+estudiante.NombreCompleto+"')"
24                  })
25                  </td>
26              </tr>
27              }
28          }
```

Código 3.14. Vista parcial `_BuscarYAgregar`

3.2.2.2.3 JavaScript

En el Código 3.15 se muestra la función `agrega`, implementada en JavaScript. Esta función recibe como parámetros la cédula y el nombre completo del estudiante elegido previamente (línea 364) y asignará al elemento HTML con identificador `txtEstID1` el valor de la cédula, y al elemento HTML con identificador `txtEst1` el nombre completo del estudiante (líneas 365 y 366), respectivamente; por último la función cerrará la ventana modal utilizada para esta operación (línea 367).

```
362  <script type="text/javascript" >
363
364      function agrega(id, nombre) {
365          document.getElementById('txtEstID1').value = id;
366          document.getElementById("txtEst1").value = nombre;
367          $('.myModalEstudiante.in').modal('hide');
368      }
```

Código 3.15. Función `agrega` de JavaScript

Finalmente en el Código 3.16 se observan los elementos del campo correspondiente a un estudiante, este campo forma parte del formulario de creación de un plan definido en la vista `Create`.

Se puede observar la asignación de `txtEst1` en el atributo HTML `id` del primer cuadro de texto (línea 46) y la asignación de `txtEstID1` en el atributo HTML `id` del segundo cuadro de texto (línea 47). Estos identificadores permitirán cargar la información previamente obtenida mediante la función `agrega`. También se observa el botón que permitirá cargar la ventana modal denominada `myModalEstudiante` (líneas 49, 50 y 51).

```
43      <div class="form-group">
44          @Html.Label( "Estudiante", htmlAttributes: new { @class = "control-label col-md-2" })
45          <div class="col-md-10">
46              @Html.Editor("Estudiante", new { htmlAttributes = new { @class = "form-control", @id = "txtEst1", @readonly = "readonly" } })
47              @Html.TextBoxFor(model => model.VEstudiante1ID, new { @class = "form-control", @id = "txtEstID1", @readonly = "readonly" })
48              @Html.ValidationMessageFor(model => model.VEstudiante1ID, "", new { @class = "text-danger" })
49              <button type="button" class="btn btn-default btn-s col-md-offset-4" data-toggle="modal" data-target="#myModalEstudiante">
50                  <span class="glyphicon glyphicon-search" aria-hidden="true"></span>
51              </button>
```

Código 3.16. Campo estudiante Vista `Create`

3.2.3 MÓDULO 3: ENVÍO DE CORREOS ELECTRÓNICOS

Este módulo permite el envío y la personalización de correos electrónicos para las situaciones definidas en las historias de usuario correspondientes. Estas situaciones se resumen en dos instancias:

- Ingreso de un plan al sistema.
- Evaluación de un plan por parte de la Comisión.

Para la primera instancia se definen dos tipos de correos electrónicos, el primero es un correo que será enviado como notificación del ingreso del plan al sistema a los estudiantes y docentes correspondientes. Mientras que el segundo correo será enviado a los miembros de la Comisión, este correo tendrá como archivo adjunto el plan de Trabajo de Titulación escaneado. Estos dos tipos de correos electrónicos han podido ser enviados gracias al uso de los métodos `EmailIngreso` y `EmailIngresoMiembro`, métodos que forman parte del controlador `PlanTTsController`. Estos métodos se ejecutarán mediante la interacción del usuario a través de enlaces definidos para cada plan en la vista `Index` del controlador `PlanTTsController`.

Para la segunda instancia se manejan tres tipos de correos electrónicos, uno para notificar las observaciones hechas al plan, otro para notificar que un plan ha sido negado y otro para notificar que un plan ha sido aprobado. Estos tres tipos de correos electrónicos han podido ser enviados gracias al uso del método `EmailEvaluacion`, método que forma parte del controlador `RubricaEvaluacionsController`. Este método se ejecutará a través de un enlace definido para cada plan evaluado en la vista `Index` del controlador `RubricaEvaluacionsController`.

3.2.3.1 Uso de Postal para el Envío de Correos Electrónicos

Luego de analizar las posibles opciones que se tenía para implementar el envío de correos electrónicos se decidió que el uso del paquete Postal [20] era el más idóneo. Este paquete ha sido creado específicamente para ser usado por el *framework* ASP.NET MVC al permitir una interacción directa a través de sus vistas. `Postal` permite el envío de un correo electrónico a través de un método de acción y una vista que defina la información a enviar, dentro del método se

define información propia del correo como: dirección de remitente, direcciones de destinatarios, asunto y cuerpo del correo electrónico, para después enviar dicho correo a través de un servicio de correo especificado en el archivo de configuración `Web.config`. Al igual que en el primer módulo, se utiliza la plataforma SendGrid para el envío de correos electrónicos. A continuación se describe el proceso seguido para el envío de correos electrónicos a través del método `EmailIngreso`.

En el Código 3.17 se muestra el uso del paquete `Postal` en el método `EmailIngreso`. Este método recibe como parámetro el identificador de un plan ingresado, con este identificador y a través de consultas LINQ se obtienen las direcciones de correos electrónicos de los estudiantes y docentes asociados a dicho plan.

Para cargar el texto que irá en el cuerpo del correo electrónico, se carga el registro correspondiente de la tabla `EmailCuerpos` en el objeto `emailObj` (línea 358), para luego en el objeto `cuerpo` cargar la información correspondiente al cuerpo del correo (línea 359).

Más adelante se define un objeto denominado `email` (línea 362) que es una instancia de la clase `Email` de Postal, la información almacenada por este objeto definirá el contenido del correo a través de la vista `EmailIngreso` (Código 3.18). Para esto se hace uso de las propiedades del objeto `email`, permitiendo asignar las direcciones de destino (línea 363), la dirección del remitente (línea 364), el asunto del correo (línea 365), y un título (línea 366) y el contenido para el cuerpo del correo electrónico (línea 367). Finalmente se envía el correo electrónico a través del método `Send` (línea 368).

Por último se definen dos propiedades `ViewBag` que cargarán un título y un mensaje de notificación en la vista que notifica al usuario que el correo electrónico ha sido enviado (líneas 371 y 372), la información de estas propiedades, así como la de propiedades definidas en las líneas 364, 365 y 366 se definen a través de una clase llamada `Textos`, clase que fue creada para manejar los textos asignados para distintas situaciones a lo largo de la aplicación. Por último se carga la vista de notificación a través del método `View` (línea 373).

```
267         [Authorize]
268         public ActionResult EmailIngreso(int id)
269         {
(...)

357             //Mensaje Email
358             var emailObj = db.EmailCuerpos.Find(IDCuerpoEmail.idIngreso);
359             var cuerpo = emailObj.CuerpoEmail;
360
361             //Envio de Correo Electronico
362             dynamic email = new Postal.Email("EmailIngreso");
363             email.To = destinatariosDoc + "," + destinariosEst;
364             email.From = Textos.direccionRemitente;
365             email.Subject = Textos.asuntoIngreso;
366             email.TituloCuerpo = Textos.tituloCuerpoIngreso;
367             email.Message = cuerpo;
368             email.Send();
369
370             //Vista notificación
371             ViewBag.TituloNotificacion = Textos.notificacionTitulo;
372             ViewBag.MensajeNotificacion = Textos.notificacionMensaje;
373             return View();
374         }
```

Código 3.17. Método `EmailIngreso`

En el Código 3.18 se presenta la vista `EmailIngreso`, la cual define la información del correo electrónico que será enviado. En esta vista se define el tipo de contenido que incluirá el correo electrónico y la codificación de caracteres empleada (línea1), un título para el cuerpo del correo electrónico (línea 3) y posteriormente el cuerpo como tal del correo (línea 5). Para ambos casos `ViewBag` permitirá cargar la información correspondiente a cada elemento a través de las propiedades definidas en el objeto `email` del método `EmailIngreso`.

```
1   Content-Type: text/html; charset=utf-8
2
3   <h2>@ViewBag.TituloCuerpo</h2>
4
5   @ViewBag.Message
```

Código 3.18. Vista `EmailIngreso`

En el Código 3.19 se presenta la vista que se cargará una vez ejecutado el método `EmailIngreso`, la misma que notificará al usuario del envío del correo

electrónico. En esta vista se define inicialmente el título del documento HTML (línea 2) y se muestran el título y el mensaje de notificación asignados previamente (líneas 5 y 8). Por último, a través de un pie de página se implementa un enlace que permite que se llame al método `Index` para mostrar la vista donde se encuentran todos los planes del sistema.

```
1   @{
2       ViewBag.Title = "Notificación de Envío de Correo Electrónico";
3   }
4
5   <h4> @ViewBag.TituloNotificacion</h4>
6
7   <p>
8       @ViewBag.MensajeNotificacion
9   </p>
10
11  <footer>
12      @Html.ActionLink("Volver", "Index", "PlanTTs")
13  </footer>
```

Código 3.19. Vista de Notificación de Envío de Correo Electrónico

3.2.3.2 Personalización del Cuerpo de un Correo Electrónico

A través de la interacción con el modelo `EmailCuerpo` el administrador del sistema puede personalizar la información a enviar en el cuerpo del correo electrónico para cada una de las distintas situaciones ya mencionadas. El controlador y las vistas correspondientes a este modelo fueron generados automáticamente vía *Scaffolding,* siguiendo el mismo proceso del módulo anterior correspondiente a la Gestión de Información.

En el Código 3.20 se muestra la clase que permite implementar el modelo `EmailCuerpo`.

Este modelo cuenta con tres propiedades denominadas `ID` (línea 10), `TipoEmail` (línea 14) y `CuerpoEmail` (línea 19). Para la propiedad `ID` se ha desactivado la propiedad SQL Identity a través del atributo `DatabaseGenerated` y de la enumeración `DatabaseGeneratedOption` (línea 9), ya que estos valores han sido asignados para cada situación a través del método `Seed` (Código 3.21), el cual permite cargar información predefinida en la aplicación.

```
7   public class EmailCuerpo
8   {
9       [DatabaseGenerated(DatabaseGeneratedOption.None)]
10      public int ID { get; set; }
11
12      [Required]
13      [Display(Name = "Tipo de Correo Electrónico")]
14      public string TipoEmail { get; set; }
15
16      [Required]
17      [DataType(DataType.MultilineText)]
18      [Display(Name = "Cuerpo del Correo Electrónico")]
19      public string CuerpoEmail { get; set; }
20
21  }
```

Código 3.20. Modelo `EmailCuerpo`

En el Código 3.21 se muestra un segmento de código del método `Seed` en el cual se observa la asignación de los identificadores para los dos primeros tipos de correos electrónicos.

Para el primer registro se define el valor del identificador establecido en la enumeración `IDCuerpoEmail` (línea 354), este identificador se relacionará con el tipo de correo correspondiente al de ingreso de plan en el sistema (línea 355). Para el segundo registro se hace lo propio según el identificador asignado (línea 362), el cual ha sido establecido para los correos que notifiquen el ingreso del plan al sistema a los miembros de la comisión (línea 363).

Esta asignación se la realiza también para los tres tipos de correos electrónicos definidos en la instancia de evaluación, es decir, para los correos que permitan notificar observaciones hechas al plan, negación del plan y aprobación del plan.

```
350 var cuerpoEmails = new List<EmailCuerpo>
351 {
352     new EmailCuerpo
353     {
354         ID = (int)IDCuerpoEmail.idIngreso,
355         TipoEmail="Ingreso Plan",
356         CuerpoEmail=@"Su plan será tratado en la siguiente reunión de la Comisión y
357         se notificará vía correo electrónico las observaciones al mismo."
358     },
359
360      new EmailCuerpo
361     {
362         ID = (int)IDCuerpoEmail.idIngresoMiembro,
363         TipoEmail="Ingreso Plan Miembros",
364         CuerpoEmail=@"Por favor revisar el contenido del plan que se adjunta en PDF para
365         poder tratar su aprobación en la siguiente reunión de la Comisión."
366     },
```

Código 3.21. Registros del modelo `EmailCuerpo` definidos en el método `Seed`

3.2.4 MÓDULO 4: GESTIÓN DE RÚBRICAS

Este módulo permite implementar la creación y el uso de rúbricas para evaluar un determinado plan de Trabajo de Titulación, además en este módulo se implementó la generación de un archivo que contenga la rúbrica usada en la evaluación, este archivo notificará a los estudiantes y docentes sobre los criterios de evaluación que no han obtenido el puntaje adecuado, para que de esta manera se puedan realizar las modificaciones necesarias para la posterior aprobación del plan.

Tanto la creación de una rúbrica, como la presentación de una rúbrica ya creada requirieron el uso de funciones JavaScript. A través de estas funciones se generaron dinámicamente los criterios de dichas rúbricas en las vistas correspondientes.

Una vez realizada la evaluación del plan, se procede a enviar un correo electrónico notificando al usuario del estado de su plan y las modificaciones que se deben realizar a dicho plan por medio de un archivo adjunto. La generación de este archivo fue realizada a través del paquete iTextSharp [21], el mismo que permitió la generación de la rúbrica a través de un método de acción.

iTextSharp es una de las soluciones más usadas por desarrolladores para la generación de archivos PDF, la ventaja de este paquete es que permite la creación de documentos a partir de código; existen otros paquetes que permiten generar el archivo PDF a partir de la información mostrada en una determinada vista, sin embargo, estos paquetes no representan fielmente la información mostrada y sobretodo no permiten la manipulación de la información, como si lo hace iTextSharp.

A continuación se muestra el uso de JavaScript para la asignación de criterios en la creación de una rúbrica y el uso de iTextSharp para la generación del archivo PDF.

3.2.4.1 Asignación de Criterios para una Nueva Rúbrica

El administrador del sistema será el encargado de crear rúbricas dentro de la aplicación, una vez que la rúbrica esté creada se la podrá utilizar para la evaluación de los planes presentados.

El proceso de creación de rúbricas se la hace en dos pasos: primero se define el nombre de la rúbrica y los porcentajes máximos para cada estado, considerando que los estados pueden ser: Negado, Pendiente por Aprobar o Aprobado. Este proceso se lo realiza siguiendo el método *Code First* de Entity Framework a través de la creación del modelo `Rubrica` y la ejecución de los métodos `Create` para las solicitudes HTTP GET y HTTP POST, estas solicitudes permitirán la creación del registro correspondiente. Posteriormente luego de asignar un nombre a la rúbrica y definir los porcentajes correspondientes, como segundo paso se procede a cargar una nueva vista denominada `Criterios`. A través de esta vista el usuario podrá ir agregando los criterios que considere necesarios dentro de la rúbrica, asignando a cada criterio un número, una descripción y un valor máximo.

En el Código 3.22 se muestra un segmento de código de la vista `Criterios`, en esta vista se crea una tabla que permita ir generando la rúbrica a través de la agregación de criterios. Esta tabla se la define a través de la etiqueta `<table>`, la cual especifica un identificador que permitirá manipular este elemento HTML para la agregación o eliminación de criterios (línea 33).

```
29  <body>
30      <h2>Crear Rúbrica</h2>
31
32      <div id="tablas">
33          <table id='tabla-informacion'>
34              <thead>
35                  <tr>
36                      <td colspan="2"><button onclick="agregarCampos()">Agregar Criterios</button>
37                      <button onclick="quitarCampos()">Quitar Criterios</button></td>
38                  </tr>
39                  <tr>
40                      <th>#</th>
41                      <th>Criterio de evaluación</th>
42                      <th>Valoración Máxima</th>
43                  </tr>
44              </thead>
45          </table>
46      </div>
47
48          <div class="form-group">
49              <div class="col-md-offset-2 col-md-10">
50                  <input type="submit" value="Crear Rúbrica" class="btn btn-default" onclick="enviarDatosViaAjax()" id="guardar" />
51              </div>
52          </div>
```

Código 3.22. Vista `Criterios`

Para la primera fila de la tabla se implementan dos botones que permiten agregar criterios y quitar criterios a través de las funciones `agregarCampos` y

quitarCampos, implementadas mediante JavaScript (líneas 36 y 37), y en la siguiente fila se establece una cabecera con el nombre de los tres campos a ser llenados: Número, Criterio de Evaluación y Valoración Máxima (líneas 40, 41 y 42). Por último y fuera de la tabla, se implementa un botón que permitirá ejecutar la función enviarDatosViaAjax para realizar el envío de la información generada al servidor (línea 50).

Mediante JavaScript se permite agregar, eliminar y enviar los criterios definidos en la rúbrica al servidor. En el Código 3.23 se muestra la función agregarCampos, esta función permite ir agregando una fila a la tabla, cada vez que se presione el botón correspondiente.

Inicialmente se define un objeto denominado tabla, que permitirá interactuar con la tabla definida en la vista a través del método getElementByID (línea 61). Luego se definen objetos que permiten crear elementos HTML, elementos correspondientes a una fila (línea 63), tres celdas (líneas 64, 65 y 66), y tres cuadros de texto (líneas 67, 68 y 69); los cuadros de textos permitirán el ingreso de la información correspondiente para el número del criterio, la descripción del criterio y la valoración máxima que puede tener el criterio.

Posteriormente se observa el uso del atributo HTML class para el objeto fila (línea 71), el cual servirá para identificar cada fila implementada, y se definen atributos para cada cuadro de texto.

Para el objeto numero se define un tipo de ingreso numérico (línea 74), un ancho de 40 píxeles (línea 75), un valor mínimo de 1 (línea 76) y un marcador de posición que representa la información a ingresar (línea 77). Este proceso se repite para los demás cuadros de texto por lo que se lo ha omitido.

Por último se observa la adición de los cuadros de texto en las celdas (línea 94, 95y 96), la adición de las celdas en la fila (líneas 98, 99 y 100) y la adición de la fila a la tabla (línea 102) a través del método appendChild.

En el Código 3.24 se muestra la función quitarCampos, mediante esta función se elimina el último criterio creado en la rúbrica, para lo cual primero se obtiene la última fila generada en la tabla a través del método getElementByClassName

(línea 111) y luego de garantizarse que esta fila exista (línea 113), se procede a eliminarla usando el método `removeChild` (línea 115).

```
59  function agregarCampos() {
60
61      const tabla = document.getElementById('tabla-informacion');
62
63      const fila = document.createElement('tr');
64      const celda1 = document.createElement('td');
65      const celda2 = document.createElement('td');
66      const celda3 = document.createElement('td');
67      const numero = document.createElement('input');
68      const criterio = document.createElement('textarea');
69      const valoracionMax = document.createElement('input');
70
71      fila.setAttribute('class', 'criterio-' + contador);
72
73      //Atributos de numero
74      numero.setAttribute('type', 'number');
75      numero.setAttribute('style', 'width: 40px');
76      numero.setAttribute('min', '1');
77      numero.setAttribute('placeholder', '#');
(...)
94      celda1.appendChild(numero);
95      celda2.appendChild(criterio);
96      celda3.appendChild(valoracionMax);
97
98      fila.appendChild(celda1);
99      fila.appendChild(celda2);
100     fila.appendChild(celda3);
101
102     tabla.appendChild(fila);
103
104     contador++;
105 }
```

Código 3.23. Función `agregarCampos`

```
108     function quitarCampos() {
109
110         --contador;
111         const fila = document.getElementsByClassName('criterio-' + contador);
112
113         if (fila.length) {
114             const ultimaFila = fila[0];
115             ultimaFila.parentNode.removeChild(ultimaFila);
116         }
117     }
```

Código 3.24. Función `quitarCampos`

En el Código 3.25 se presenta la función `enviarDatosViaAjax`, esta función permite el envío de los criterios generados a través de AJAX.

Primero se verifica que no existan campos sin llenar en los cuadros de texto (línea 140) y mediante una llamada a la función `capturarDatos` (línea 141) se obtiene toda la información de los criterios.

Posteriormente se ejecuta la petición AJAX a través del método `ajax`, método que forma parte de la librería jQuery de JavaScript. Este método define las características de la petición a través de varias propiedades, entre estas propiedades están: la URL que define el método de acción que recibirá la petición (línea 143), el método HTTP que se usará para realizar la petición (línea 144), el formato de los datos que se enviarán, (línea 145), el tipo de contenido que se envía (línea 147) y el tipo de datos que se espera recibir como respuesta (línea 147).

Finalmente se definen dos funciones: una función que se ejecutará en caso de que exista un error al enviar la información (líneas 148, 149 y 150), esta función presentará un mensaje que alerte al usuario que ha existido algún error; y, una función que se ejecutará en caso de que el envío sea exitoso (línea 151, 152, 153), esta función cargará la URL asignada en el método de acción que recibe esta petición.

```
139  function enviarDatosViaAjax() {
140      if (verificarCamposVacios()) {
141          const datos = capturarDatos();
142          $.ajax({
143              url: "@Url.Action("Criterios")",
144              type: "POST",
145              data: JSON.stringify(datos),
146              contentType: "application/json; charset=utf-8",
147              dataType: "json",
148              error: function (response) {
149                  alert(response.responseText);
150              },
151              success: function (json) {
152                  if (json.isRedirect) {
153                      window.location.href = json.redirectUrl;
154                  }
155              }
156          });
157      }
158  }
```

Código 3.25. Función `enviarDatosViaAjax`

En el Código 3.26 se muestra la función `capturarDatos`, esta función permite almacenar la información de los criterios definiendo un arreglo de objetos JSON. Mediante una primera iteración se obtiene la información de las filas de la tabla (línea 123 a la 132) y a través de una segunda iteración se obtiene la información de las celdas de cada fila creada (líneas 125 a 131). La información de un criterio se almacena en el objeto `camposObj`, el cual obtiene los datos de los cuadros de texto a través de sus propiedades `NumeroCriterio`, `Descripcion` y `ValoracionMaxima` (líneas 126, 127 y 128), el objeto creado en cada iteración se añadirá al arreglo `camposArr` a través del método `push` (línea 129). Finalmente se retorna el arreglo `camposArr` en el cual se han almacenado todos los objetos JSON que corresponden a los criterios definidos, esta información será recibida en el servidor por el método de acción `Criterios` que maneja la solicitud HTTP POST.

```
118  function capturarDatos() {
119          let campos = {};
120          let camposObj = {};
121          let camposArr = [];
122
123          for (let i = 0; i < contador; i++) {
124              campos[i] = document.getElementsByClassName('criterio-' + i);
125              for (let j = 0; j < campos[i].length; j++) {
126                  camposObj.NumeroCriterio = campos[i][j].childNodes[0].childNodes[0].value;
127                  camposObj.Descripcion = campos[i][j].childNodes[1].childNodes[0].value;
128                  camposObj.ValoracionMaxima = campos[i][j].childNodes[2].childNodes[0].value;
129                  camposArr.push(camposObj);
130                  camposObj = {};
131              }
132          }
133          return camposArr;
134  }
```

Código 3.26. Función `capturarDatos`

En el Código 3.27 se observa el método `Criterios` que maneja la solicitud HTTP POST enviada vía AJAX desde el cliente y que permite guardar la información de los criterios en la base de datos una vez que se reciba esta información en el servidor.

Este método recibe como parámetros una lista de objetos de tipo `CriterioNuevo` y un objeto `id`, en donde `CriterioNuevo` es el modelo que

almacenará la información de cada criterio definido en la rúbrica, e `id` corresponde al identificador de la rúbrica (línea 127).

Luego de verificar que exista información dentro de la lista `criterios` (línea 129) se guarda la información de cada criterio recibido a través de una iteración (línea 131 a la 138). En esta iteración se definen las propiedades `RubricaID` y `CriterioNuevoID` (líneas 133 y 134) que son las propiedades del modelo `CriterioNuevo` que no se especificaron desde el cliente, las propiedades `NumeroCriterio`, `Descripcion` y `ValoracionMaxima` son directamente manipuladas a través de la lista `criterios`.

Posteriormente se guarda cada registro en la propiedad `DbSet` correspondiente (línea 135) y en la base de datos (línea 137). Si la acción ha sido exitosa se carga la vista donde se encuentran todas las rúbricas creadas hasta el momento (línea 139 a 143), caso contrario se envía un mensaje de error para alertar al usuario que no se ha podido cumplir con la solicitud encomendada.

```
126     [HttpPost]
127     public ActionResult Criterios(List<CriterioNuevo> criterios, int id)
128     {
129         if (criterios != null)
130         {
131             foreach (CriterioNuevo criterio in criterios)
132             {
133                 criterio.CriterioNuevoID = id.ToString() + "_" + criterio.NumeroCriterio;
134                 criterio.RubricaID = id;
135                 db.CriteriosNuevos.Add(criterio);
136             }
137             db.SaveChanges();
138 
139             return Json(new
140             {
141                 redirectUrl = Url.Action("Index"),
142                 isRedirect = true,
143             });
144         }
145         else
146         {
147             return Json("Ha existido un error, intente de nuevo.");
148         }
149     }
```

Código 3.27. Método `Criterios` (Solicitud HTTP POST)

3.2.4.2 Generación del archivo PDF con la rúbrica de evaluación

Una vez realizada la evaluación del plan de Trabajo de Titulación a través de la rúbrica correspondiente, se debe informar al estudiante el estado de su plan y las

observaciones que deben hacerse para poder aprobar su plan. Esta notificación se la realiza a través de un correo electrónico, tal como se explicó en el módulo anterior, para esto, es necesario primero crear un archivo PDF en donde se incluya la rúbrica utilizada en la evaluación del plan.

Dentro de la vista que define el historial de planes evaluados se tendrá un enlace que permita empezar con este proceso. Al dar clic en el enlace se generará el archivo PDF a través del método `GenerarPDF`, para luego presentar una vista donde se notifique que el archivo ha sido generado, y que muestre un enlace que permita enviar el correo electrónico correspondiente.

El método `GenerarPDF` utiliza el paquete iTextSharp para la generación del archivo PDF, el código de este método se lo describe a continuación.

3.2.4.2.1 Ruta del Archivo PDF

En el Código 3.28 se presenta el primer segmento de código del método `GenerarPDF`, este método recibe como argumento un objeto `id` que corresponde al identificador de la rúbrica de evaluación utilizada, este identificador permitirá obtener la información necesaria para la generación del archivo PDF.

A través del objeto `rubrEval` se obtiene una instancia del modelo `RubricaEvaluacion`, la cual contendrá información del registro correspondiente a la rúbrica de evaluación (línea 129). Luego se establece el nombre del archivo (línea 131) y la ruta que contiene la ubicación y el nombre del archivo (línea 132). Esta ruta será almacenada en la propiedad `PathEvaluacionPDF` del objeto `rubrEval` (línea 133), para poder actualizar el registro definido por `rubrEval` (líneas 134 y 135), de tal forma que cuando se requiera adjuntar el archivo al correo electrónico se localice fácilmente dicho archivo.

y

```
126     [Authorize]
127     public ActionResult GenerarPDF(int? id)
128     {
129         var rubrEval = db.RubricasEvaluadas.Find(id);
130
131         string nombreArchivo = rubrEval.PlanTT.Titulo + "_" +rubrEval.RubricaEvaluacionID.ToString();
132         string fullPath = Server.MapPath(Textos.pathPDFGenerado + nombreArchivo + ".pdf");
133         rubrEval.PathEvaluacionPDF = fullPath;
134         db.Entry(rubrEval).State = EntityState.Modified;
135         db.SaveChanges();
```

Código 3.28. Definición de la ruta donde se guardará el archivo PDF

3.2.4.2.2 Generación de la Rúbrica

En el Código 3.29 se muestra la creación de una tabla que contendrá la información de la rúbrica de evaluación. La creación de esta tabla se logra gracias al uso de las clases `PdfPTable` y `PdfPCell` que forman parte del paquete iTextSharp. Para generar la tabla se define el objeto `table`, el mismo que especifica el número de columnas que tendrá la tabla (línea 138).

Más adelante se definen las celdas que formarán la primera fila, es decir, la cabecera de la tabla; estas celdas serán las correspondientes a los títulos Descripción, Valoración y Observaciones (línea 141 a la 154). Posteriormente se añaden estas celdas a la tabla mediante el método `AddCell` (líneas 156, 157 y 158).

```
137     //GENERACION DE LA RUBRICA
138     PdfPTable table = new PdfPTable(3);
139     var boldFont = FontFactory.GetFont("Arial", 12, Font.BOLD);
140
141     var cellDescr = new PdfPCell();
142     var p1 = new Phrase();
143     p1.Add(new Chunk(Textos.descripcion, boldFont));
144     cellDescr.AddElement(p1);
145
146     var cellVal = new PdfPCell();
147     var p2 = new Phrase();
148     p2.Add(new Chunk(Textos.valoracion, boldFont));
149     cellVal.AddElement(p2);
150
151     var cellObs = new PdfPCell();
152     var p3 = new Phrase();
153     p3.Add(new Chunk(Textos.observaciones, boldFont));
154     cellObs.AddElement(p3);
155
156     table.AddCell(cellDescr);
157     table.AddCell(cellVal);
158     table.AddCell(cellObs);
```

Código 3.29. Creación de la tabla y su cabecera

En el Código 3.30 se presenta la adición de los criterios de la rúbrica en la tabla previamente creada.

A través de una iteración que permita cargar los criterios del registro definido en `rubrEval` (línea 160) se crean tres celdas más por cada fila, es decir, que para cada criterio se obtiene su descripción, su valoración y su observación (líneas 162 a 172), posteriormente se agregan estas tres celdas a la fila correspondiente (líneas 174, 175 y 176). Este proceso se repite por cada criterio existente.

```
160     foreach (var criterioEvaluado in rubrEval.CriterioEvaluado)
161     {
162         var critDescr = new PdfPCell();
163         var c1 = new Phrase(criterioEvaluado.Descripcion);
164         critDescr.AddElement(c1);
165
166         var critVal = new PdfPCell();
167         var c2 = new Phrase(criterioEvaluado.Valoracion.ToString());
168         critVal.AddElement(c2);
169
170         var critObs = new PdfPCell();
171         var c3 = new Phrase(criterioEvaluado.Observacion);
172         critObs.AddElement(c3);
173
174         table.AddCell(critDescr);
175         table.AddCell(critVal);
176         table.AddCell(critObs);
177     }
```

Código 3.30. Adición de los criterios a la tabla

3.2.4.2.3 Generación de Información Adicional

Además de la rúbrica de evaluación, el archivo PDF debe contener ciertos títulos, subtítulos e información adicional correspondiente al plan y a los estudiantes que lo realizan. En el Código 3.31 se muestra la creación de dos títulos que forman parte de esta información adicional. Para generar texto en el archivo se usa la clase `Paragraph` de iTextSharp. Para cada título se definen las propiedades correspondientes a la alineación (líneas 186 y 191), la fuente (líneas 187 y 192) y el contenido del mismo (líneas 188 y 193).

```
184     //INFORMACION ADICIONAL
185     Paragraph titleFac = new Paragraph();
186     titleFac.Alignment = Element.ALIGN_CENTER;
187     titleFac.Font = FontFactory.GetFont("Arial", 14, Font.BOLD);
188     titleFac.Add("\n" + Textos.tituloFacultad + "\n");
189
190     Paragraph titleCarr = new Paragraph();
191     titleCarr.Alignment = Element.ALIGN_CENTER;
192     titleCarr.Font = FontFactory.GetFont("Arial", 12, Font.BOLD);
193     titleCarr.Add(Textos.tituloCarrera + "\n");
```

Código 3.31. Información adicional del documento

3.2.4.2.4 Generación del Documento

Finalmente en el Código 3.32 se observa la generación del documento a través de la incorporación de los elementos descritos anteriormente.

Primero se instancia un objeto de la clase `Document` de iTextSharp (línea 254), este objeto permite guardar el archivo generado en una determinada ruta (líneas

256 y 257), más adelante se añade la información previamente generada a través del método `Add` (línea 259 a la 264), esta información corresponde a títulos, subtítulos, datos del plan y de los estudiantes, y la tabla donde se carga la rúbrica con todos sus criterios.

Finalmente se muestran dos propiedades `ViewBag` que definen textos de notificación (líneas 267 y 268), las cuales se mostrarán en la vista de notificación `GenerarPDF` generada a través del método `View` (línea 270), para lo cual se requiere el uso del modelo `rubrEval`.

```
253         //GENERACIÓN DEL DOCUMENTO
254         var doc = new Document();
255
256         string path = Server.MapPath(Textos.pathPDFGenerado);
257         PdfWriter.GetInstance(doc, new FileStream(path + nombreArchivo + ".pdf", FileMode.Create));
258
259             doc.Open();
260             doc.Add(titleFac);
261             doc.Add(titleCarr);
262             doc.Add(subtitles);
263             doc.Add(titleDatos);
264             doc.Add(table);
265             doc.Close();
266
267         ViewBag.NotificacionTitulo=Textos.notificacionPDFTitulo;
268         ViewBag.NotificacionMensaje = Textos.notificacionPDFMensaje;
269
270         return View("GenerarPDF", rubrEval);
271     }
```

Código 3.32. Generación del archivo PDF

En el Código 3.33 se muestra la vista `GenerarPDF`, la cual notificará al usuario que el archivo ha sido guardado.

Esta vista tiene la particularidad de incorporar el modelo `RubricaEvaluacion` (línea 1) que permite obtener el identificador de la rúbrica de evaluación y de esta manera asociar dicha rúbrica al correo electrónico correspondiente.

Posteriormente se establece el título del documento HTML (línea 4), un título para la notificación (línea 7) y un mensaje de notificación (línea 9).

Por último se observa un enlace que permite enviar el correo de notificación de evaluación del plan, al presionar en este enlace se ejecutará el método `EmailEvaluacion` con el parámetro identificador de la rúbrica de evaluación, a

través de este identificador se podrá obtener toda la información necesaria para el envío del correo electrónico incluyendo al archivo PDF previamente creado.

```
@model AAPTT.Models.RubricaEvaluacion

@{
    ViewBag.Title = "Notificación PDF Generado";
}

<h2>@ViewBag.NotificacionTitulo</h2>

@ViewBag.NotificacionMensaje

<div>
    @Html.ActionLink("Enviar Correo Electrónico Evaluación", "EmailEvaluacion", new { id = Model.RubricaEvaluacionID })
</div>
```

Código 3.33. Vista `GenerarPDF`

3.3 PRUEBAS

La aplicación fue sometida a pruebas de funcionalidad para comprobar el correcto funcionamiento de cada módulo y del sistema completo. También se realizaron pruebas de validación de datos, las que permitieron garantizar que el usuario ingrese información correcta en cada formulario, y además pruebas de compatibilidad, que permitieron verificar el correcto funcionamiento de la aplicación en distintos navegadores web.

A continuación se presentan varios ejemplos de las pruebas realizadas.

3.3.1 AUTENTICACIÓN DE USUARIOS

Para ingresar a la aplicación es necesario que un usuario se autentique usando su correo electrónico y su contraseña, para esto es necesario dar clic en la opción “Entrar” del menú de la página de inicio, esta página se presenta en la Figura 3.12.

Al hacer clic en “Entrar” se cargará el formulario de *login* correspondiente. Cuando se trata de ingresar a la aplicación y se deja vacío un campo, o no se especifica un correo electrónico con un formato válido, se muestran mensajes de validación que indican al usuario los errores cometidos en el ingreso de la información solicitada para cada campo, tal cual se observa en la Figura 3.13.

Figura 3.12. Página de Inicio

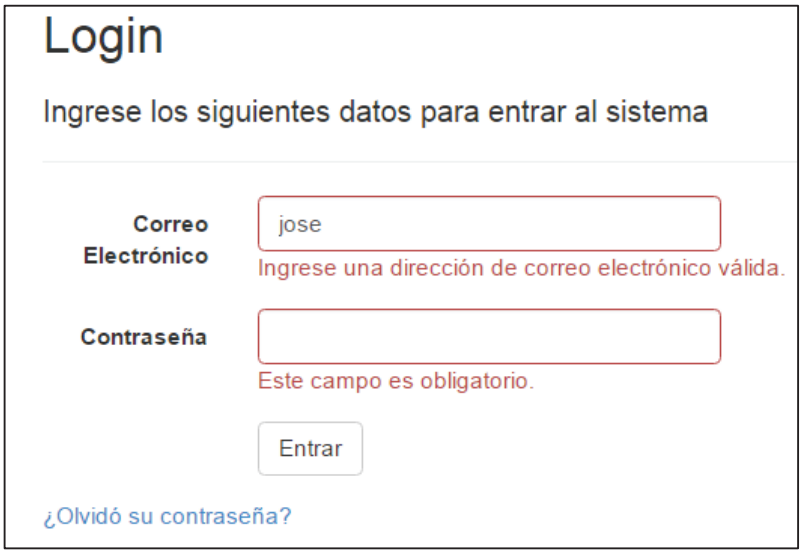

Figura 3.13. Mensajes de validación en el formulario de *login*

También existe la posibilidad de que el usuario al querer ingresar a la aplicación mediante el formulario de *login* digite una contraseña incorrecta. En la Figura 3.14 se presenta el mensaje que indica al usuario dicha situación.

Cuando el usuario ha ingresado una contraseña incorrecta la página vuelve a cargar el formulario manteniendo solo la dirección de correo electrónico en el campo correspondiente.

Figura 3.14. Validación de Correo Electrónico y Contraseña en el formulario de *login*

Si el usuario ingresa correctamente la información en ambos campos, al hacer clic sobre el botón "Entrar", el usuario podrá ingresar a la aplicación y el sistema presentará la página de inicio según el rol asignado a ese usuario.

En la Figura 3.15 se muestra el formulario de *login* y en la Figura 3.16 se muestra la página de inicio para el rol Administrador, en este caso se evidencia el ingreso exitoso del usuario "jabro.pipoec@gmail.com", el cual cuenta con un rol Administrador.

Figura 3.15. Ingreso de datos en el formulario de *login*

Figura 3.16. Página de Inicio para un usuario con rol Administrador

En la Figura 3.17 se presenta la página de inicio para los roles Miembro de Comisión y Secretaria. En esta página se observa que las opciones del menú "Configuración" y "Administración" no son presentadas, ya que dichas opciones solo están disponibles para el administrador.

Figura 3.17. Página de Inicio para un usuario con rol Miembro de Comisión o rol Secretaria

Puede ocurrir que un usuario no recuerde cuál es su contraseña, por lo que el sistema permite generar una nueva contraseña haciendo clic en el enlace "¿Olvidó su contraseña?" del formulario de *login* (Figura 3.15).

En la Figura 3.18 se presenta la página para recuperar la contraseña. Esta página solicita el correo electrónico del usuario para posteriormente, a través del botón "Enviar link", enviar un correo electrónico con un enlace que permite restablecer la contraseña.

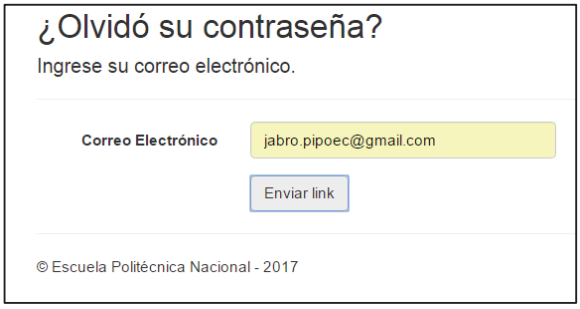

Figura 3.18. Página Olvido Contraseña

Luego de presionar el botón, se carga una página que muestra al usuario un mensaje como se aprecia en la Figura 3.19.

Figura 3.19. Página Confirmación Olvido Contraseña

En la Figura 3.20 se presenta el correo electrónico que contiene el enlace que permitirá restablecer la contraseña del usuario.

Figura 3.20. Correo electrónico que permite restablecer contraseña

Al hacer clic en el enlace para restablecer la contraseña, se cargará un formulario en el explorador web que permite definir una nueva contraseña, este formulario se presenta en la Figura 3.21.

Figura 3.21. Formulario para restablecer una nueva contraseña

Finalmente el usuario contará con una nueva contraseña que permitirá al mismo ingresar a la aplicación. En la Figura 3.22 se presenta la página que se carga luego de realizar la solicitud de restablecimiento de contraseña. Esta página muestra un mensaje que confirma que la contraseña ha sido restablecida, y a través de un enlace, permite al usuario ingresar al sistema.

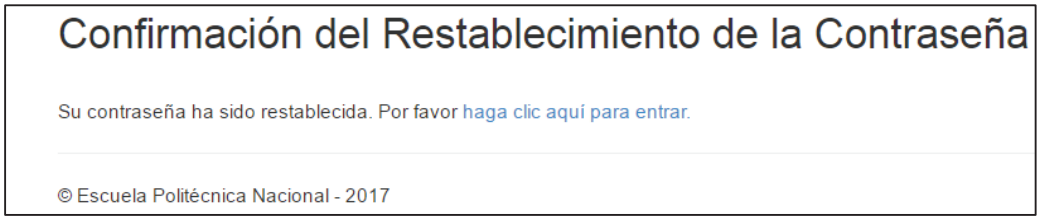

Figura 3.22. Página de Confirmación del Restablecimiento de la Contraseña

3.3.2 ROLES Y USUARIOS

Se han definido tres roles dentro de la aplicación: Administrador, Miembro de Comisión y Secretaria. En esta sección se presentan los resultados de las pruebas realizadas.

3.3.2.1 Gestión de Roles

Para acceder a la gestión de roles es necesario dar clic en el submenú "Administrar Roles" que se encuentra dentro de la opción "Administración" del menú (Figura 3.23), este submenú cargará la página donde se muestran los roles del sistema.

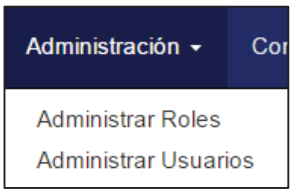

Figura 3.23. Submenús de Administración

En la Figura 3.24 se muestra la página con los roles del sistema: Administrador, Miembro de Comisión y Secretaria. Para cada rol se muestra un enlace que presentará una página con los detalles de dicho rol.

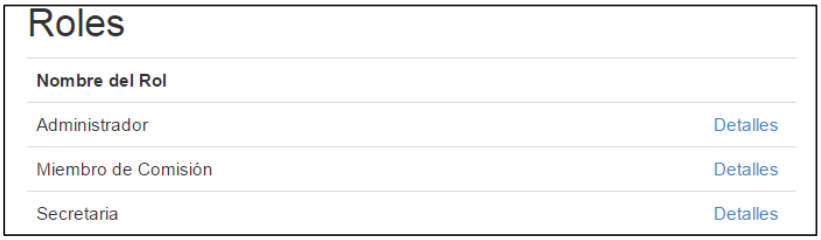

Figura 3.24. Lista de Roles de la Aplicación

En la Figura 3.25 se muestra un ejemplo de los usuarios asignados a un rol, especificamente del rol Miembro de Comisión, en la página "Detalles".

Figura 3.25. Usuarios asignados a un rol

3.3.2.2 Gestión de Usuarios

Un usuario con el rol Administrador es el encargado de crear otros usuarios y asignar un rol a cada usuario.

Al hacer clic en el submenú "Administrar Usuarios" se carga la página que muestra todos los usuarios del sistema (Figura 3.26). En esta página se presenta para cada usuario enlaces que permiten realizar varias acciones, como editar, eliminar u obtener detalles, además se incluye el enlace "Crear nuevo usuario" que permitirá crear un nuevo registro a través del formulario correspondiente.

Figura 3.26. Lista de usuarios de la aplicación

En la Figura 3.27 se muestra el formulario que permite crear un nuevo usuario, sin embargo, en este ejemplo el formulario cuenta con información incorrecta en los campos Correo Electrónico, Contraseña y Confirmar contraseña, esto ocasionará la aparición de mensajes de validación que indican los cambios que se deben realizar para crear el usuario exitosamente. Una vez que se realicen los cambios necesarios, los mensajes de validación desaparecerán, tal como se aprecia en la Figura 3.28.

Para este caso se creó un usuario definido por el correo electrónico "jose.briones.r@hotmail.com", y además se asignó para este usuario el rol Administrador. En la Figura 3.29 se comprueba la creación del usuario "jose.briones.r@hotmail.com" en la lista de usuarios de la aplicación.

Figura 3.27. Mensajes de validación en el formulario que permite crear un usuario

Figura 3.28. Formulario que permite crear un usuario

Figura 3.29. Lista de usuarios con el nuevo usuario

Al hacer clic en el enlace “Editar” del usuario previamente creado se cargará la página presentada en la Figura 3.30. En esta figura se presenta la edición de un registro al asignar el rol Miembro de Comisión al usuario “jose.briones.r@hotmail.com”, el cual ya contaba con el rol Administrador; estos cambios permitirán actualizar dicha información al hacer clic sobre el botón “Guardar”, posterior a esto se cargará nuevamente la lista de usuarios (Figura 3.29).

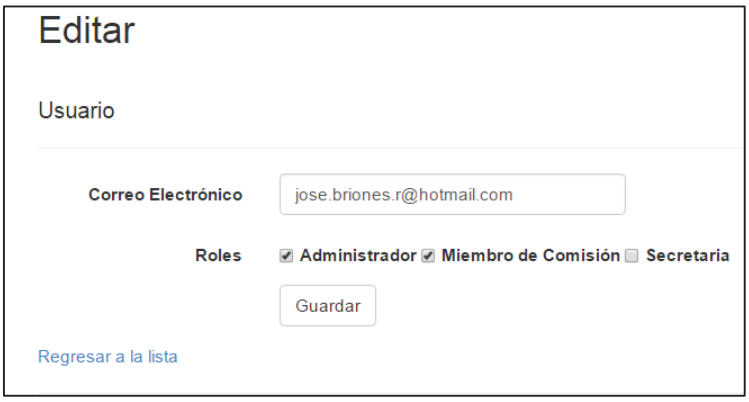

Figura 3.30. Página que permite editar un usuario

Al hacer clic en el enlace “Detalles” del usuario sobre el que se realizó la edición previamente descrita, se carga la página presentada en la Figura 3.31, la cual permite comprobar la edición de dicho registro, al presentar los dos roles con los que ahora cuenta este usuario.

Figura 3.31. Página que muestra los detalles de un usuario

Al hace clic en el enlace ”Eliminar” de un determinado usuario se cargará la página presentada en la Figura 3.32. Una vez que se haga clic sobre el botón “Eliminar”, toda la información del usuario “jabro.pipoec@gmail.com” será borrada.

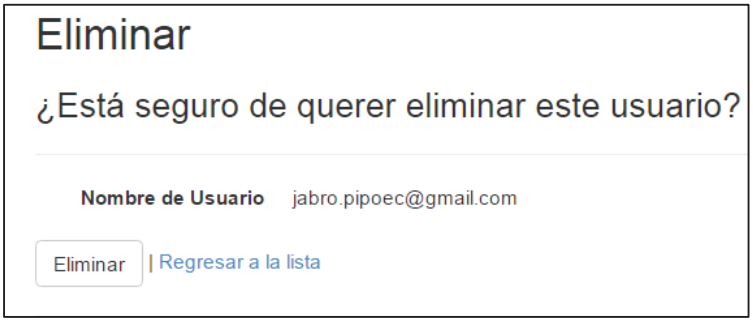

Figura 3.32. Página que permite eliminar un usuario

En la Figura 3.33 se comprueba la eliminación del usuario “jabro.pipoec@gmail.com”, ya que dicho usuario no consta dentro de la lista de usuarios.

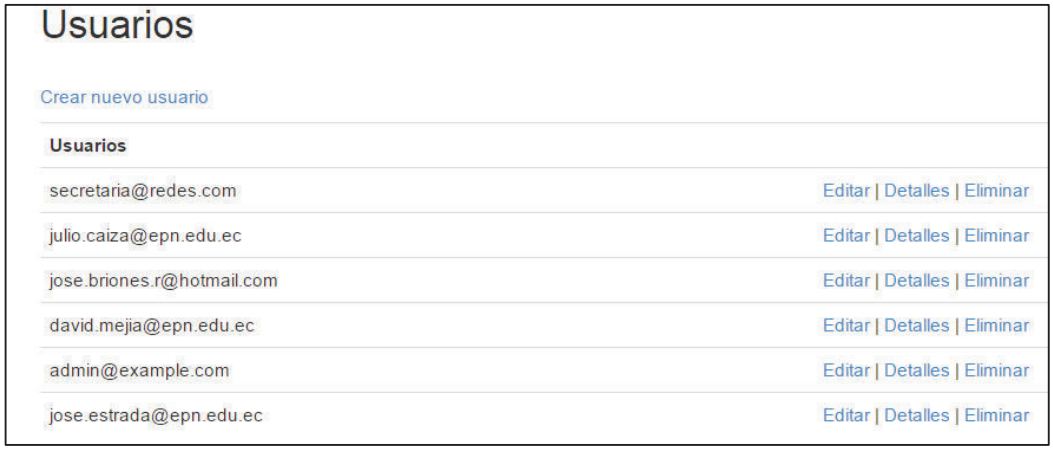

Figura 3.33. Lista de usuarios sin el usuario eliminado

3.3.3 GESTIÓN DE LA INFORMACIÓN

Dentro de los requerimientos se estableció que la aplicación permitirá manipular información correspondiente a estudiantes, docentes y planes de Trabajos de Titulación. A continuación se muestran solamente los resultados de las pruebas

de funcionalidad correspondientes a la gestión de registros de estudiantes, puesto que para el resto de entidades el proceso es muy similar.

Para poder manipular información de estudiantes es necesario acceder a la opción “Estudiantes y Docentes” del menú; el cual se muestra en la Figura 3.34. Dentro de esta opción se debe elegir el submenú “Estudiantes”, con lo cual el sistema presentará una página donde se listan los registros de los estudiantes existentes.

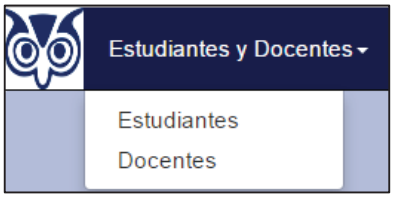

Figura 3.34. Submenús de Estudiantes y Docentes

En la Figura 3.35 se presenta un segmento de la página que cuenta con los registros de varios estudiantes. En esta página se puede crear, editar, buscar o mostrar los detalles de un registro determinado.

Figura 3.35. Página de Gestión de Estudiantes

Para crear un registro de un estudiante es necesario hacer clic en el enlace “Crear Nuevo”, esto permitirá al sistema cargar un formulario mediante el cual se permita crear el registro deseado.

Si al querer crear el registro de estudiante se ingresa información errónea se desplegarán mensajes de validación que indiquen al usuario los cambios que este debe hacer para poder crear el registro. En la Figura 3.36 se muestran mensajes de validación para: el campo Cédula, ya que no se han ingresado los 10 dígitos correspondientes a este campo, los campos Nombres, Apellidos y Teléfono, ya que no existe información para estos campos, y para el campo Correo Electrónico, ya que no se ha ingresado información con un formato válido.

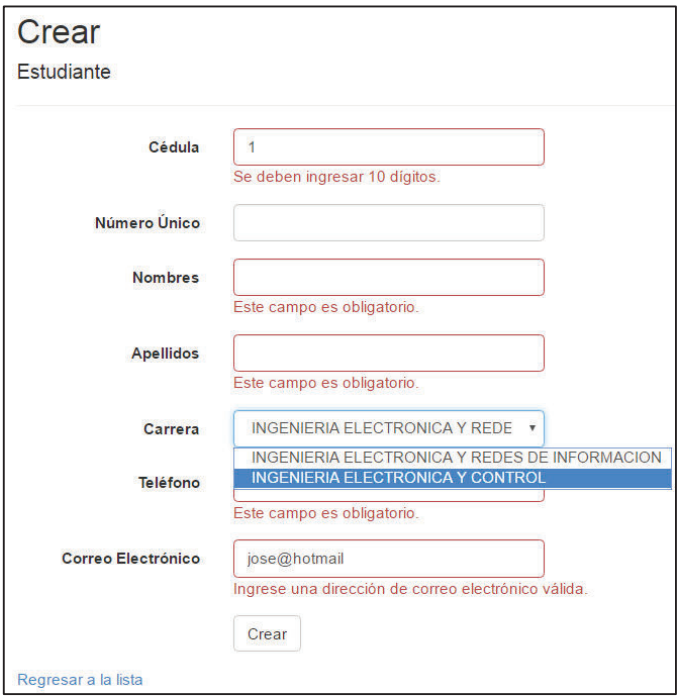

Figura 3.36. Mensajes de validación en el formulario que permite crear un registro de estudiante

Luego de que el usuario ingrese la información correcta para cada campo, los mensajes de validación desaparecerán y el formulario lucirá como se presenta en la Figura 3.37. Si no existen mensajes que alerten de información errónea en algún campo, se hace clic en el botón “Crear” y se crea el registro correspondiente, en este caso el del estudiante “JUAN ANDRES BELTRAN AVILES”. Posteriormente se cargará la página en la cual se listan los registros de estudiantes existentes en el sistema, es decir, la página de Gestión de Estudiantes (Figura 3.35).

Figura 3.37. Formulario que permite crear un estudiante

Para evidenciar la creación del registro de estudiante previamente mencionado se realiza una búsqueda en la página de Gestión de Estudiantes. Dentro del cuadro de texto definido para buscar un registro se ingresa el apellido del estudiante y se hace clic en el botón “Buscar”.

En la Figura 3.38 se presenta el resultado de la búsqueda realizada, mostrando el registro del estudiante “JUAN ANDRES BELTRAN AVILES” y verificando de esta manera la creación de este. Se puede observar también que este registro cuenta con la propiedad Estado definida como activa ya que se muestra un *checkbox* marcado, esto sucede ya que al crear un registro se asigna el estado de activo por defecto.

Figura 3.38. Búsqueda de un registro de estudiante en la página Gestión de Estudiantes

En la Figura 3.39 se presenta la edición de un registro de estudiante. En esta página se muestra que el *checkbox* de la propiedad Estado se ha desmarcado, esto indica que dicho registro de estudiante pasará a estar inactivo y por tal razón no se podrán asignar planes de Trabajos de Titulación a este estudiante.

Figura 3.39. Página que permite editar un registro de estudiante

Para verificar que el registro ha sido editado se realiza una búsqueda que permite mostrar el registro correspondiente. En la Figura 3.40 se presenta el resultado de la búsqueda realizada, esto comprueba la edición del registro ya que en la propiedad Estado se presenta un *checkbox* sin marcar, lo cual indica el estado inactivo que se le asignó en la edición del registro.

Figura 3.40. Búsqueda de un registro de estudiante que permite comprobar la edición de dicho registro

3.3.4 BÚSQUEDA DE REGISTROS

La aplicación permite buscar registros correspondientes a estudiantes, docentes y planes de Trabajos de Titulación, pero también permite realizar búsquedas de planes evaluados, de estudiantes y los planes que estos han realizado, y de docentes y los planes que estos han dirigido o codirigido. En esta sección se presentan los resultados de las pruebas de funcionalidad realizadas para las búsquedas de registros asociados a planes de Trabajos de Titulación.

Dentro de la opción “Planes” del menú, presentado en la Figura 3.41 se muestran varios submenús que permiten al usuario interactuar con los planes de Trabajos de Titulación.

Figura 3.41. Submenús de Planes

Al hacer clic en el submenú “Planes de TT” se carga una página que lista a los planes existentes en la aplicación. En la Figura 3.42 se presenta un segmento de esta página, en la cual se permite realizar una búsqueda más avanzada al usar diferentes filtros que permiten discriminar información.

En la Figura 3.43 se presenta el resultado de una búsqueda avanzada para un plan de Trabajo de Titulación. Esta búsqueda permite encontrar planes que contengan dentro de su título la palabra “plan” y además que cuenten con un estado: Aprobado.

Al hacer clic en el submenú “Estudiantes y Planes de TT” (Figura 3.41), se carga una página que lista a estudiantes que han realizado un determinado plan de

Trabajo de Titulación. En la Figura 3.44 se presenta el resultado de la búsqueda realizada para determinar los planes realizados por estudiantes que tengan el apellido “Aguirre”.

Figura 3.42. Lista de planes de Trabajos de Titulación

Figura 3.43. Búsqueda avanzada de planes

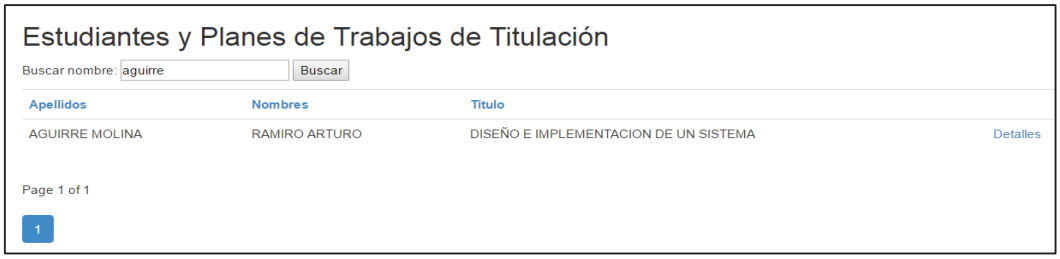

Figura 3.44. Búsqueda de un estudiante y sus planes de Trabajos de Titulación

Al hacer clic en el submenú “Directores y Planes de TT” (Figura 3.41), se carga una página que lista a docentes que han dirigido Trabajos de Titulación. En la Figura 3.45 se presenta el resultado de la búsqueda realizada para determinar los trabajos dirigidos por docentes que tengan el apellido “Espinoza”.

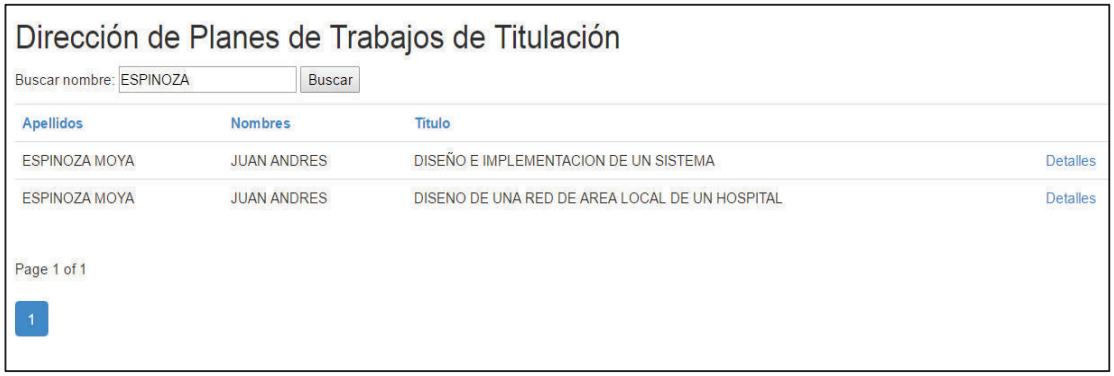

Figura 3.45. Búsqueda de trabajos dirigidos por un docente

Al hacer clic en el submenú “Codirectores y Planes de TT” (Figura 3.41) se carga una página que lista a docentes que han codirigido Trabajos de Titulación. En la Figura 3.46 se presenta el resultado de la búsqueda realizada para determinar los planes codirigidos por docentes que tengan el apellido “Morales”.

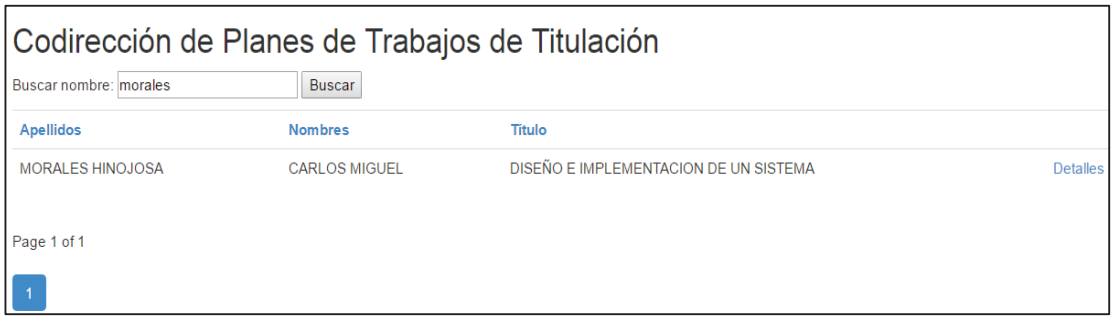

Figura 3.46. Búsqueda de trabajos codirigidos por un docente

Al hacer clic en el submenú “Planes de TT Evaluados” (Figura 3.41) se carga una página que lista a los planes evaluados en la aplicación.

En la Figura 3.47 se presenta el resultado de la búsqueda realzada para determinar planes evaluados que contengan en su título la palabra “plan”, se

observa para este caso que no se ha filtrado información permitiendo mostrar registros con cualquier estado.

Figura 3.47. Búsqueda de planes evaluados

3.3.5 ENVÍO DE CORREOS ELECTRÓNICOS

A través de enlaces presentes en la página Gestión de Planes de Trabajos de Titulación (Figura 3.42) y en la página Gestión de Planes de Trabajos de Titulación Evaluados (Figura 3.47) se pueden enviar correos electrónicos para notificar el ingreso de un plan en el sistema y para notificar la evaluación realizada a un determinado plan, respectivamente.

Cada vez que se haga clic en el enlace correspondiente al envío de un correo electrónico, se mostrará la página presentada en la Figura 3.48, esta página notifica al usuario que el correo electrónico ha sido enviado.

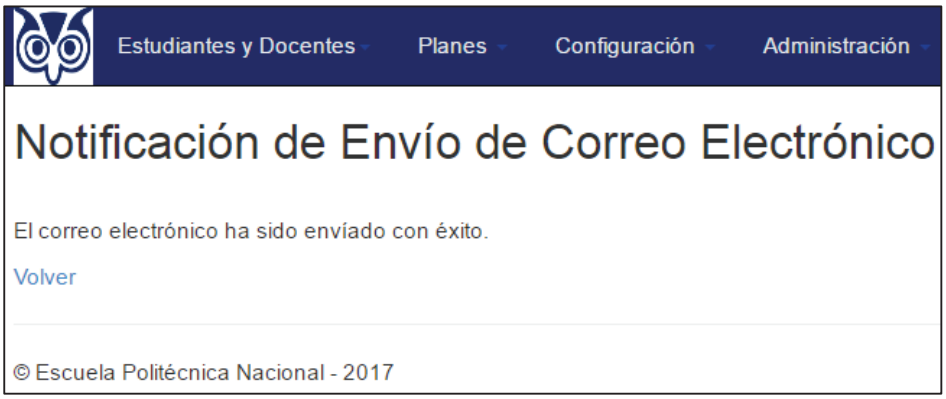

Figura 3.48. Página Notificación de Envío de Correo Electrónico

A continuación se muestran capturas de correos electrónicos que permiten comprobar el envío de los correos por parte de la aplicación para cada uno de los casos definidos en las historias de usuario correspondientes.

En la Figura 3.49 se presenta el correo electrónico que está dirigido a estudiantes y docentes, y que se envía luego del ingreso del plan al sistema.

Figura 3.49. Correo electrónico que notifica el ingreso de un plan al sistema

En la Figura 3.50 se presenta el correo electrónico que está dirigido a los miembros de la Comisión luego del ingreso del plan al sistema, se observa también que este correo adjunta un archivo PDF que contiene el plan escaneado.

En la Figura 3.51 se presenta el correo electrónico que notifica a estudiantes y docentes de la evaluación de un plan. Para este caso se presenta un plan que ha mantenido el estado de "Pendiente por Aprobar", se observa también que este correo adjunta un archivo PDF que contiene la rúbrica de evaluación utilizada, la misma que permitirá notificar al estudiante de las modificaciones que se deben hacer al plan para posteriormente poder aprobarlo

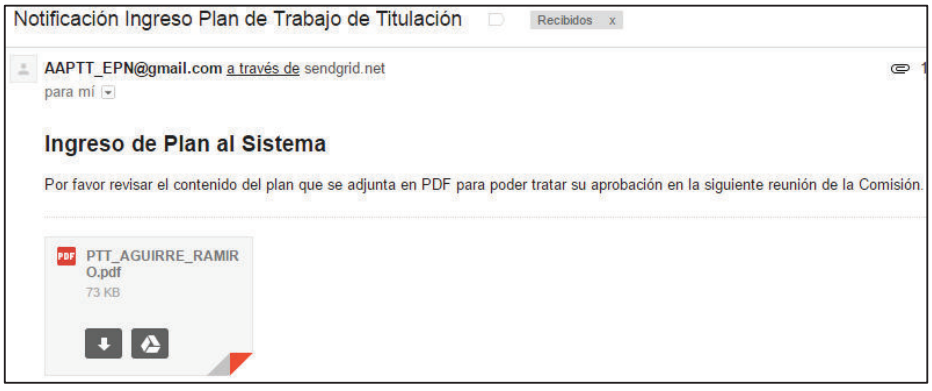

Figura 3.50. Correo electrónico enviado a los miembros de la Comisión al ingresar un plan al sistema

Figura 3.51. Correo electrónico que notifica las observaciones hechas al plan

En la Figura 3.52 se presenta el correo electrónico que notifica a estudiantes y docentes de la evaluación de un plan que ha sido negado, se observa también que este correo adjunta un archivo PDF que contiene la rúbrica de evaluación utilizada.

Figura 3.52. Correo electrónico que notifica la negación de un plan

En la Figura 3.53 se presenta el correo electrónico que notifica a estudiantes y docentes de la evaluación de un plan que ha sido aprobado, se observa también que este correo adjunta un archivo PDF que contiene la rúbrica de evaluación utilizada.

Figura 3.53. Correo electrónico que notifica la aprobación de un plan

3.3.6 EVALUACIÓN DE PLANES A TRAVÉS DE RÚBRICAS

En esta sección se presentan los resultados de las pruebas funcionales que se han realizado al crear y utilizar una rúbrica para evaluar un determinado plan.

3.3.6.1 Creación de una rúbrica

Para crear una rúbrica se debe hacer clic en el submenú “Rúbricas” de la opción “Configuración”, esta opción se presenta en la Figura 3.54. Solo usuarios que tengan el rol Administrador podrán crear una rúbrica.

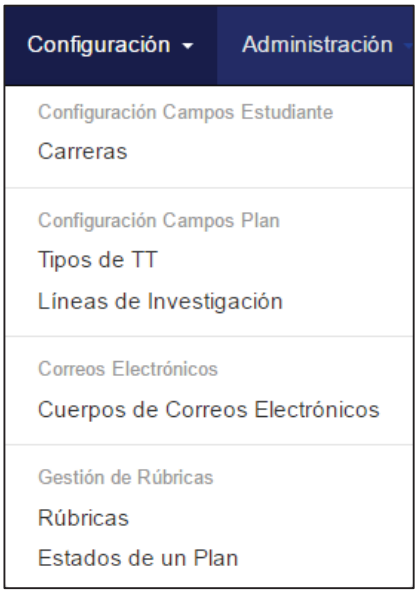

Figura 3.54. Submenús de Configuración

Posteriormente se cargará la página que se presenta en la Figura 3.55, en esta página se presentan las rúbricas existentes en la aplicación y se permite crear una nueva rúbrica a través del enlace “Crear Rúbrica”.

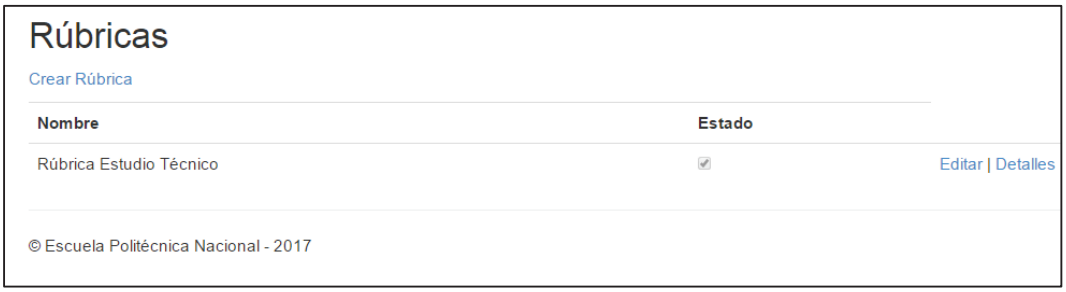

Figura 3.55. Lista de rúbricas

La creación de una rúbrica se la hace en dos pasos, primero se define el nombre de la rúbrica y los valores de porcentajes máximos para cada estado, y después se arma la rúbrica a partir de la agregación de criterios de evaluación.

Al hacer clic en el enlace “Crear Rúbrica” se carga la página presentada en la Figura 3.56. Para este caso se asigna el nombre “Rúbrica Proyecto Integrador” y los porcentajes máximos 30, 70 y 100 para los estados NEGADO, PENDIENTE POR APROBAR y APROBADO, respectivamente. Luego de ingresar esta información se hace clic en el botón “Crear” para crear el registro correspondiente y posteriormente cargar la página en donde se asignarán los criterios a la rúbrica.

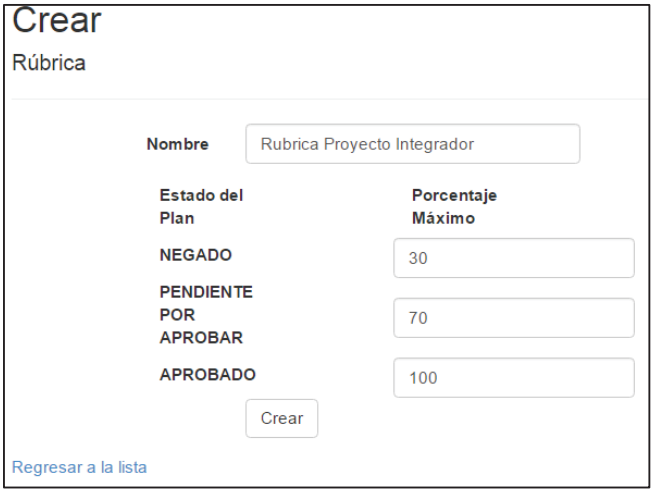

Figura 3.56. Página Crear Rúbrica

En la Figura 3.57 se presenta la página que permite armar la rúbrica mediante la adición de criterios de evaluación. Esta página permite agregar o quitar criterios al hacer clic sobre los botones correspondientes.

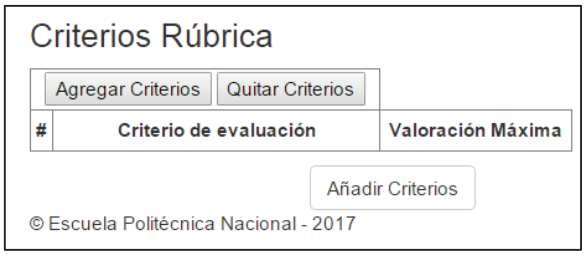

Figura 3.57. Página Criterios Rúbrica

Luego de agregar los criterios necesarios se hace clic en el botón "Añadir Criterios" para finalizar el proceso de creación de la rúbrica. Si existen campos vacíos en algún criterio se mostrará un mensaje de validación tal como se aprecia en la Figura 3.58.

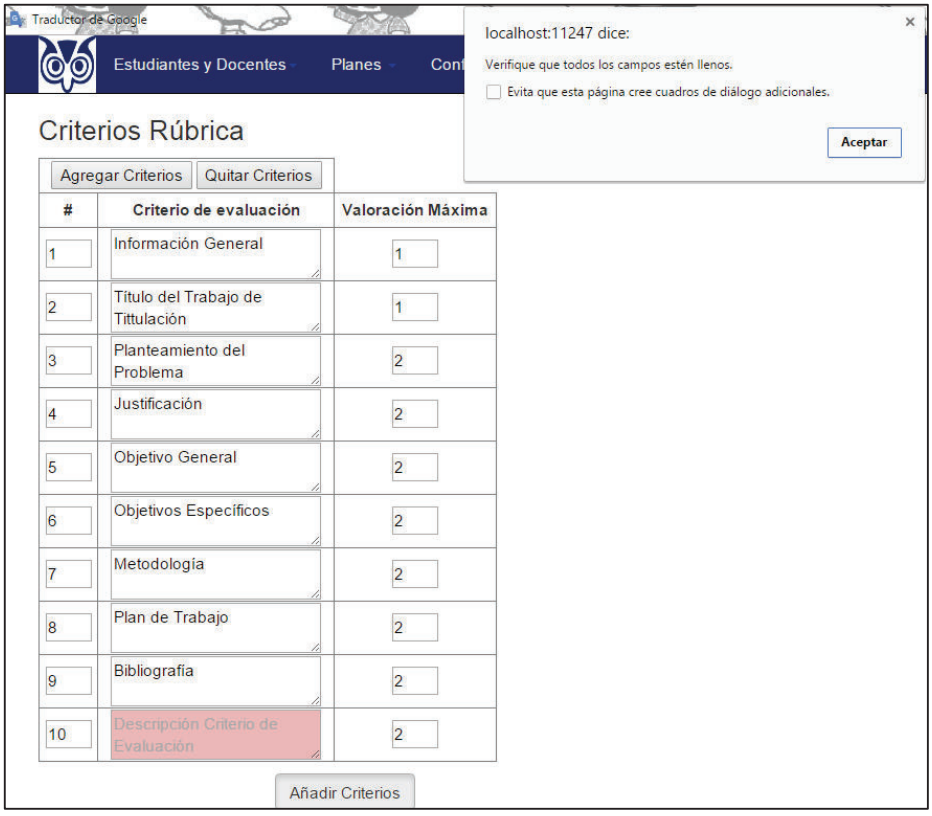

Figura 3.58. Validación de campos en la Página Criterios Rúbrica

Una vez que no existan campos vacíos y se añadan los criterios correspondientes al hacer clic en el botón “Añadir Criterios”, se cargará la página Rúbricas, en donde se presentan las rúbricas existentes en la aplicación. En la Figura 3.59 se evidencia la creación de la rúbrica denominada “Rúbrica Proyecto Integrador” a través de la página Rúbricas.

Figura 3.59. Página Rúbricas con la nueva rúbrica creada

3.3.6.2 Uso de una rúbrica para evaluar un plan

Para evaluar un plan de Trabajo de Titulación se debe elegir primero una rúbrica con la que se pueda calificar a dicho plan, para esto se debe hacer clic en el submenú “Evaluar un Plan de TT” (Figura 3.41), esto permitirá cargar la página que se presenta en la Figura 3.60. Esta página muestra todas las rúbricas que tienen un estado activo, cada rúbrica cuenta con un enlace que permitirá elegir dicha rúbrica para evaluar un plan de Trabajo de Titulación.

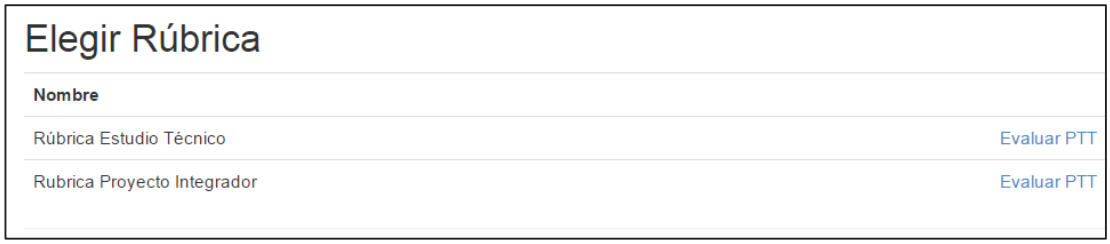

Figura 3.60. Página Elegir Rúbrica

Luego de elegir la rúbrica deseada se carga la página que se presenta en la Figura 3.61. Esta página muestra información sobre la rúbrica escogida previamente y permite elegir el plan a evaluar a través del botón que cuenta con

un ícono de lupa. Al hacer clic en este botón se cargará una ventana modal que permitirá elegir el plan de Trabajo de Titulación a evaluar.

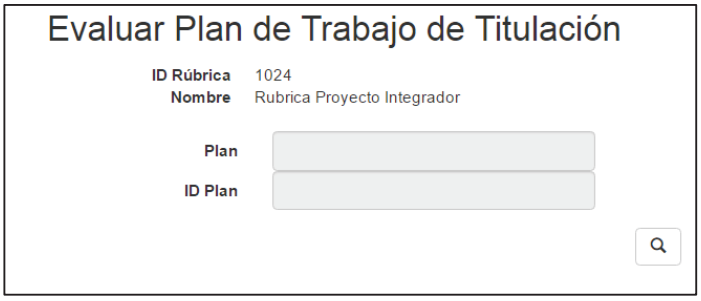

Figura 3.61. Página Evaluar Plan de Trabajo de Titulación

En la Figura 3.62 se presenta la ventana modal que permite elegir el plan a evaluar. Para este caso se ha buscado y elegido el plan que tiene por título: "DISEÑO E IMPLEMENTACION DE UN SISTEMA". En esta búsqueda solo estarán disponibles planes con el estado: Pendiente por Aprobar.

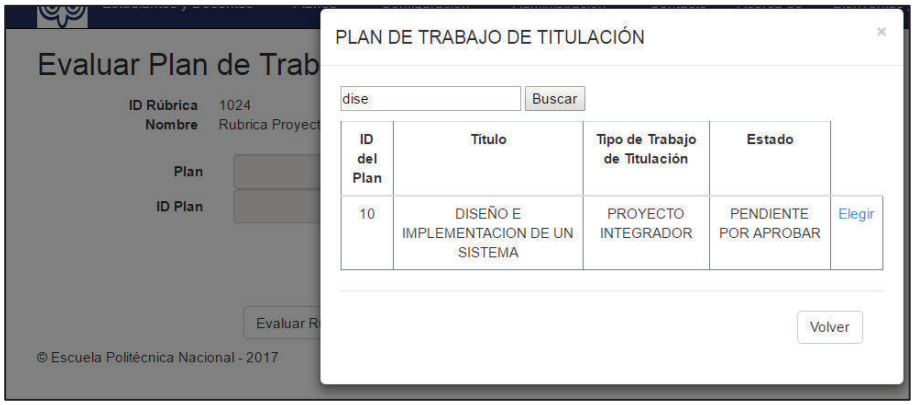

Figura 3.62. Ventana modal que permite elegir un plan para ser evaluado

Luego de elegir el plan que se quiera evaluar, la ventana modal se cerrará, la información del plan será asignada y la rúbrica se cargará en la página Evaluar Plan de Trabajo de Titulación, tal cual se observa en la Figura 3.63. Más adelante se asigna una valoración para cada criterio de evaluación y se escriben las observaciones necesarias, para finalmente concluir la evaluación del plan haciendo clic en el botón "Evaluar Plan".

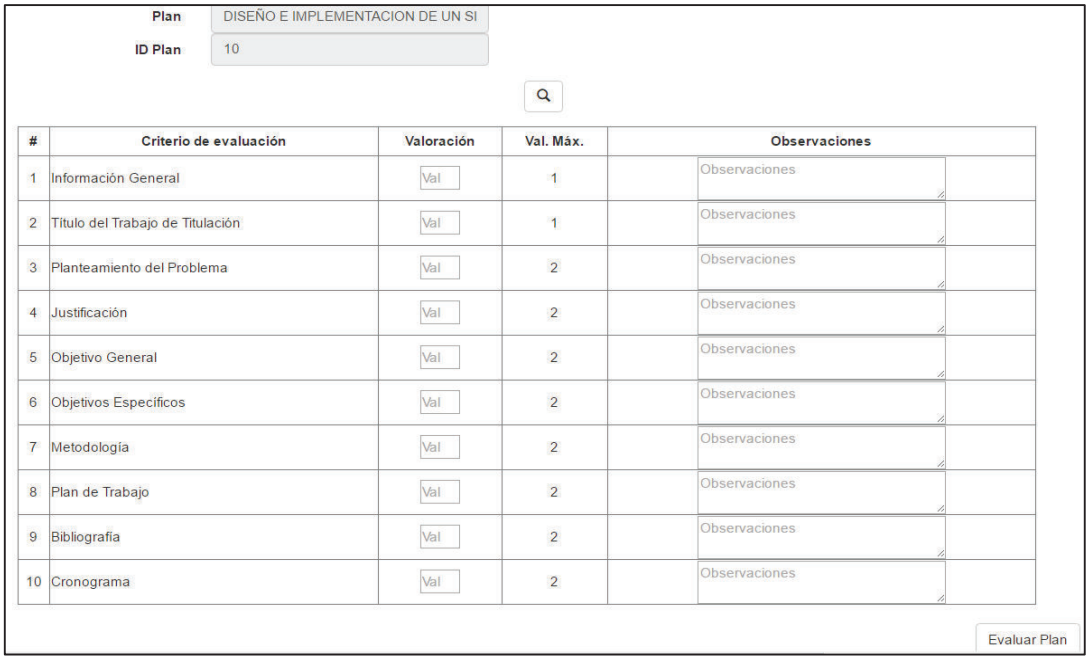

Figura 3.63. Página Evaluar Plan de Trabajo de Titulación con la rúbrica cargada

Si existe algún criterio que no define una valoración se mostrará un mensaje de validación tal como se presenta en la Figura 3.64.

Figura 3.64. Validación de campos en la Página Evaluar Plan de Trabajo de Titulación

Una vez que no existan criterios sin valoración y se haga clic sobre el botón “Evaluar Plan” se dará fin a la evaluación del plan y además se cargará la página Rúbricas permitiendo empezar nuevamente con el proceso de evaluación de un plan.

Para comprobar que se ha realizado la evaluación previamente descrita se procede a cargar la página donde se listan los planes evaluados. Esta página se presenta en la Figura 3.65. Se observa que el primer plan de la lista es el plan previamente descrito, ya que este tiene el título: “DISEÑO E IMPLEMENTACION DE UN SISTEMA” y ha sido evaluado con la rúbrica “Rúbrica Proyecto Integrador”.

Figura 3.65. Página Planes de Trabajos de Titulación Evaluados

3.3.7 PRUEBAS DE COMPATIBILIDAD

En esta sección se muestran resultados de pruebas de compatibilidad realizadas usando los navegadores Google Chrome y Mozilla Firefox. Cada requerimiento establecido ha sido probado con éxito en ambos navegadores.

A continuación y mediante el formulario de creación de un plan de Trabajo de Titulación y la página de adición de criterios a la rúbrica en su fase de creación, se presentan los resultados de pruebas de compatibilidad que permiten constatar el correcto funcionamiento de la aplicación en diversas situaciones tales como: mostrar mensajes de validación, cargar ventanas modales, abrir exploradores de archivos y correr *scripts* de JavaScript tanto para Google Chrome como para Mozilla Firefox.

3.3.7.1 Google Chrome

En la Figura 3.66 se presenta el formulario de creación de un plan de Trabajo de Titulación cuando se intenta crear un plan dejando campos vacíos, por lo que se presentan mensajes de validación para los campos que no pueden estar vacíos. De esta manera se verifica que estos mensajes son cargados sin ningún problema en este navegador.

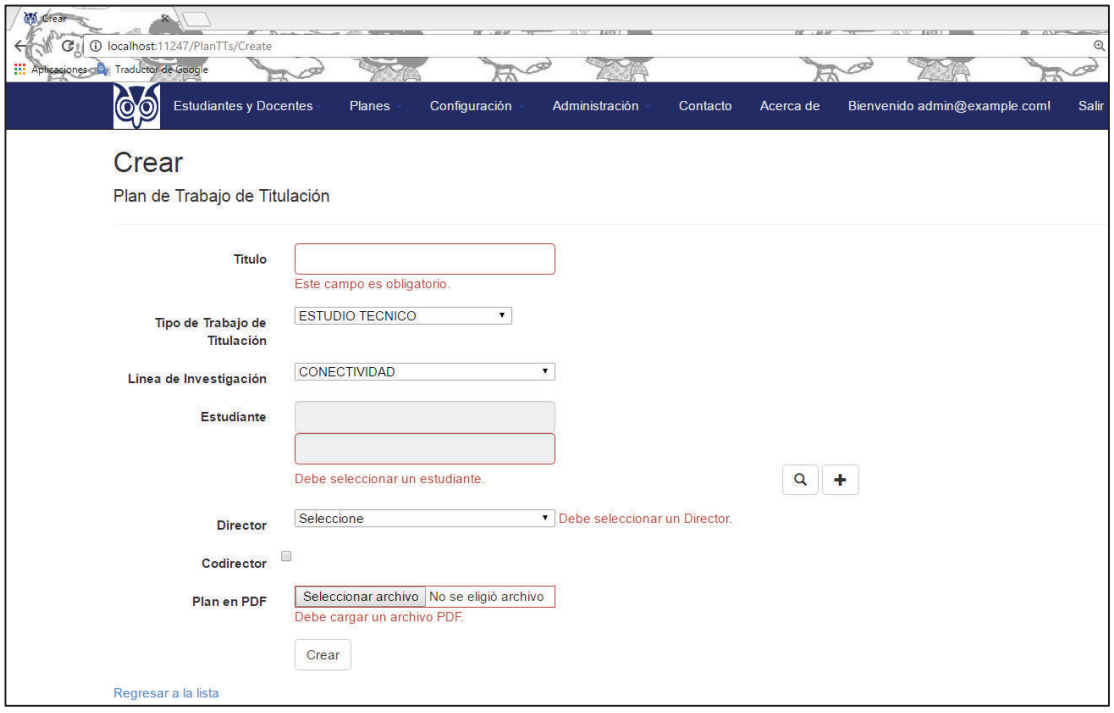

Figura 3.66. Mensajes de validación en formulario de creación de un plan en Google Chrome

En la Figura 3.67 se presenta la ventana modal que permite buscar y elegir un estudiante que sea parte del plan a crear. Se observa que la ventana modal ha podido ser cargada adecuadamente en Google Chrome, pero también se evidencia el funcionamiento correcto de AJAX al presentar el resultado de la búsqueda realizada en la ventana modal. En la Figura 3.68 se presenta un explorador de archivos que es cargado al dar clic en el botón “Seleccionar archivo”, el cual permite elegir un archivo PDF que contendrá el plan presentado por los estudiantes. Se observa que el explorador de archivos es cargado sin ningún problema usando Google Chrome.

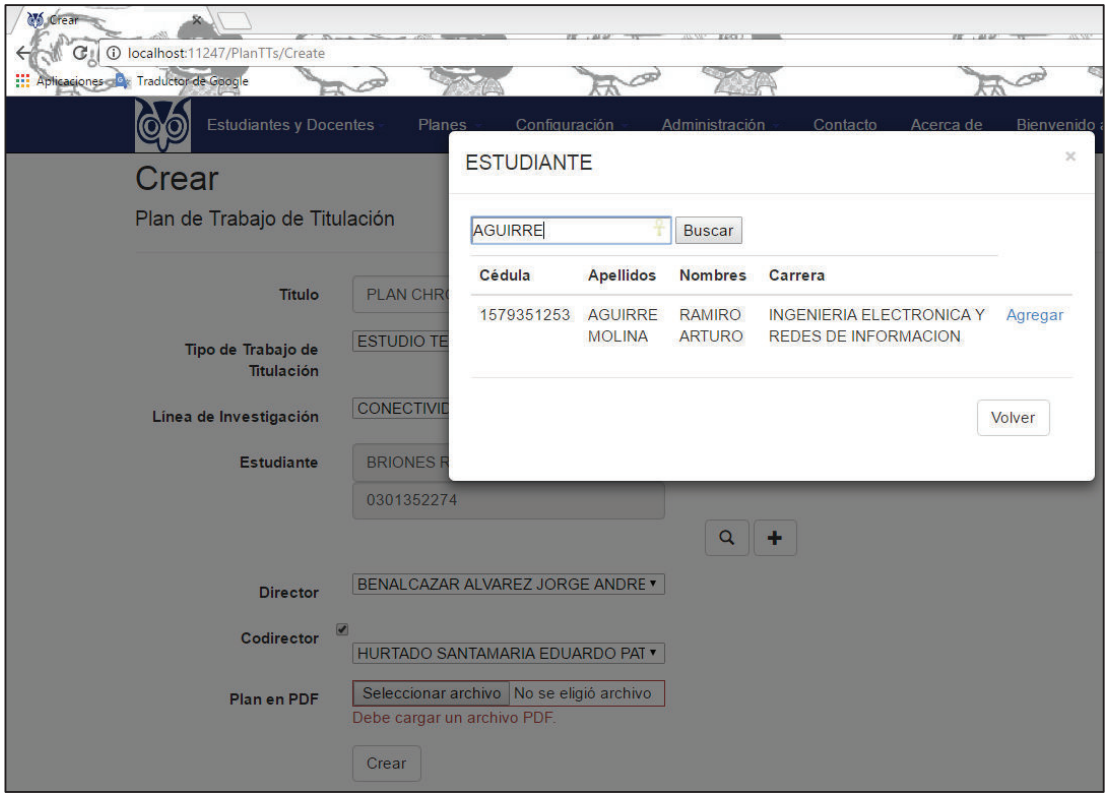

Figura 3.67. Ventana Modal cargada en Google Chrome

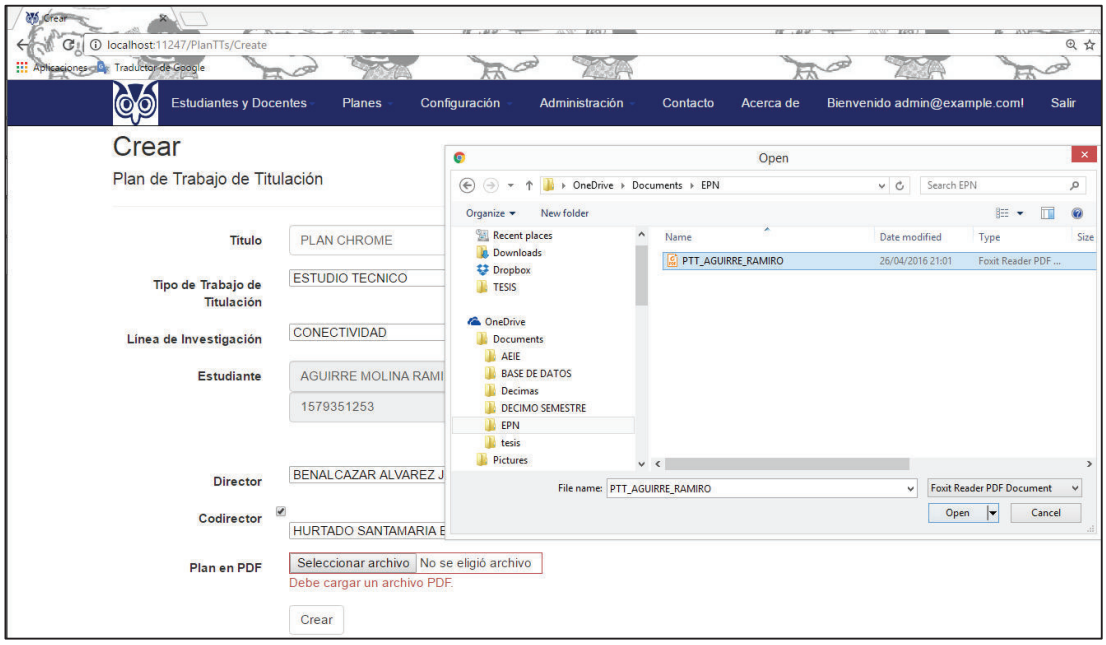

Figura 3.68. Explorador de Archivos cargado en Google Chrome

En la Figura 3.69 se presenta la página Criterios Rúbricas cuando se intentan añadir criterios de evaluación a una rúbrica pero se tienen campos sin información, por lo que se carga un mensaje de validación que notifica al usuario la presencia de campos vacíos dentro de los criterios añadidos. De esta manera se evidencia el funcionamiento correcto de JavaScript en Google Chrome al permitir añadir los criterios a la rúbrica y al mostrar el mensaje de validación.

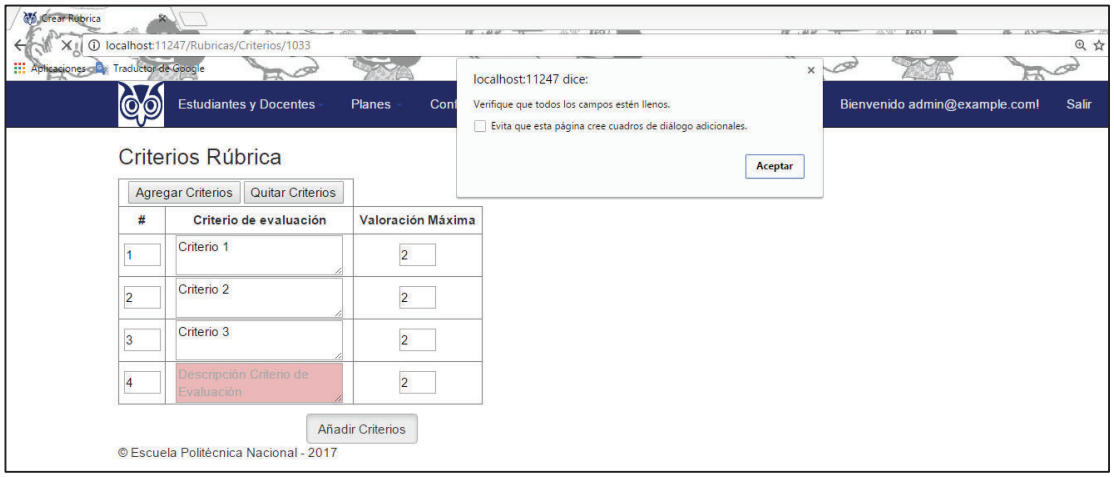

Figura 3.69. Mensaje de validación en página Criterios Rúbrica en Google Chrome

3.3.7.2 Mozilla Firefox

Para comprobar el correcto funcionamiento de la aplicación en el navegador Mozilla Firefox se han realizado las mismas pruebas presentadas anteriormente.

Por lo tanto se presenta en la Figura 3.70, el formulario de creación de un plan de Trabajo de Titulación al intentar crear un plan dejando campos vacíos, en la Figura 3.71, la ventana modal que permite buscar y elegir un estudiante en el formulario de creación de un plan, en la Figura 3.72, un explorador de archivos que es cargado al dar clic en el botón “Examinar…”, el cual permite elegir un archivo PDF que contendrá el plan presentado por los estudiantes, y en la Figura 3.73, la página Criterios Rúbricas con un mensaje de validación que se carga al intentar añadir criterios de evaluación aun cuando existen campos vacíos. De esta manera se evidencia que no existe error alguno al momento de usar JavaScript, AJAX, ventanas modales, exploradores de archivos y mensajes de validación en Firefox.

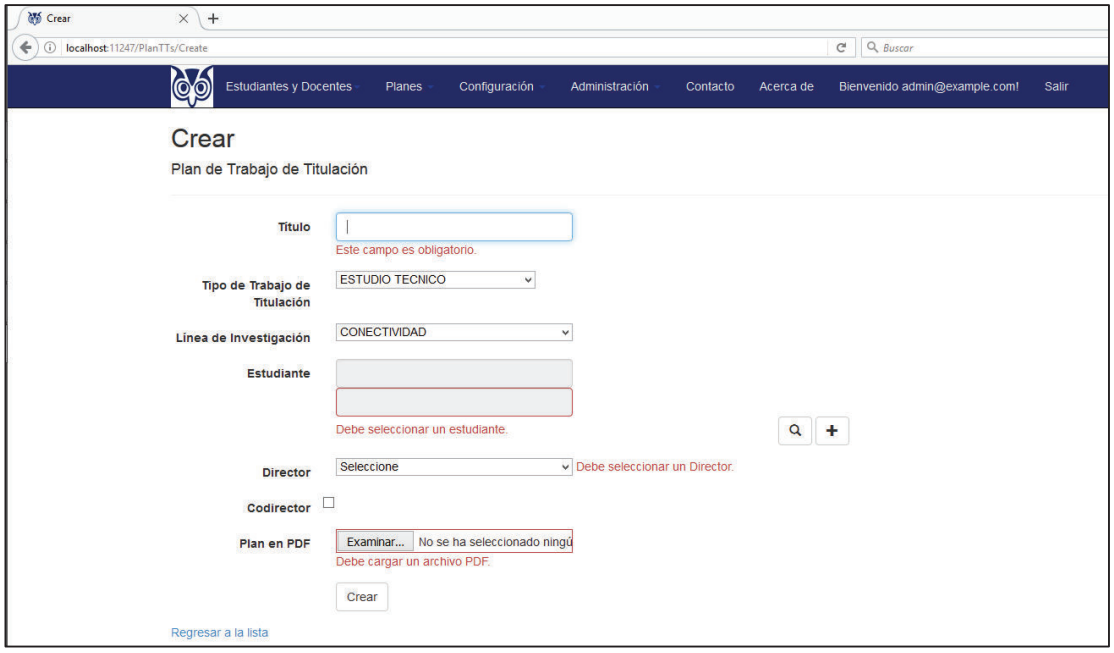

Figura 3.70. Mensajes de validación en formulario de creación de un plan en Mozilla Firefox

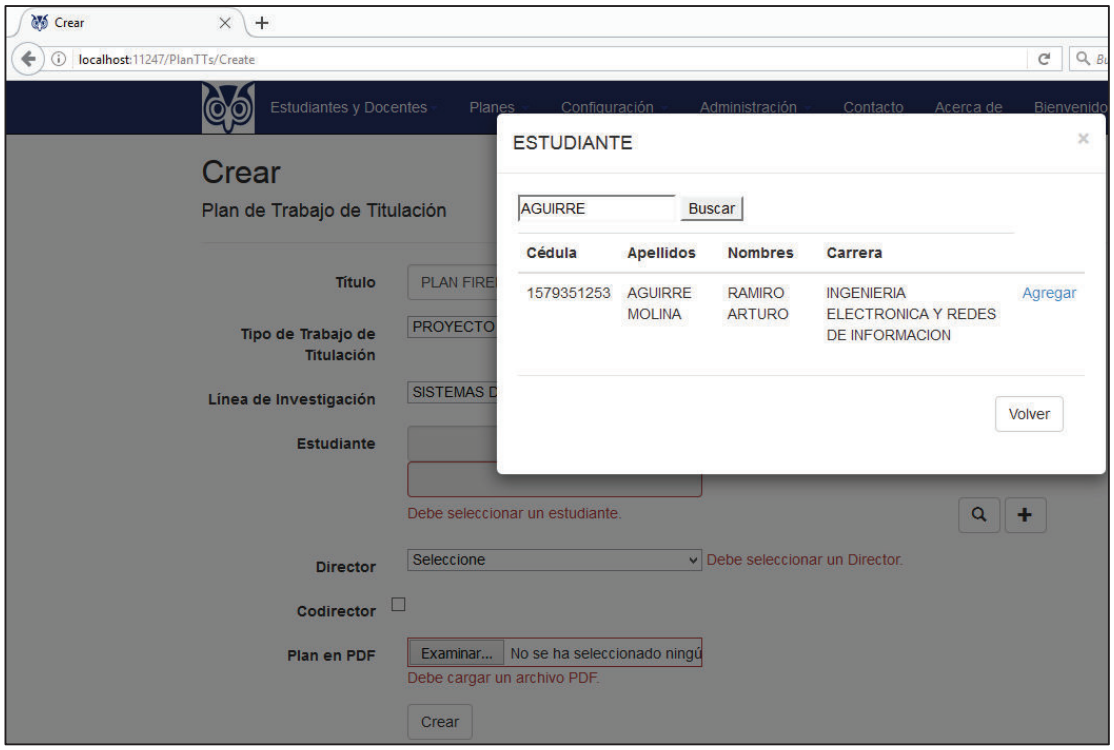

Figura 3.71. Ventana Modal cargada en Mozilla Firefox

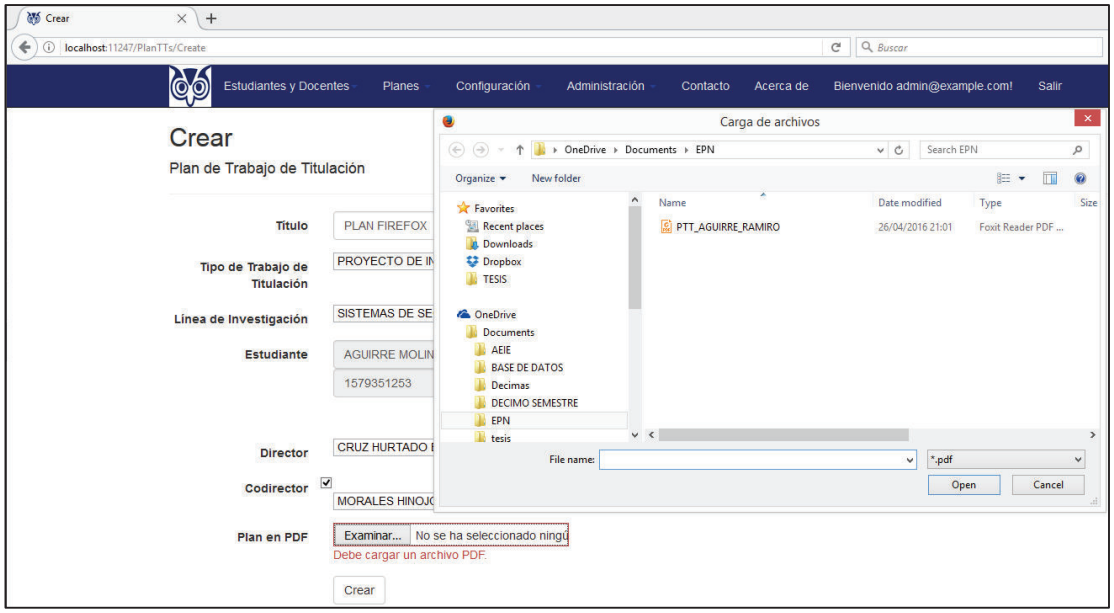

Figura 3.72. Explorador de Archivos cargado en Mozilla Firefox

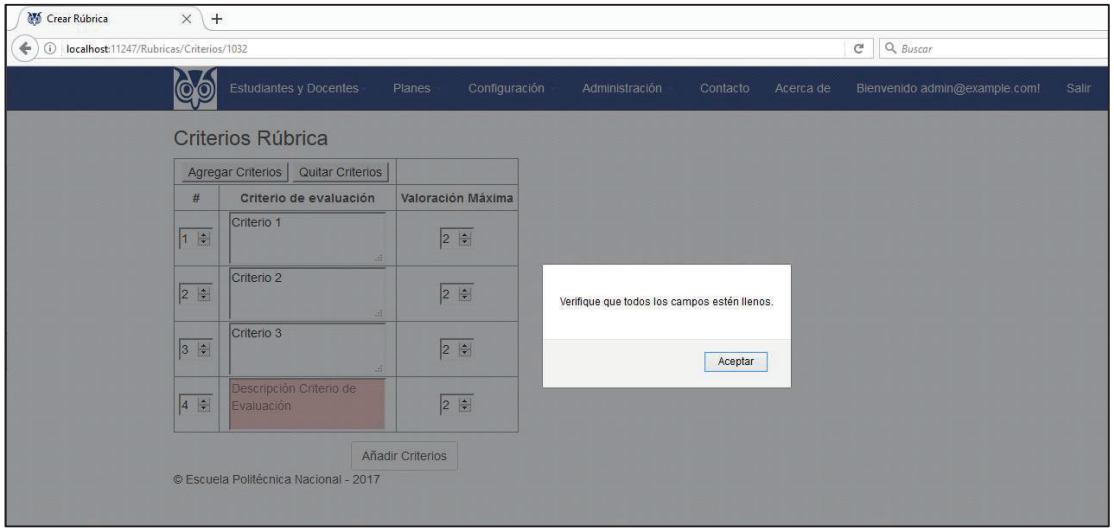

Figura 3.73. Mensaje de validación en página Criterios Rúbrica en Mozilla Firefox

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- La presente aplicación web ha sido desarrollada para ser una herramienta que permita automatizar el proceso de aprobación de planes de Trabajos de Titulación en la Carrera de Ingeniería Electrónica y Redes de Información generando una interacción directa entre las personas involucradas y agilizando la evaluación de los planes. Esta aplicación fue desarrollada siguiendo la metodología Kanban y consta de cuatro módulos que definen los requerimientos generados a través de 32 historias de usuario.

- En el primer módulo se define el sistema de autenticación de la aplicación y la gestión de usuarios y de roles de usuarios. En el segundo módulo se gestiona la información correspondiente a estudiantes, docentes y planes de Trabajos de Titulación, permitiendo la creación, edición, búsqueda y desactivación de registros. En el tercer módulo se maneja la personalización y el envío de correos electrónicos, y en el cuarto módulo se gestiona la creación y el uso de rúbricas para evaluar un determinado plan de Trabajo de Titulación.

- Esta aplicación ha sido desarrollada usando el patrón de diseño Modelo-Vista-Controlador a través del *framework* ASP.NET MVC, en donde el controlador gestiona las acciones que permiten responder a las solicitudes del usuario a través del modelo, el cual se ocupa de manejar los datos de la aplicación, y la vista, que presenta la información necesaria generando contenido HTML que será desplegado en un explorador web.

- Como parte del desarrollo de la aplicación se levantó la información sobre el proceso actual que permite aprobar los planes de Trabajos de Titulación en la Carrera de Ingeniería en Electrónica y Redes de Información, identificando cómo funciona el proceso y quiénes están involucrados.

- El uso de herramientas como Entity Framework, ASP.NET Identity, iTextSharp o Postal reducen el código que el programador debería implementar sobre una funcionalidad en particular. Para el caso de Entity Framework se evita generar clases que manejen operaciones SQL ya que gracias al enfoque *Code First* se crean modelos que se asocien a tablas en la base de datos y se define dicha relación en la clase Context, permitiendo que el programador se dedique únicamente a escribir código correspondiente a la funcionalidad de las clases. ASP.NET Identity permite gestionar un sistema de autenticación sin necesidad de preocuparse por los procesos internos que se realizan para lograr dicha funcionalidad. Por su parte iTextSharp permite generar archivos PDF a través de la definición de los elementos del archivo mediante el uso de objetos C# y Postal permite crear correos electrónicos a través de la personalización de vistas y de objetos propios del paquete.

- El uso de *Scaffolding* evita al programador tener que escribir código específico en el controlador y las vistas, ya que este mecanismo genera a partir de un modelo un conjunto de plantillas que se pueden modificar en función de las necesidades del desarrollador.

- Para que una aplicación web tenga una funcionalidad adecuada a veces es necesario usar otras herramientas como JavaScript. En el caso de este proyecto, JavaScript permitió interactuar con información sin necesidad de generar una petición al servidor realizando las acciones necesarias desde el lado del cliente. La desventaja de usar este lenguaje es que el usuario puede deshabilitar JavaScript desde su navegador evitando cumplir con las funcionalidades definidas.

- Para enviar información generada del lado del cliente fue necesario el uso de AJAX, esta técnica permitió el envío asíncrono de información al servidor de tal forma que no sea necesario recargar toda la página sino solo interactuar con una parte de la misma.

- Para poder presentar información al usuario sin que este tenga que cargar otras páginas web se utilizó ventanas modales y AJAX. Esto permite generar una ventana que se presenta sobre la página principal, la cual da la posibilidad al usuario de interactuar con datos que se cargarán sobre la página principal, evitando cargar otras páginas y salir de la página actual.

- Para poder transmitir los datos entre cliente y servidor se puede usar el formato JSON, este es un formato estándar que permite la comunicación entre ambos lados evitando inconvenientes al transportar información entre lenguajes diferentes como JavaScript y C#.

- En el desarrollo de las interfaces de usuario definidas a través de vistas se empleó el *framework* Bootstrap, este *framework* dispone de un conjunto de estilos propios que permiten aplicar un formato específico a cada vista sin necesidad de tener que escribir código CSS.

4.2 RECOMENDACIONES

- Para casos en los que una aplicación web precise una interacción dinámica entre el usuario y la información presentada en la página web, se recomienda utilizar JavaScript. El desarrollo de funcionalidades a través de este lenguaje evita realizar peticiones al servidor para cargar páginas adicionales.

- Para el envío de correos electrónicos en la presente aplicación se utilizó la plataforma SendGrid. Esta plataforma al momento de implementarla en la aplicación no tenía ningún costo, sin embargo, para futuros proyectos se

recomienda el uso de otras soluciones que permitan brindar este servicio, ya que actualmente no existen planes gratuitos.

- El desarrollo de aplicaciones web a través de ASP.NET permite escoger varias opciones de autenticación al crear el correspondiente proyecto, se recomienda utilizar la opción de *Individual User Accounts*, la cual implementa el *framework* ASP.NET Identity. Este *framework* permite gestionar usuarios y roles para cada usuario, y además manejar un sistema de autenticación con varias características.

- Dentro del desarrollo de aplicaciones para gestión universitaria es muy común que se necesite generar archivos para presentar cierto tipo de información, por tal razón se recomienda el uso del paquete iTextSharp para la creación de archivos PDF, este paquete permitirá definir la información necesaria a través de los objetos definidos en el paquete.

- Existen varios paquetes que permiten crear y personalizar un correo electrónico definiendo los elementos pertenecientes al mismo, sin embargo, para proyectos desarrollados mediante el *framework* ASP.NET MVC se recomienda el uso del paquete Postal ya que este paquete permite una mayor personalización de los correos electrónicos al definir la información necesaria usando vistas.

- Se recomienda implementar la presente aplicación web dentro de la Carrera de Ingeniería Electrónica y Redes de Información para aprovechar las funcionalidades que esta aplicación presenta como la notificación a estudiantes y docentes a través del envío de correos electrónicos, o la evaluación de un plan de Trabajo de Titulación mediante el uso de rúbricas.

- Se realizaron pruebas de compatibilidad sobre dos de los navegadores más utilizados por usuarios de Internet: Google Chrome y Mozilla Firefox, sin embargo, se recomienda realizar dichas pruebas sobre otros

navegadores para comprobar el correcto funcionamiento de la aplicación, sobre todo por las funcionalidades definidas mediante JavaScript.

REFERENCIAS BIBLIOGRÁFICAS

[1] «Portal de la Junta de Andalucía,» [En línea]. Disponible en: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122. [Último acceso: 20 Junio 2016].

[2] G. Heileman, «MOOC Web Application Architectures,» University of New Mexico, 2013.

[3] R. Williams, «Code Project,» octubre 2013. [En línea]. Disponible en: http://www.codeproject.com/Articles/665118/Web-Forms-MVC-and-Web-Pages-Oh-my. [Último acceso: 14 julio 2016].

[4] J. Galloway y C. Harrison, «Microsoft Virtual Academy,» Microsoft, agosto 2014. [En línea]. Disponible en: https://mva.microsoft.com/en-US/training-courses/introduction-to-asp-net-mvc-8322?l=4fo2g9Zy_1404984382. [Último acceso: 20 junio 2016].

[5] «Programacion.net,» [En línea]. Disponible en: http://programacion.net/articulo/motores_de_persistencia_231#joa_persistencia_motores. [Último acceso: 13 julio 2016].

[6] «Microsoft Developer Network,» [En línea]. Disponible en: https://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf. [Último acceso: 13 julio 2016].

[7] «Oracle,» [En línea]. Disponible en: http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/dotnet/CodeFirst/index.htmlMétodo. [Último acceso: 18 julio 2016].

[8] «W3Schools,» [En línea]. Disponible en: https://www.w3schools.com/aspnet/razor_syntax.asp. [Último acceso: 14 julio 2016].

[9] K. Mikoluk, diciembre 2013. [En línea]. Disponible en: https://blog.udemy.com/jquery-vs-javascript-2-cual-es-la-diferencia-en-

definitiva/. [Último acceso: 14 julio 2016].

[10] «Solvetic,» enero 2015. [En línea]. Disponible en: http://www.solvetic.com/tutoriales/article/1393-entendiendo-ajax-con-jquery/. [Último acceso: 15 julio 2016].

[11] «W3Schools,» [En línea]. Disponible en: http://www.w3schools.com/js/js_json_objects.asp. [Último acceso: 5 enero 2017].

[12] D. Caballero, «SlideShare,» diciembre 2013. [En línea]. Disponible en: https://es.slideshare.net/lechuck2010/scrum-kanban-xp. [Último acceso: 17 mayo 2017].

[13] «Manifiesto por el Desarrollo Ágil de Software,» [En línea]. Disponible en: http://www.agilemanifesto.org/iso/es/manifesto.html. [Último acceso: 23 febrero 2016].

[14] «Manifiesto por el Desarrollo Ágil de Software,» [En línea]. Disponible en: http://www.agilemanifesto.org/iso/es/principles.html. [Último acceso: 23 febrero 2016].

[15] R. Rabanal, «Software Guru,» febrero 2013. [En línea]. Disponible en: http://sg.com.mx/sgvirtual/7/sessions/metodo-kanban-ingenieria-software. [Último acceso: 23 febrero 2016].

[16] «PMOinformatica.com,» octubre 2012. [En línea]. Disponible en: http://www.pmoinformatica.com/2012/10/plantillas-scrum-historias-de-usuario.html. [Último acceso: 24 febrero 2016].

[17] A. García, «javiergarzas.com,» julio 2014. [En línea]. Disponible en: http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html. [Último acceso: 20 julio 2016].

[18] «SendGrid,» [En línea]. Disponible en: https://sendgrid.com/. [Último acceso: 23 febrero 2017].

[19] «Microsoft Azure,» enero 2016. [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/app-service-web/sendgrid-dotnet-how-to-send-email. [Último acceso: 23 febrero 2017].

[20] «Postal,» [En línea]. Disponible en: http://aboutcode.net/postal/. [Último acceso: 3 marzo 2017].

[21] «iTextSharp,» [En línea]. Disponible en: https://sourceforge.net/projects/itextsharp/. [Último acceso: 3 marzo 2017].

ANEXOS

ANEXO A: Backlog

ANEXO B: Historias de Usuario

ANEXO C: Manual de Usuario

ANEXO D: Código

ANEXO A. BACKLOG

Identificador (ID) de la Historia	Enunciado de la Historia	Estado
HU01_01_USUARIOS	Como administrador del sistema necesito gestionar usuarios (crear, editar, mostrar y eliminar) para permitir el ingreso de los mismos al sistema.	Realizado
HU01_02_ROLES	Como administrador del sistema necesito gestionar roles para asignar a los usuarios un determinado rol y discriminar el acceso a cierta información en la aplicación según el rol establecido.	Realizado
HU01_03_AUTENTICACION	Como usuario del sistema necesito digitar mi correo electrónico y mi contraseña para poder autenticarme e ingresar al sistema.	Realizado
HU01_04_OLVIDO_CONTRASEÑA	Como usuario del sistema necesito poder generar una nueva contraseña a través de un enlace enviado vía correo electrónico para poder ingresar al sistema con la nueva contraseña generada.	Realizado
Identificador (ID) de la Historia	Enunciado de la Historia	Estado
HU02_01_01_CREAR_ESTUDIANTE	Como secretaria necesito ingresar los datos: Cédula, Número Único (opcional), Nombres, Apellidos, Carrera, Número de Teléfono y Correo Electrónico para poder crear un nuevo registro de estudiante.	Realizado
HU02_01_02_EDITAR_ESTUDIANTE	Como secretaria necesito poder editar los campos del registro estudiante para actualizar y corregir información errónea.	Realizado
HU02_01_03_BUSCAR_ESTUDIANTE	Como secretaria necesito poder buscar un registro estudiante con el apellido como parámetro de búsqueda para visualizar la información de dicho registro.	Realizado
HU02_02_01_CREAR_DOCENTE	Como secretaria necesito ingresar los datos: Cédula, Nombres, Apellidos y Correo Electrónico para poder crear un nuevo registro de docente.	Realizado
HU02_02_02_EDITAR_DOCENTE	Como secretaria necesito poder editar los campos del registro docente para actualizar y corregir información errónea.	Realizado
HU02_02_03_BUSCAR_DOCENTE	Como secretaria necesito poder buscar un registro docente con el apellido como parámetro de búsqueda para visualizar la información de dicho registro.	Realizado

HU02_03_01_CREAR_PTT	Como secretaria necesito ingresar los datos: Título, Tipo de Trabajo de Titulación, Línea de Investigación para poder crear un nuevo registro de Plan de Trabajo de Titulación (PTT).	Realizado
HU02_03_02_PDF_PTT	Como secretaria necesito cargar un archivo PDF que contenga el plan escaneado para registrar un nuevo PTT y poder enviar el plan a los miembros de la Comisión posteriormente.	Realizado
HU02_03_03_ESTUDIANTE_PTT	Como secretaria necesito buscar y elegir la información de uno a cuatro estudiantes para incluir esta información durante el registro de un PTT.	Realizado
HU02_03_04_DOCENTE_PTT	Como secretaria necesito elegir la información de un docente para asignar un director, y opcionalmente de otro más para asignar un codirector, para incluir esta información durante el registro de un PTT.	Realizado
HU02_03_05_EDITAR_PTT	Como secretaria necesito poder editar los campos del registro de un PTT para actualizar y corregir información errónea.	Realizado
HU02_03_06_BUSCAR_PTT	Como secretaria necesito poder buscar uno o más registros de un PTT usando diferentes filtros para visualizar la información de dichos registros.	Realizado
HU02_03_07_BUSCAR_PLAN_ESTUDIANTE	Como secretaria necesito poder buscar los PTT que ha presentado un estudiante usando el apellido del mismo como parámetro de búsqueda para visualizar dichos registros.	Realizado
HU02_03_08_BUSCAR_PLAN_DOCENTE	Como secretaria necesito poder buscar los PTT que ha dirigido o codirigido un docente, usando el apellido del mismo como parámetro de búsqueda, para visualizar dichos registros.	Realizado
HU02_03_09_BUSCAR_HISTORIAL_PLAN	Como secretaria necesito poder buscar el historial de evaluación de uno o más PTT para poder visualizar dichos registros.	Realizado
HU02_04_DESACTIVAR	Como secretaria necesito poder especificar el estado de los registros estudiante, docente o PTT, ya sea este inactivo o activo para evitar eliminar registros y solo mantener la información desactivada en caso de ya no ser necesaria.	Realizado
HU02_05_CAMPOS_CONFIGURABLES	Como administrador necesito poder crear, actualizar y desactivar los campos: Tipo de Trabajo de Titulación, Línea de Investigación y Carrera (Estudiante) para poder gestionar estos datos en caso de que cambie en un futuro dicha información.	Realizado

HU02_06_ANULACION_PTT	Como secretaria necesito poder registrar la anulación de un PTT por parte de un estudiante para poder cambiar el estado de un PTT a: Anulado.	Realizado
Identificador (ID) de la Historia	Enunciado de la Historia	Estado
HU03_01_CUERPO_CORREO	Como administrador necesito poder cambiar el texto de los cuerpos de los correos electrónicos para personalizar dicha información.	Realizado
HU03_02_CORREO_INGRESO	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar a estos del ingreso del PTT en el sistema.	Realizado
HU03_03_CORREO_MIEMBRO	Como secretaria necesito enviar un correo electrónico con el PTT adjunto en formato PDF a los miembros de la comisión para que estos cuenten con información del plan previamente a la evaluación del mismo.	Realizado
HU03_04_CORREO_OBSERVACIONES	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar los cambios que debe hacer al PTT para poder aprobarlo.	Realizado
HU03_05_CORREO_NEGACION_PTT	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar que el PTT ha sido negado por la comisión.	Realizado
HU03_06_CORREO_APROBACION_PTT	Como secretaria necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación para informar que el PTT ha sido aprobado por la comisión.	Realizado
Identificador (ID) de la Historia	Enunciado de la Historia	Estado
HU04_01_RUBRICA_CREACION	Como administrador necesito crear una rúbrica en donde se definan criterios de evaluación y el valor máximo que se le puede asignar a cada criterio para poder evaluar un plan.	Realizado
HU04_02_RUBRICA_PORCENTAJES	Como administrador debo asignar en la rúbrica creada un porcentaje de valoración de criterios para los estados de evaluación: negado, pendiente por aprobar y aprobado; para poder determinar el estado del plan luego de la evaluación.	Realizado
HU04_03_RUBRICA_EVALUACION	Como administrador necesito usar una rúbrica en donde se dé una valoración a cada criterio y una observación (de ser necesario) para evaluar un determinado plan de Trabajo de Titulación.	Realizado
HU04_04_RUBRICA_PDF	Como secretaria necesito generar un archivo PDF que contenga la rúbrica del plan evaluado para poder notificar al estudiante vía correo electrónico de las observaciones a su plan.	Realizado

ANEXO B. HISTORIAS DE USUARIO

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU01_01_USUARIOS	Como administrador del sistema	necesito gestionar usuarios (crear, editar, mostrar y eliminar)	para permitir el ingreso de los mismos al sistema.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU01_03_AUTENTICACION	Como usuario del sistema	necesito digitar mi correo electrónico y mi contraseña	para poder autenticarme e ingresar al sistema.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU01_02_ROLES	Como administrador del sistema	necesito gestionar roles	para asignar a los usuarios un determinado rol y discriminar el acceso a cierta información en la aplicación según el rol establecido.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU01_04_OLVIDO_CONTRASEÑA	Como usuario del sistema	necesito poder generar una nueva contraseña a través de un enlace enviado vía correo electrónico	para poder ingresar al sistema con la nueva contraseña generada.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_01_01_CREAR_ESTUDIANTE	Como secretaria	necesito ingresar los datos: Cédula, Número Único (opcional), Nombres, Apellidos, Carrera, Número de Teléfono y Correo Electrónico	para poder crear un nuevo registro de estudiante.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_01_02_EDITAR_ESTUDIANTE	Como secretaria	necesito poder editar los campos del registro estudiante	para actualizar y corregir información errónea.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_01_03_BUSCAR_ESTUDIANTE	Como secretaria	necesito poder buscar un registro estudiante con el apellido como parámetro de búsqueda	para visualizar la información de dicho registro.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_02_01_CREAR_DOCENTE	Como secretaria	necesito ingresar los datos: Cédula, Nombres, Apellidos y Correo Electrónico	para poder crear un nuevo registro de docente.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_02_02_EDITAR_DOCENTE	Como secretaria	necesito poder editar los campos del registro estudiante	para actualizar y corregir información errónea.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_02_03_BUSCAR_DOCENTE	Como secretaria	necesito poder buscar un registro docente con el apellido como parámetro de búsqueda	para visualizar la información de dicho registro.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_01_CREAR_PTT	Como secretaria	necesito ingresar los datos: Título, Tipo de Trabajo de Titulación, Línea de Investigación	para poder crear un nuevo registro de Plan de Trabajo de Titulación (PTT).

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_02_PDF_PTT	Como secretaria	necesito cargar un archivo PDF que contenga el plan escaneado	para registrar un nuevo PTT y poder enviar el plan a los miembros de la Comisión posteriormente.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_03_ESTUDIANTE_PTT	Como secretaria	necesito buscar y elegir la información de uno a cuatro estudiantes	para incluir esta información durante el registro de un PTT.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_04_DOCENTE_PTT	Como secretaria	necesito elegir la información de un docente para asignar un director, y opcionalmente de otro más para asignar un codirector	para incluir esta información durante el registro de un PTT.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_05_EDITAR_PTT	Como secretaria	necesito poder editar los campos del registro PTT	para actualizar y corregir información errónea.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_06_BUSCAR_PTT	Como secretaria	necesito poder buscar uno o más registros de un PTT usando diferentes filtros	para visualizar la información de dichos registros.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_07_BUSCAR_PLAN_ESTUDIANTE	Como secretaria	necesito poder buscar los PTT que ha realizado un estudiante usando el apellido del mismo como parámetro de búsqueda	para visualizar dichos registros.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_08_BUSCAR_PLAN_DOCENTE	Como secretaria	necesito poder buscar los PTT que ha dirigido o codirigido un docente, usando el apellido del mismo como parámetro de búsqueda	para visualizar dichos registros.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_03_09_BUSCAR_HISTORIAL_PLAN	Como secretaria	necesito poder buscar el historial de evaluación de uno o más PTT	para visualizar dichos registros.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_04_DESACTIVAR	Como secretaria	necesito poder especificar el estado de los registros estudiante, docente o PTT, ya sea este inactivo o activo	para evitar eliminar registros y solo mantener la información desactivada en caso de ya no ser necesaria.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_05_CAMPOS_CONFIGURABLES	Como administrador	necesito poder crear, actualizary desactivar los campos: Tipo de Trabajo de Titulación, Línea de Investigación y Carrera (Estudiante)	para poder gestionar estos datos en caso de que cambie en un futuro dicha información.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU02_06_ANULACION	Como secretaria	necesito poder registrar la anulación de un PTT por parte de un estudiante	para poder cambiar el estado de un PTT a: Anulado.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_01_CUERPO_CORREO	Como administrador	necesito poder cambiar el texto de los cuerpos de los correos electrónicos	para personalizar dicha información.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_02_CORREO_INGRESO	Como secretaria	necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación	para informar a estos del ingreso del PTT en el sistema.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_03_CORREO_MIEMBRO	Como secretaria	necesito enviar un correo electrónico con el PTT adjunto en formato PDF a los miembros de la comisión	para que estos cuenten con información del plan previamente a la evaluación del mismo.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_04_CORREO_OBSERVACIONES	Como secretaria	necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación	para informar de los cambios que debe hacer a su PTT para poder aprobarlo.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_05_CORREO_NEGACION_PTT	Como secretaria	necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación	para informar que el PTT ha sido negado por la comisión.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU03_06_CORREO_APROBACION_PTT	Como secretaria	necesito enviar un correo electrónico al estudiante y director/codirector del Trabajo de Titulación	para informar que el PTT ha sido aprobado por la comisión.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU04_01_RUBRICA_CREACION	Como administrador	necesito crear una rúbrica en donde se definan criterios de evaluación y el valor máximo que se le puede asignar a cada criterio	para poder evaluar un plan.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU04_02_RUBRICA_PORCENTAJES	Como administrador	asignar en la rúbrica creada un porcentaje de valoración de criterios para los estados de evaluación: negado, pendiente por aprobar y aprobado	para poder determinar el estado del plan luego de la evaluación.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU04_03_RUBRICA_EVALUACION	Como administrador	necesito usar una rúbrica en donde se dé una valoración a cada criterio y una observación (de ser necesario)	para evaluar un determinado plan de Trabajo de Titulación.

	Enunciado de la Historia		
Identificador (ID) de la Historia	Rol	Característica / Funcionalidad	Razón / Resultado
HU04_04_RUBRICA_PDF	Como secretaria	necesito generar un archivo PDF que contenga la rúbrica del plan evaluado	para poder notificar al estudiante vía correo electrónico de las observaciones a su plan.

ANEXO C. MANUAL DE USUARIO

El manual de usuario se encuentra en el CD adjunto.

ANEXO D. CÓDIGO

El código de la aplicación se encuentra en el CD adjunto.