UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
Ingenieros de Sistemas

TEMA:
"REFACTORIZACIÓN DE LA APLICACIÓN WEB REGISTRO Y SEGUIMIENTO DE OPCIONES DE TITULACIÓN CON MIRAS A LA CREACIÓN DE UNA SUITE DE HERRAMIENTAS INFORMÁTICAS DE APOYO A LA GESTIÓN DE LA CARRERA DE INGENIERÍA DE SISTEMAS"

AUTORES:
DIEGO ALEJANDRO ARIAS PALAQUIBAY
ROBERTH ALEXANDER PROAÑO CHIRIBOGA

TUTOR:
FRANKLIN EDMUNDO HURTADO LARREA

Quito, marzo del 2021

CESIÓN DE DERECHOS DE AUTOR

Nosotros, Diego Alejandro Arias Palaquibay con documento de identidad N°.1727479348 y Roberth Alexander Proaño Chiriboga, con documento de identidad N°.1721050506, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación intitulado: "**REFACTORIZACIÓN DE LA APLICACIÓN WEB REGISTRO Y SEGUIMIENTO DE OPCIONES DE TITULACIÓN CON MIRAS A LA CREACIÓN DE UNA SUITE DE HERRAMIENTAS INFORMÁTICAS DE APOYO A LA GESTIÓN DE LA CARRERA DE INGENIERÍA DE SISTEMAS"**, mismo que ha sido desarrollado para optar por el título de: INGENIEROS DE SISTEMAS, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, subscribimos este documento en el momento que hacemos entrega del trabajo final en digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, marzo del 2021

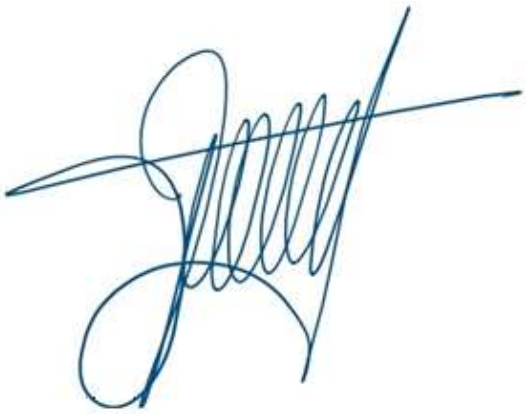
……
DIEGO ALEJANDRO
ARIAS PALAQUIBAY
C.I: 1724479348

……
ROBERTH ALEXANDER
PROAÑO CHIRIBOGA
C.I: 1721050506

DECLARATORIA DE COAUTORÍA DEL TUTOR

Yo, declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación con el tema: "**REFACTORIZACIÓN DE LA APLICACIÓN WEB REGISTRO Y SEGUIMIENTO DE OPCIONES DE TITULACIÓN CON MIRAS A LA CREACIÓN DE UNA SUITE DE HERRAMIENTAS INFORMÁTICAS DE APOYO A LA GESTIÓN DE LA CARRERA DE INGENIERÍA DE SISTEMAS"** realizado por Diego Alejandro Arias Palaquibay y Roberth Alexander Proaño Chiriboga, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerado como trabajo final de titulación.

Quito, marzo del 2021



……
Franklin Edmundo Hurtado Larrea
C.I: 1713382016

AGRADECIMIENTOS

A nuestras familias por apoyarnos en cada momento de nuestra vida y en especial en esta etapa universitaria, que con sus consejos supieron guiarnos para cumplir este logro tan importante dentro nuestra carrera profesional.

Agradecemos especialmente a nuestro tutor de proyecto de titulación, Ing. Franklin Hurtado, quien supo orientarnos y guiarnos a lo largo de este proceso, quien con su ayuda hoy culminamos una etapa más de nuestra vida.

Diego Alejandro Arias Palaquibay

Roberth Alexander Proaño Chiriboga

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En el presente documento se detalla el proceso de refactorización de la aplicación web Registro y Seguimiento de Opciones de Titulación, para que se ajuste a nuevos requerimientos de la carrera de Ingeniería de Sistemas, al ejecutar este proceso la aplicación refactorizada adquirió nuevas características que facilitan la integración y la interoperabilidad siendo estas la mantenibilidad, flexibilidad y la escalabilidad.

Además, se detalla las modificaciones que se implementaron en la arquitectura para lograr que la aplicación refactorizada adquiera mayor capacidad de integración.

En los capítulos que se desarrolla a continuación se expondrá todo el proceso que se siguió para llevar a cabo la refactorización de la aplicación web Registro y Seguimiento de Opciones de Titulación, además se evidenciará las modificaciones realizadas en el software y las pruebas respectivas que se ejecutaron al mismo para comprobar su correcto funcionamiento.

ABSTRACT

This document details the refactoring process of the web application Registration and Monitoring of Degree Options, in order to adjust to new requirements of the Systems Engineering career. By executing this process, the refactored application acquired new features that facilitate integration and interoperability, being these the maintainability, flexibility and scalability.

It also details the modifications that were implemented in the architecture to achieve that the refactored application acquires greater integration capacity.

In the chapters that follow, the whole process followed to carry out the refactoring of the web application Registration and Monitoring of Titling Options will be explained, as well as the modifications made to the software and the respective tests that were executed to verify its correct functionality.

INTRODUCCIÓN

Con el avance de la tecnología el mundo cambia cada vez más rápido, se observa tanto en los productos, servicios y herramientas que se utiliza a diario para hacer más fácil la vida de las personas, y el software no es la excepción.

Las aplicaciones web eran creadas en forma monolíticas ya que, no existía una separación entre el back-end y el front-end generando un software robusto pero complicado de mantener y difícil de realizar cambios una vez puesto en producción.

Con las nuevas herramientas y tecnologías que fueron surgiendo para desarrollar software, se planteó el desarrollo en capas dividiendo o segmentando la capa de interfaz de usuarios de la capa de acceso a datos más conocido como front-end y back-end en la actualidad.

Al transcurrir el tiempo la aplicación web Registro y Seguimiento de Opciones de Titulación, ya no se acoplaba a los requerimientos actuales haciendo complicado su mantenimiento e integración.

La refactorización es una de las técnicas más adecuada para realizar cambios en una aplicación ya que, permite reestructurarlo sin modificar su estructura lógica y funcionalidades.

"La refactorización es el proceso de cambiar un sistema de software de tal manera que no altera el comportamiento externo del código, pero mejora su estructura interna" (Fowler, 2018, p.13).

Antecedentes

La carrera de Ingeniería de Sistemas de la Universidad Politécnica Salesiana cuenta con diversas aplicaciones web que fueron creadas para manejar información relevante para la institución y para los usuarios (Docentes/ Estudiantes), estas aplicaciones web han sido desarrolladas con el fin de apoyar en la gestión de las áreas más importantes de carrera.

Siendo una de estas la aplicación web Registro y Seguimiento de Opciones de Titulación la cual ya no se ajusta a los nuevos requerimientos de los usuarios por lo cual, se requiere que esta

aplicación adquiera nuevas características como la integración, mantenibilidad y escalabilidad por los que debe mejorar entrar en un proceso de refactorización para mejorar las características antes mencionadas sin cambiar la funcionalidad de la aplicación.

Problemática

La carrera de Ingeniería de Sistemas de la Universidad Politécnica Salesiana cuenta con diversas aplicaciones web que fueron creadas para manejar información relevante para la institución y para los usuarios (Docentes/ Estudiantes), estas aplicaciones web han sido desarrolladas con el fin de apoyar en la gestión de las áreas más importantes de carrera.

Dichas aplicaciones fueron creadas en diferentes lenguajes de programación y patrones arquitectónicos, esto debido a que cuando se crean aplicaciones por separado, no necesariamente se toma en cuenta los requerimientos a futuro, siendo la integración con otras aplicaciones, un requerimiento importante y que la mayoría de veces se pasa por alto.

En la actualidad, por las características de la gestión de las organizaciones, tarde o temprano, será necesario que las aplicaciones compartan información, esto quiere decir que a nivel tecnológico estas deben ser integradas, siendo el caso de la Universidad Politécnica Salesiana campos Sur en la Carrera Ingeniería de Sistemas.

Entonces, debido a que, como se ha mencionado, se busca generar una suite de herramientas para la carrera, la aplicación Registro y Seguimiento de Opciones de Titulación debe ser integrada a otras soluciones de software, con lo que se hace necesario la refactorización de ésta.

Justificación

Para otorgar las características de mantenibilidad e integración a la aplicación web Registro y Seguimiento de Opciones de Titulación, se procedió a realizar la refactorización de dicha aplicación ya que, esta fue desarrollada en forma monolítica complicando su modificación o mejoras al código.

Con esto se segmentó el código dividiendo la capa de interfaz de usuario (front-end) y la capa de acceso a datos (back-end), al segmentarlo se podrá reestructurar el código sin alterar su funcionamiento y lógica.

Este procedimiento no se empleó con el fin de corregir errores sino más bien con el objetivo de llevar a la aplicación a un proceso de reestructuración para agregar nuevas funcionalidades como de mantenibilidad e integración, esto sin alterar su comportamiento.

Por este motivo, se requiere preparar a esta aplicación con un esquema que facilite la integración y mantenibilidad, ejecutando una refactorización de la aplicación web Registro y Seguimiento de Opciones de Titulación para repotenciarla con lo cual se pretende restructurar el acceso a la información de forma óptima y eficaz, dando así una mejor experiencia al usuario.

Por ello el proposito de este proyecto, es la refactorizacion, para dotar a la aplicación de una arquitectura, de modo que esté preparada y sea apta para realizar una integracion a futuro.

Este proyecto es factible ponerlo en la fase de desarrollo ya que, existe una factibilidad técnica y operativa por lo que, al momento se cuenta con las herramientas tecnológicas y un equipo capaz de llevarlo a cabo.

Objetivos

Objetivo General

Refactorizar la aplicación de Registro y Seguimiento de Opciones de Titulación para facilitar su integración con otras aplicaciones, en el marco de la creación de una suite de herramientas informáticas para la carrera de Ingeniería de Sistemas.

Objetivos específicos

Analizar los estilos, funcionalidades, arquitectura y estructura de datos de la aplicación “Registro y Seguimiento de Opciones de Titulación”, para realizar la refactorización de la misma.

Establecer un estilo de diseño (CSS), que pueda ser aplicado a la refactorización de esta aplicación y a las futura aplicaciones que se pretendan integrar, para formar una suite de herramientas informáticas.

Crear el portal y las funcionalidades unificadas para el ingreso de la suite, en base al estilo establecido.

Realizar la refactorización integral (aspectos visuales y arquitectónicos) de la aplicación Registro y Seguimiento de Opciones de Titulación.

Diseñar y ejecutar pruebas que permitan validar la refactorización con miras en la creación de la suite de herramientas informáticas para la carrera.

Marco Metodológico

Para desarrollar el proceso de refactorización de la aplicación antes mencionada se utilizó el marco de trabajo Scrum

Al implementar dicho marco de trabajo se organizó y designó a los miembros del equipo los roles y funciones estableciendo las actividades a cumplir tal como, se detalla a continuación:

Scrum Master: Más conocido como facilitador, su función es liderar y eliminar los obstáculos que impiden que el equipo cumpla con su objetivo.

- Tutor del proyecto

Dueño del producto (Product owner): Se asegura de que el proyecto se esté desarrollando acorde con la estrategia del negocio. Escribe historias de usuario, las prioriza, y las coloca en el Product Backlog.

- Tutor del proyecto: el tutor del proyecto es el dueño del producto ya que, es una de las personas más importantes de la Unidad de Titulación siendo el responsable de que el aplicativo refactorizado alcance los resultados deseados.

Equipo de desarrollo: es la persona o grupo de personas que tienen como objetivo principal el desarrollar el producto o software, siendo este equipo el responsable de que el producto esté listo con los requerimientos solicitados.

Este grupo se encargará de ejecutar la refactorización del aplicativo antes mencionado.

- Diego Arias y Roberth Proaño

Al implementar el marco de trabajo antes mencionado se debe establecer el sprint backlog (Anexo A), para determinar tiempos y tener un seguimiento constante de todas las tareas a realizar por lo que, se empleó la herramienta JIRA ya que, dicha herramienta se ajustó a los requerimientos y necesidades del proyecto.

Esta herramienta facilitó la creación de las actividades que se llevó a cabo en el desarrollo de toda la aplicación, permitiendo listar todas las tareas que se debían hacer en la parte del backlog, como se muestra en la figura 1.

Spring backlog

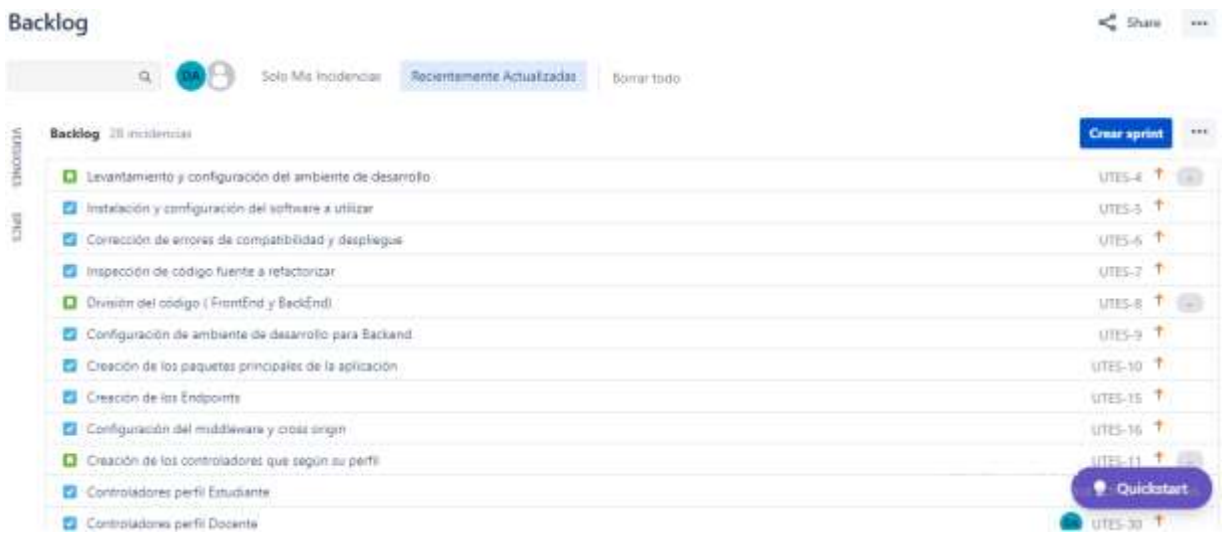


Figura 1. Spring backlog.
Elaborado por: Diego Arias y Roberth Proaño

En la figura 1 se muestra la parte inicial del sprint backlog en donde figuran los primeros módulos del software que se elaboraron.

Al utilizar la herramienta JIRA esta tiene una ventaja, que ayuda el programador a visualizar todas las tareas y el estado de cada una de ellas, también conocido como tablero Kanban como se muestra en la figura 2, este tablero se encuentra dividido en tres secciones que son: por hacer, en curso y listo, se pueden ir ubicando cada una de las tareas para ir comprobando su avance y así optimizar el tiempo al no ejecutar una tarea más de una vez.

Tablero Kanban

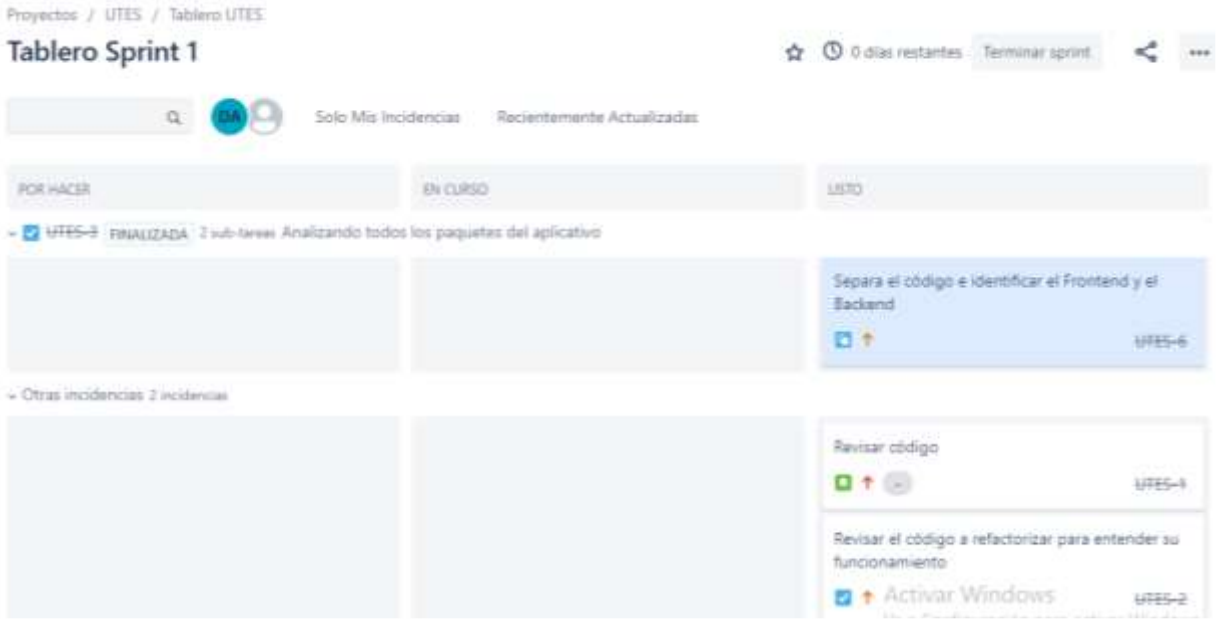


Figura 2. Tablero Kanban.
Elaborado por: Diego Arias y Roberth Proaño

En la figura 2 se visualiza el tablero Kanban, en el cual constan las tareas planificadas divididas en tres fases, de esta manera se logró tener un control más estricto de las tareas y del estado en que estas se encontraban y así visualizar el avance de cada tarea.

A continuación, se detallará de manera breve los sprints generados en el desarrollo del aplicativo refactorizado.

Tabla 1. Spring ejecutados

Sprints	Descripción
1	- Instalación y configuración de los frameworks y herramientas a utilizar, - Corrección de errores de compatibilidad y despliegue. - Ejecución del aplicativo base para compresión de su funcionamiento. - Inspección del código fuente
2	- Configuración del ambiente de desarrollo para el back-end. - Creación y configuración de paquetes principales del aplicativo.
3	- Creación de los modelos y entidades del aplicativo - Implementación del funcionamiento lógico (lógica del negocio) - Creación de endpoints - Creación de los RestController
4	- Creación de la estructura del front-end. - Creación de modelos y servicios - Configuración y visualización de pantalla de ingreso login. - Configuración y visualización de pantallas por perfil de usuario
5	- Restricciones y validaciones según el perfil del usuario. - Visualización de información según el perfil de usuario
6	- Configuración de las funcionalidades según el perfil del usuario
7	- Creación de reportes
8	- Corrección de errores y pruebas del aplicativo

Nota: Esta tabla contiene una breve descripción de los sprints ejecutados.

CAPÍTULO 1

MARCO REFERENCIAL Y TEÓRICO

1. Marco Referencial

La Universidad Politécnica Salesiana en el año 2014, crea la Unidad de Titulación Especial (UTE), al acogerse al nuevo reglamento del Consejo de Educación con resolución N° 173-09-2014-10-15 (Flores & Quisupangui, 2016).

La Unidad de Titulación Especial tras aplicar la resolución del Consejo Superior N° RPC-SE-13 051-2013 y acogerse al artículo 21 implementa las siguientes opciones de titulación Proyecto Técnico, Artículo Académico y Examen Complexivo (UNIVERSIDAD POLITÉCNICA SALESIANA, 2015).

En el año 2015 la Unidad de Titulación Especial cambia de nombre a Unidad de Titulación y se acoge al nuevo instructivo aprobado por el Consejo Superior con resolución N° 180-12-2015-11-10 (UNIVERSIDAD POLITÉCNICA SALESIANA, 2015).

En este instructivo se establece la estructura y proceso de desarrollo de los trabajos de titulación, otorgando los lineamientos a seguir para optar por una opción de titulación.

1.1. Marco Teórico.

1.1.1. Refactorización del Software. Con la refactorización de un software busca realizar la restructuración, reorganización y en muy pocos casos la eliminación de una cierta funcionalidad, logrando así la encapsulación y optimización del código fuente sin cambiar su funcionamiento o ejecución.

Las aplicaciones de software siempre están sujetas a procesos de transformación y/o mejoras para mantenerse actualizadas a los requerimientos cambiantes del ambiente en el que se desempeñan. Al realizar cambios, uno de los que riesgos es que decaiga su calidad, para lo cual se utilizan varias métricas que permiten evaluar la forma en que se verá afectada la calidad.

1.1.2. Refactorización con perspectiva al código. El principal objetivo que pretende la técnica de refactorización de código es suprimir código que no aporta valor o de baja calidad, con el fin de mejorar su interpretación, mantenimiento, estructura y diseño, incrementando así su desempeño (Franco, 2010).

Entre los principales objetivos del proceso de refactorización al código son:

- Eliminar código duplicado, innecesario o sin funcionalidad (código muerto).
- Código más fácil de comprender.
- Reducir clases o métodos.
- Mejorar la mantenibilidad, usabilidad.

1.1.3. Refactorización de con perspectiva en la arquitectura. "Al efectuar una refactorización a nivel arquitectónico se mejora su estructura sin que ello implique alterar el comportamiento externo del sistema" (Roldán, 2013, p. 40).

Al refactorizar la arquitectura de un software, ayuda al mismo a ser más versátil y poder adecuar así más componentes o servicios de una mejor forma sin tener tantas complicaciones.

1.2. Scrum. Scrum es un marco de trabajo que apoya al mantenimiento, mejora y gestión de un sistema recién creado o existente. Este marco de trabajo se centra en organizar los miembros del equipo con el fin de generar un sistema moldeable en un entorno cambiante (Caso, 2004).

Scrum es considerado como uno de los mejores marcos de trabajo al momento de desarrollar software, dicho marco de trabajo establece el procedimiento a seguir para que el proyecto este lo más organizado posible, asegurando agilidad y un producto de calidad.

1.2.1. Progreso del proyecto. Scrum se maneja mediante un proceso incremental en el desarrollo de proyectos

- Revisión de las Iteraciones

Scrum se maneja mediante iteraciones o sprint de corta duración que van desde quince días a un mes, al final de cada sprint se deberá presentar un entregable, si existiese algún inconveniente esto se debe notificar para que se trate y sea resuelto a tiempo para no afectar la duración del mismo.

- Autoorganización

Los equipos son capaces de organizarse por sí solos, no necesitan roles para la gestión (Gallego, 2012).

Scrum brinda una característica muy importante que es la capacidad de autoorganización del equipo para la toma de decisiones importantes siendo estas oportunas.

1.3. Diagrama BPMN. "Business Process Model and Notation (BPMN) es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio" (Nextech Education Center, 2020, p. 34).

Esta notación se caracteriza por facilitar la coordinación de secuencias entre los procesos y los mensajes que se intercambian entre los participantes de las diferentes actividades, siendo un lenguaje que expresa los procesos de forma eficiente, clara y detallada.

Los objetivos del BPMN son:

- Establecer un patrón de referencia que sea fácil y entendible para los involucrados diseñando un lenguaje común eliminando brechas de comunicación al momento de plantear el proceso del negocio y su implementación.
- Mejorar el dinamismo del negocio brindando una mayor competitividad y productividad a la organización, logrando adaptarse a un entorno cambiante a través de los procesos integrados reduciendo así tiempos y costos.

1.4. JIRA. “Jira Software está diseñado para que todos los miembros de tu equipo de software puedan planificar, supervisar y publicar un magnífico software” (Atlassian, 2020, p. 26).

Esta herramienta permite organizar y planificar las tareas de los proyectos de forma flexible, además de brindar seguimiento a todas las tareas e incidencias que se han producido, ayudando a gestionar de una mejor manera el proyecto en desarrollo aumentando la eficiencia y mejorando las estimaciones de tiempo de culminación.

Dicha herramienta fue acogida ya que, trabaja con los tableros Kanban y Scrum lo que, permite agilitar el proceso de desarrollo y el flujo de trabajo entre el equipo, a pesar de que no es la única herramienta, está se adaptó a las necesidades y requerimientos solicitados.

1.5. Swagger. “Swagger es un conjunto de herramientas que permite el desarrollo en todo el ciclo de vida de la API, desde el diseño y la documentación, hasta las pruebas y la implementación” (SmartBear, 2020, p. 02).

Es un framework que ayuda a documentar API’s REST, una de las ventajas de este framewok es que te permite integrar fácilmente diferentes lenguajes de programación, entre ellos: JavaScript, C#, PHP, Java, Etc.

Gracias a este framework se puede representar, elaborar, consumir y visualizar API’s, una de las características de este framework es que, ayuda a ensayar los servicios creados con el fin de evitar y corregir errores a tiempo y una vez que este funcionado correctamente documentarlos.

1.6. Arquitectura de software. "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución" (Hilliard, 2000, p. 11).

Al momento de dotar a un software de una arquitectura uno de los objetivos es alcanzar estabilidad y mantenibilidad de alto nivel logrando identificar las zonas estáticas y dinámicas con lo que, se

otorgara varias características importantes al software como la mantenibilidad, seguridad, reusabilidad, rendimiento, etc.

Dependiendo de la selección de una arquitectura esta afectara el desarrollo, comportamiento y la utilidad del sistema.

1.6.1. Arquitectura cliente/servidor. "El modelo Cliente/Servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes" (Marini, 2012, p. 28).

Dicho modelo permite fraccionar el trabajo que ejecuta cada aplicación, de forma que los clientes no se saturen por lo que, la gestión de la información y responsabilidades es más eficiente al repartir la carga y el esfuerzo que se va a tener al ejecutar o realizar una acción determinada.

1.6.2. Arquitectura orientada a los servicios. "Propone convertir los recursos de software en servicios disponibles que puedan ser reutilizables y fáciles de integrar en grandes y complejas aplicaciones, facilitando la interoperabilidad entre las aplicaciones y el web service" (Caicedo, 2008, p. 06).

1.7. NodeJs. "Concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para construir aplicaciones en red escalables" (NodeJs, 2020, p. 09).

Node.js implementa un modelo de eventos que se maneja como un entorno y no como una librería, facilitando la ejecución de scripts sin tener problemas de bloqueos y esperas innecesarias.

1.8. Spring Framework. "Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java" (VMware, 2020, p. 15).

Spring Framework es una herramienta que entre sus objetivos principales están agilizar, facilitar y simplificar la creación y desarrollo de aplicaciones que tengan como lenguaje de programación Java, Kotlin o Groovy. Dicho framework se caracteriza por:

- Reducir código al implementar inyección de dependencias en vez de crearlas nuevamente
- Modularidad
- Escalabilidad

1.9. PostgreSQL. "Es un potente sistema de base de datos relacional de objetos de código abierto que usa y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de manera segura las cargas de trabajo de datos más complicadas" (PostgreSQL, 2020, p. 05).

1.10. Angular. Es un framework desarrollado por Google de código abierto, que fue desarrollado con el objetivo de separar el front-end del back-end y crear aplicaciones web SPA (Single Page Application) (Angular.io, 2020).

Angular utiliza como lenguaje de programación TypeScript, así toda la sintaxis y el modo de codificar es el mismo, añadiendo coherencia y consistencia tanto al código como al proyecto. Entre las características principales se resalta:

- Separar el front-end del back-end
- Se maneja con el patrón arquitectónico MVC
- Desarrollo de aplicativos más eficaces y eficientes

1.10.1. Single page application (SPA). "Es una aplicación web o un sitio web que carga todos los recursos necesarios para navegar por el sitio en la primera carga de la página" (Developers, 2020, p. 12)**.**

Una SPA es una aplicación web la cual agrupa todo el contenido en una sola página, la característica principal de una SPA es que carga el contenido según se vaya utilizando el aplicativo con lo cual, reduce el tiempo de ejecución y agilita la navegación.

1.11. **Patrones de diseño.** "Un patrón de diseño proporciona un esquema para refinar componentes o subsistemas de un sistema de información o las relaciones entre ellos" (Díaz, 2004, p.23).

1.11.1. Patrones de diseño orientados a clases y objetos.

1.11.1.1. Patrones de diseño de creacionales. "Los patrones creacionales proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización del código existente" (Shvets, 2020, p.10).

Singleton (Instancia única)

"Singleton es un patrón de diseño creacional que permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia" (Shvets, 2020, p.10).

Builder (Constructor)

"Permite producir diferentes tipos y representaciones de un objeto utilizando el mismo código de construcción" (Profile, 2020, p. 12).

1.11.1.2. Patrones de comportamiento. "El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones" (Profile, 2020, p. 12).

Observer (Observador)

"Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando" (Shvets, 2020, p.10).

1.11.2. Patrones de diseño orientado a subsistemas de componentes. "Este tipo de patrones de diseño definen estructuras de componentes o sus relaciones, por ejemplo: patrones de diseño orientado al acceso de datos, dominio o presentación, como también definen la estructura y comportamiento de clases y objetos" (RJCodeAdvance, 2020, p. 11).

1.11.2.1. Patrones de diseño orientado a la capa de negocio. Estos patrones asumen la responsabilidad de manejar y controlar las transferencias de datos, validaciones y reglas de negocio.

DTO (Data Transfer Object)

"Son clases que sirven principalmente para el transporte de información entre todas las capas" (Cáceres Alvarez, 2010, p. 37).

1.11.2.2. Patrones orientados de integración o acceso a datos. Estos patrones se ocupan de explotar los datos de una base de datos o de cualquier fuente de archivos.

DAO

Desacopla la lógica de negocio de la lógica de acceso a datos, de modo que se pueda cambiar la fuente de datos fácilmente (Presedo Varela, 2010).

1.12. REST. "REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON" (Bbvaapimarket, 2020, p. 08).

Esta interfaz ayuda a comunicarse de una manera más ágil utilizando protocolos HTTP entre varios aplicativos o sistemas ya que, al utilizar un formato JSON la transferencia de datos e información es más óptima.

Características de REST

- Se maneja mediante el protocolo cliente/servidor albergando solo información necesaria para ejecutarla

- Maneja varias peticiones entre las principales: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- Maneja identificadores únicos que facilitan el acceso a la información.
- Al realizar peticiones al servidor esta devolverá una respuesta dependido la acción ejecutada entre las principales: 200 petición realizada correctamente, 201 petición de creación de nuevo recurso exitosa, etc.

1.13. GitHub. "GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código" (Kinsta, 2020, p. 30).

GitHub es una herramienta que permite tener repositorios en la nube. Esto permite centralizar y dinamizar el contenido de un proyecto o varios en un solo repositorio, y así realizara aportaciones desde diferentes lugares donde los colaboradores de una organización se encuentren.

1.14. Visual Paradigm Online (VP Online). Es una herramienta de dibujo en línea que ayuda y diseñar cualquier tipo de diagramas, para brindar soluciones de software y transformación empresarial (Visual Paradigm, 2020).

Esta herramienta facilita la creación y diseño de cualquier diagrama para entender o explicar cualquier tipo de información, entre los diagramas que esta herramienta brinda se detalla los siguientes: diagrama UML, diagrama BPMN y diagramas de arquitectura entre otras.

1.15. Apache JMeter. "La aplicación Apache JMeter es un software de código abierto, una aplicación Java 100% pura diseñada para cargar el comportamiento funcional de prueba y medir el rendimiento" (Apache Software Foundation, 2020, p. 07).

Uno de los usos de esta herramienta es comprobar el rendimiento de los recursos estáticos y dinámicos. Dicha herramienta es utilizada para poner a prueba el sitio web simulando peticiones al servidor para analizar el rendimiento general bajo diferentes tipos de carga.

1.16. Sirius. Se define como un sistema de evaluación de la usabilidad web, planteando una evaluación heurística que aplica una serie de aspectos y criterios establecidos para evaluar y detectar errores de usabilidad (Torrente, 2011).

A continuación, se describirán los frameworks utilizados al igual que las herramientas empleadas.

Tabla 2. Frameworks utilizados

Framework	Descripción
Angular	Se utilizó para desarrollar el front-end de la aplicación refactorizada
Spring Framework	Se utilizó para desarrollar el back-end de la aplicación refactorizada

Nota: Esta tabla contiene los frameworks empleados para el desarrollo de la aplicación refactorizada.

Tabla 3. Descripción de las herramientas empleadas

Herramienta	Función
Swagger	Se utilizó Swagger para documentar el API REST del back-end de la aplicación refactorizada ya que, esta herramienta facilitó describir, probar y consumir los servicios implementados en la aplicación refactorizada.
Apache JMeter	Se utilizó esta herramienta para realizar las pruebas de la aplicación refactorizada ya que, esta herramienta ayuda a poner a prueba las aplicaciones desarrolladas bajo diferentes escenarios.
Git Hub	Esta herramienta se utilizó para albergar el código desarrollado en la nube, permitiendo versionar el código, registrando los cambios que se crearon y guardando un historial del lugar y fecha de cuando se generó un nuevo cambio al código antes guardado.

Nota: Esta tabla contiene las herramientas utilizadas.

1.17. Pruebas del sistema. La ejecución de pruebas a un sistema tiene como principal objetivo exponer, conocer, entender, probar y comprobar, su correcto funcionamiento de forma general e integral, y si hubiera errores esta fase o etapa ayudara a identificar el lugar exacto donde

se está produciendo la falla o inconsistencia, corrigiendo y reduciendo así la cantidad de errores al mínimo.

1.17.1. Tipos de pruebas de software. Todos los tipos de pruebas de software que existen, básicamente, se agrupan de la siguiente manera: pruebas funcionales y pruebas no funcionales. Existen varios tipos de pruebas que se pueden aplicar a un sistema o software antes de salir a producción para verificar varias características que deben cumplir entre unas de las principales esta la calidad y eficiencia del software.

- Pruebas funcionales

  Las pruebas funcionales se centran en comprobar que el sistema desarrollado funcione acorde a las especificaciones requisitos solicitados por el cliente. Estas pruebas ayudan a detectar las posibles fallas procedentes de errores en la fase de programación.

  Una de ellas es las pruebas de aceptación cuyo objetivo principal es validar que un sistema cumpla con el funcionamiento solicitado por el usuario, el cual determinara su aceptación desde el punto de vista de usabilidad y funcionalidad.

- Pruebas no funcionales

  “Hacen referencia a las pruebas necesarias “para medir las características de los sistemas y software que pueden cuantificarse según una escala variable” (Paz, 2016, p. 13).

  Estas pruebas son orientadas y canalizadas hacia el comportamiento externo del aplicativo entre las que se menciona las pruebas de rendimiento y carga.

  Estas pruebas permiten determinar la respuesta de una aplicación cuando es puesta a prueba con una cantidad especifica de usuarios, transacciones o procesos simultáneamente, simulando un ambiente de producción.

CAPÍTULO 2

ANÁLISIS

Para el desarrollo de este documento se utilizó las siguientes abreviaturas para identificar las aplicaciones y el perfil de usuario:

- aplicación base: aplicación web para el Registro y Seguimiento de Opciones de Titulación.
- aplicación refactorizada: aplicación web para el Registro y Seguimiento de Opciones de Titulación refactorizada.
- CUTS: Coordinación de la Unidad de Titulación de Ingeniería de Sistemas.

2. Análisis de usuarios

Para empezar el proceso de refactorización se procedió a ejecutar el aplicativo base para identificar el número de usuarios que interactuaban en dicho aplicativo, logrando identificar los usuarios que se muestran en la figura 3.

Organizador de usuarios

Figura 3. Organizador de usuarios.
Elaborado por: Diego Arias y Roberth Proaño

Se implementó el organizador gráfico ya que, esta herramienta permitió estructurar la información de forma visual, ayudando a una mejor comprensión al registrar los usuarios que interactúan en el aplicativo base.

Tabla 4. Perfiles del aplicativo base

Usuarios	Acciones
Estudiante	• Realiza inscripción al proceso de titulación • Selección de opción a titulación • Visualización del listado de temas Proyectos técnicos y Artículos académicos • Registro de hitos
Docente	• Registra los temas para las opciones de titulación: Proyectos técnicos y Artículos académicos • Enviar temas a publicar para su aprobación • Asigna temas a/los estudiante/s • Registro de citas • Tareas para evolución de trabajo • Valida hitos ingresados por los estudiantes • Revisa temas asignados para la publicación
CUTS	• Registra periodos y convocatorias para inscripciones • Publicación de temas • Asignar revisores y lectores de plan y proyectos • Revisar temas de Proyectos técnicos y Artículos académicos • Publicar temas • Registrar resoluciones
Administrador	• Administra el sistema • Configuración del aplicativo para su correcto funcionamiento

Nota: Esta tabla contiene los usuarios y las acciones que puede ejecutar.

Como se muestra en la tabla 4, se identificó los usuarios que intervienen en el aplicativo base con lo cual, se determinó las acciones y funciones que cado uno de los usuarios realiza dentro del mismo.

2.1. Análisis del proceso global de Registro y Seguimiento de Opciones de Titulación. El proceso general se lo obtuvo del trabajo de titulación de Registro y Seguimiento de Opciones de Titulación ya que, para desarrollar el aplicativo antes mencionado se diseñó el proceso para su posterior automatización.

El proceso de Registro y Seguimiento de Opciones de Titulación no ha tenido ningún cambio, pero por fines académicos y didácticos se lo diagramó nuevamente obteniendo un diagrama más

específico para entender la lógica aplicada al negocio y comprender de una mejor manera el funcionamiento del aplicativo base como se muestra en la figura 4.

Proceso Global de Titulación

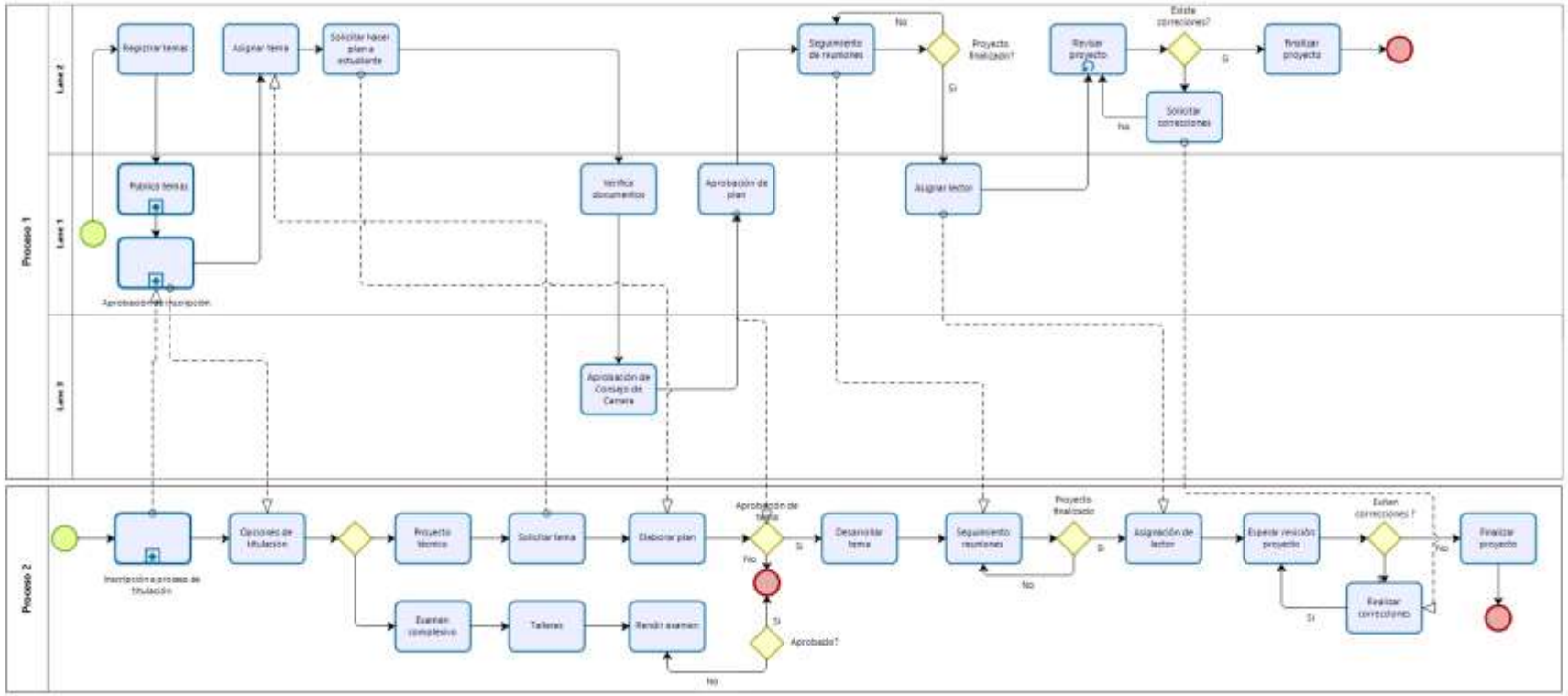


Figura 4. Proceso Global de Titulación
Fuente: Diego Arias y Roberth Proaño

2.1. Análisis de la aplicación base. En este apartado se describe todo el proceso de análisis que se ejecutó para comprender su funcionamiento, estructura del código fuente y lo más importante la lógica del negocio.

2.1.1. Descripción de la documentación de los casos de uso. A continuación, en la tabla 5 se define y realiza una breve descripción general de los casos de uso que fueron implementados en la aplicación base.

Tabla 5. Casos de uso aplicación base y su descripción general

Caso de Uso	Descripción
Registro de usuario	Proceso que el usuario sigue para registrarse en la aplicación
Publicación de temas	Proceso que realiza un docente para publicar un tema ya sea Proyecto técnico o Artículo académico.
Inscripción	Proceso que realiza el estudiante para inscribirse en la Unidad de Titulación.
Aprobación de tema con resolución	Proceso que realizan: docente, estudiante y la CUTS para asignación de tema a un estudiante.
Seguimiento de proyecto	Proceso en el cual, se registran citas y tareas para la evolución de trabajo, para la validación con el cronograma dispuesto.

Nota: Esta tabla contiene los casos de uso de la aplicación base.

Los casos de uso antes detallados se analizaron de la siguiente forma:

- Se identificó a todos los involucrados que intervienen en la aplicación base.
- Se ejecutó el aplicativo base.
- Se comprobó el flujo de información principal de cada uno de los casos de uso antes mencionados.

2.1.2. Análisis de la documentación de los casos de uso. A continuación, se analizó cada caso de uso y se detalló el funcionamiento de cada uno.

Como se muestra en la figura 5, el diagrama de registro de usuarios permite agregar nuevos usuarios al sistema para que estos puedan ingresar posteriormente.

Registro de usuarios

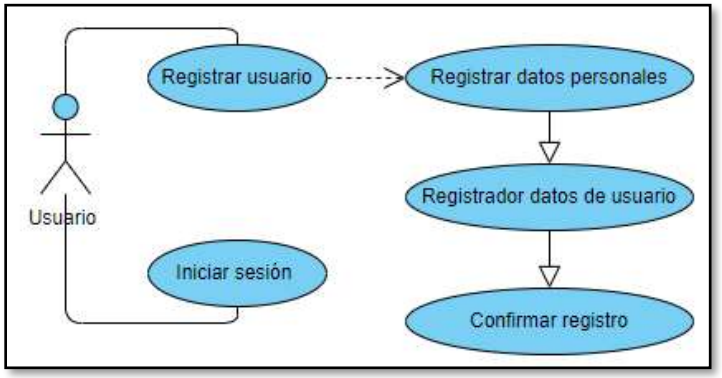


Figura 5. Registro de usuario.
Fuente: (Flores & Quisupangui, 2016).

Al ejecutar la aplicación base se identificó dos tipos de registros de usuarios siendo el primero el registro de estudiantes y el otro el registro por parte del personal administrativo de la aplicación.

- Registro de estudiantes: los estudiantes se pueden registrar de manera autónoma, ya que, el sistema de manera automática los registra con un perfil de estudiante.
- Registro de docente: el docente será ingresado por el administrador del aplicativo para determinar qué perfil ocupará dentro del aplicativo.

Publicación de temas

Como se muestra en la figura 6, el diagrama de publicación de temas permite la creación y publicación de temas de titulación.

Publicación de temas

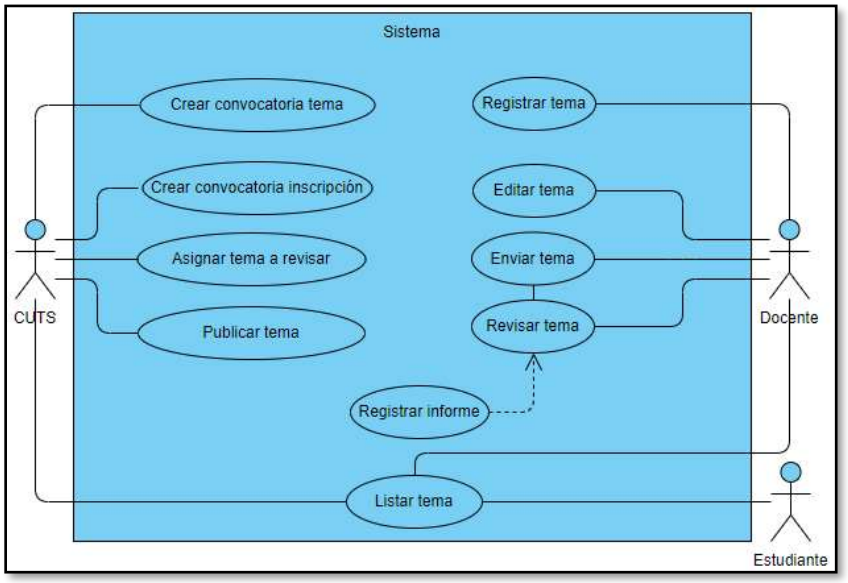


Figura 6. Publicación de temas.
Fuente: (Flores & Quisupangui, 2016).

En el proceso de publicación de temas intervienen dos actores, el primer actor es el docente quien propone los temas de titulación ya sea para Proyecto técnico o Artículo académico, y el segundo actor en este caso la CUTS, quien valida dichos temas para su publicación.

En este proceso los docentes pueden tener creados varios temas de titulación en el sistema, una vez que es abierta la convocatoria por parte de la CUTS, el docente podrá elegir que temas envía para su validación.

Inscripción

En la figura 7, el diagrama de inscripción permite al estudiante registrarse en la Unidad de Titulación para posteriormente una vez validada su inscripción por parte de la CUTS, este pueda optar por una opción de titulación.

Inscripción

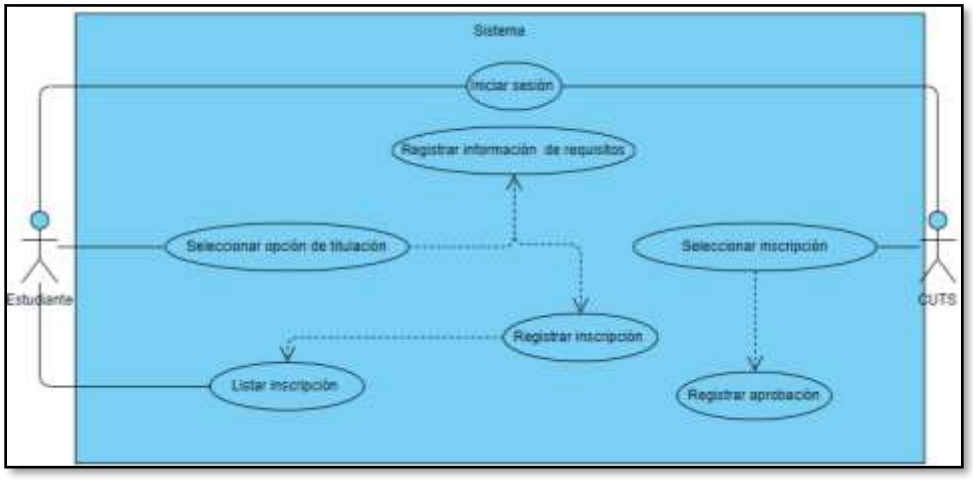


Figura 7. Inscripción a Unidad de Titulación.
Fuente: (Flores & Quisupangui, 2016).

El estudiante para inscribirse a la Unidad de Titulación debe cumplir con ciertos requisitos que son solicitados por la CUTS, si en alguno de los requisitos tuviese problemas el estudiante debe detallarlos para que la CUTS lo tenga en cuenta.

Mediante este proceso la CUTS podrá aprobar o negar la inscripción del estudiante a la Unidad de Titulación.

Aprobación de proyecto

Como se muestra en la figura 8, en el diagrama de aprobación de proyecto intervienen tres actores: estudiante, docente y CUTS.

Aprobación de proyecto

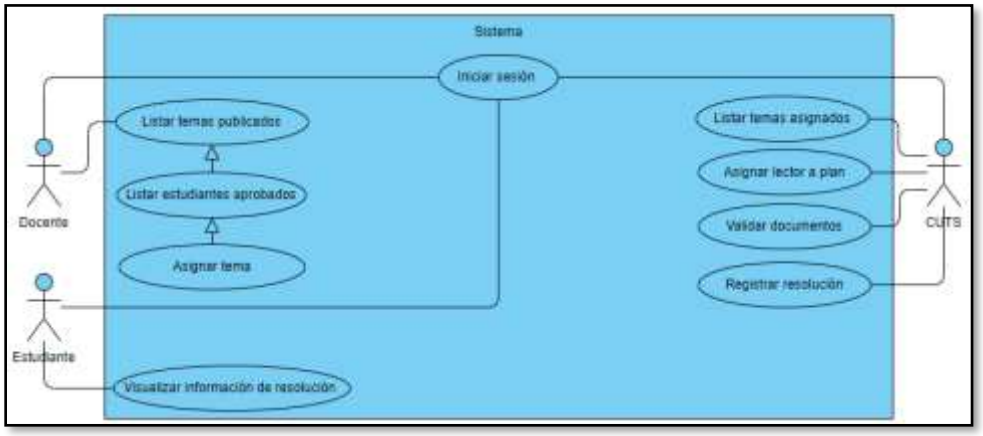


Figura 8. Aprobación del tema con resolución.
Fuente: (Flores & Quisupangui, 2016).

En dicho proceso el docente selecciona uno de los temas publicados por parte de él, dicho tema es asignado a un estudiante o grupo de estudiantes quienes deben tener una inscripción aprobada por la CUTS.

La CUTS recibe los temas asignados a los estudiantes, este asigna un lector de plan quien valida los requisitos para su aprobación, si el tema asignado es aprobado se registrará la resolución de aprobación para que inicie el proceso de desarrollo de opción de titulación seleccionada (Proyecto técnico o Artículo académico).

Seguimiento de proyecto

Como se muestra en la figura 9, el diagrama de seguimiento de proyectos se encarga de tener un control de los proyectos de titulación.

Seguimiento de proyecto

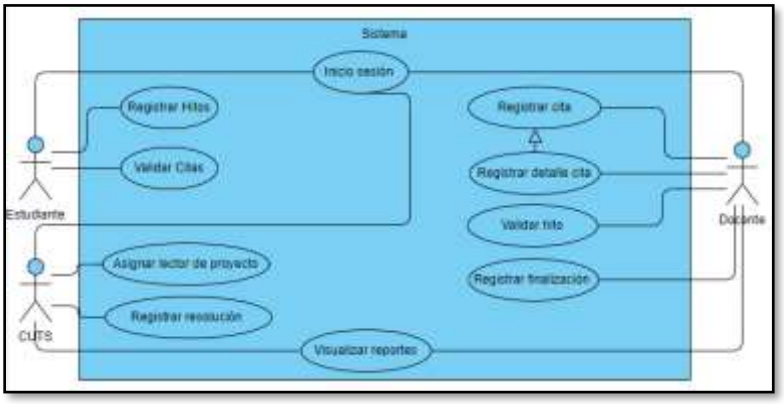


Figura 9. Seguimiento de proyecto.
Fuente: (Flores & Quisupangui, 2016).

En dicho proceso tanto el docente como el estudiante van observando el avance o evolución del proyecto de titulación, teniendo un registro de todos los hitos que ha tenido en el proyecto de titulación; tanto el docente como el estudiante tendrán acceso a un reporte donde se encuentran todos hitos y evoluciones registradas en el sistema.

Mediante los diagramas de caso de uso se identificó los principales flujos de información, actores y la interacción entre ellos que tiene la aplicación base, los cuales se visualiza en la tabla 6, de esta manera se obtuvo una idea clara del funcionamiento del aplicativo dando opciones en donde se podrían implementar las modificaciones para mejorar su funcionamiento.

Tabla 6. Casos de uso, descripción y actores que interviene en el proceso

Id	Caso de Usos	Descripción	Actores
001	Registro de usuarios	Registro de usuario.	• estudiante • docente • CUTS • administrador
002	Publicación de temas	Publicación de temas nuevos (Proyectos Técnicos o Artículo Académico).	• CUTS • docente • estudiante
003	Inscripción a la Unidad de Titulación	Inscripción a Unidad de Titulación.	• estudiante • CUTS
004	Aprobación del tema	- Docentes visualizan los temas publicados. - Asignación de tema a estudiante por CUTS. - CUTS visualiza temas asignados, asigna lector, valida documentos y registra resoluciones.	• estudiante • CUTS • Docente
005	Seguimiento del proyecto	Registro de citas y tareas, avance de proyecto, registro de hitos, cumplimiento de cronograma	• estudiante • CUTS • Docente

Nota: Esta tabla contiene los actores y procesos que cada uno ejecuta.

2.2. Análisis del código fuente de la aplicación base. Este proceso se lleva a cabo con la examinación del código fuente para identificar su estructura a nivel de paquetes, funciones, datos e información que esta aplicación maneja.

El primer paso fue examinar todo el árbol de directorios e ir revisando directorio por directorio para comprobar la información que cada uno contiene en su interior como se muestra en la figura 10.

Árbol de directorios

Figura 10. Árbol de directorios
Fuente: (Flores & Quisupangui, 2016)

A continuación, en la tabla 7 se detalla los directorios más relevantes con una descripción general de cada uno.

Tabla 7. Descripción del árbol de directorio de la aplicación base

Directorio	Descripción
Web Pages	Interfaz de usuario.
Source Package	Directorio que contiene el acceso a datos.
com.utes.bean	Directorio que se encarga de encapsular información para ser reutilizada en el proyecto.
com.utes.bean.data com.utes.util	Manejan archivos de configuración, validación y reportes.
com.utes bean.modelo	Mapeo de la base de datos.
com.utes configura	Configuración de Hibernate.
com.utes dao	Acceso de información hacia la base de datos.
com.utes dao.impl	Archivos de configuración para implementación de funciones.

Nota: Esta tabla contiene los directorios más relevantes de la aplicación base y una descripción general.

Entre los directorios relevantes se puede mencionar el “Web Pages” y el “Source Packages”, dichos directorios son la parte central del aplicativo base ya que, en ellos se agrupan la interfaz de usuarios y el acceso a datos como se muestra en la figura 11.

Directorios principales

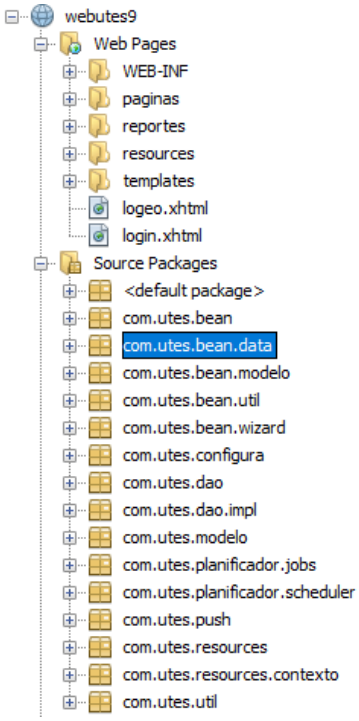


Figura 11. Directorios principales “Web Pages” y “Source Packages”
Fuente: (Flores & Quisupangui, 2016).

Como se observa en la figura 11, en el directorio “Web Pages” se encuentran todas las interfaces de usuarios que serán visualizadas por los usuarios, mientras que en el directorio “Source Packages” se encuentra todo el funcionamiento y lógica del negocio.

Método Convocatoria

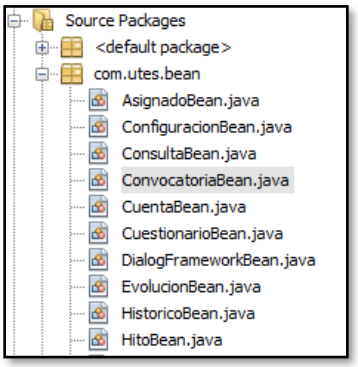


Figura 12. Método “ConvocatoriaBean”.
Fuente: (Flores & Quisupangui, 2016).

En la figura 11 se observa todos los subdirectorios o paquetes que fueron creados y utilizados para la aplicación, uno de estos directorios es “com.utes.bean”, tal como se muestra en la figura 12, en este directorio se almacena toda la información en archivos que encapsula cierta funcionalidad como es el caso del archivo “ConvocatoriaBean”, en el cual entre varias funciones destaca la función de crear una nueva convocatoria “saveConvocatoria” como se muestra en la figura 13.

Archivo “ConvocatoriaBean”

```
public void saveConvocatoria(ActionEvent actionEvent) {
  RequestContext context = RequestContext.getCurrentInstance();
  boolean validacion = false;

  try {
    utilfecha = new FechaUtil();

    if (getCvnombre().isEmpty() != true) {
      if (cvnumtema != 0) {
        this.enconvocatoria = new Convocatoria();
        this.enconvocatoria.setConNombre(cvnombre);
        this.enconvocatoria.setConNumerotema(cvnumtema);
        this.enconvocatoria.setConPeriodo(cvperiodo);
        this.enconvocatoria.setConFechainicio(cvfini);
        this.enconvocatoria.setConFechafin(cvffin);
        this.enconvocatoria.setConActivo(cvactivo);
        this.enconvocatoria.setConSecuencia(getCvsequence());

        if (daoconvocatoria.create(this.enconvocatoria)) {
          this.enconfigura = new SysConfiguracion();
          this.enconfigura.setConfEstado(cvactivo);
```

Figura 13. Archivo “ConvocatoriaBean”, función “ saveConvocatoria “.
Fuente: (Flores & Quisupangui, 2016).

2.3. Análisis de la base de datos. El análisis de la base de datos se realizó mediante un modelo lógico obtenido del trabajo de titulación Registro y Seguimiento de Opciones de Titulación, el que se visualiza en la figura 14, este diagrama describe todas las tablas y sus relaciones entre sí, que se emplearon para generar el aplicativo.

Diagrama general de Base de Datos

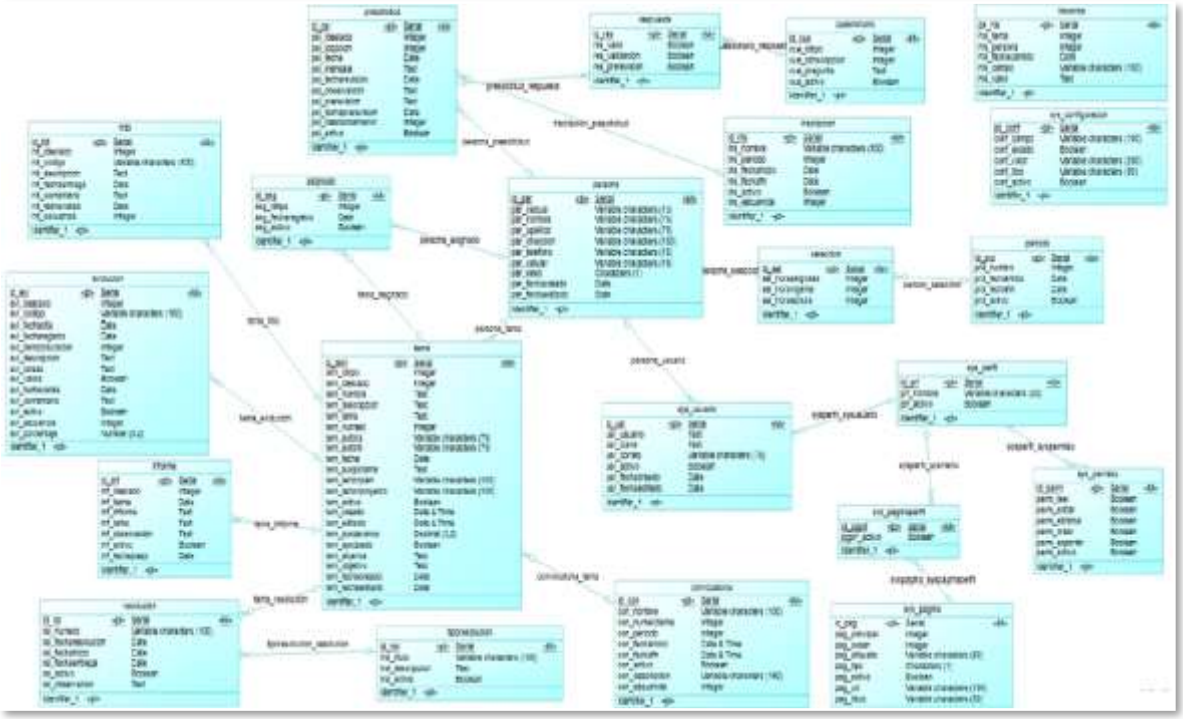


Figura 14. Diagrama lógico de base de datos de la aplicación base.
Fuente: (Flores & Quisupangui, 2016).

En este proceso se analizó la estructura e información de la base de datos para comprender su funcionamiento, relaciones y los datos que en ella se guardan.

Después de analizar la base de datos de la aplicación base se decidió mantener el mismo modelo ya que, no es necesario cambiar la estructura de datos por una nueva estructura o migrar a un nuevo motor de base de datos.

2.4. Análisis de las hojas de estilos CSS. La aplicación base al ser desarrollada bajo el framework JSF sus hojas de estilos o archivos CSS son muy básicos ya que, este framework está enfocado a la parte funcional dejando un poco de lado la interfaz de usuario realizando una interacción con el usuario minimalista.

Al desarrollar cualquier aplicación bajo este framework e ir agregando componente a estos se los agregan un estilo general con lo cual, se crean hojas de estilos para configurar:

- Tamaño y tipo de letra
- Ubicación de los componentes al ser visualizado por el usuario
- Tamaño a los campos o inputs.
- Colores que algún componente debe tomar al ejecutar una acción en especifica. Etc.

Estos archivos se crean para que los estilos afecten al aplicativo de forma general en la figura 15 se visualiza la configuración que adoptaran ciertas etiquetas dentro del aplicativo base.

Archivo "default.css"

```
body {
 background-color: #ffffff;
 font-size: 12px;
 font-family: 'Open Sans', sans-serif , Verdana, "Verdana CE",  Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE", sans-serif;
 color: #000000;
 margin: 10px;
}

h1 {
 font-family: 'Open Sans', sans-serif ,Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE";
 border-bottom: 1px solid #AFAFAF;
 font-size:  16px;
 font-weight: bold;
 margin: 0px;
 padding: 0px;
 color: #D20005;
}

a:link, a:visited {
 color: #045491;
 font-weight : bold;
 text-decoration: none;
}
```

Figura 15. Archivo "default.css".
Fuente: (Flores & Quisupangui, 2016).

En la figura 16 se visualiza la configuración que se utiliza para ubicar ciertos componentes dentro de aplicativo base al ser llamados o invocados por el usuario.

Archivo "cssLayout.css"

```
#top {
    position: relative;
    background-color: #036fab;
    color: white;
    padding: 5px;
    margin: 0px 0px 10px 0px;
}

#bottom {
    position: relative;
    background-color: #c2dfef;
    padding: 5px;
    margin: 10px 0px 0px 0px;
}

#left {
    float: left;
    background-color: #ece3a5;
    padding: 5px;
    width: 150px;
}

#right {
    float: right;
    background-color: #ece3a5;
    padding: 5px;
    width: 150px;
}
```

Figura 16. Archivo "cssLayout.css".
Fuente: (Flores & Quisupangui, 2016).

Perfil CUTS, menú “asignación-revisor”

Figura 17. . Perfil CUTS, menú “asignación-revisor”
Fuente: (Flores & Quisupangui, 2016).

Como se muestra en la figura 17, se visualiza la aplicación de las hojas de estilos que se muestran en las figuras 15 y 16. En el (Anexo B), se explica con mayor detalle el análisis de las hojas de estilos del aplicativo base y del aplicativo refactorizado.

CAPÍTULO 3

REFACTORIZACIÓN

3. Definición de la arquitectura

La aplicación refactorizada se estructuró bajo la arquitectura cliente-servidor y el estilo arquitectónico REST.

Al tener claro el estilo arquitectónico REST, se realizó la creación de la API REST, que actualmente se está utilizando por ser eficiente al momento de crear servicios e implementar URI's simplificando el tiempo de programación.

Se consideró que para solucionar los problemas de interoperabilidad y escalabilidad y llevar a cabo la refactorización las dos arquitecturas antes mencionadas se acoplaron y ayudaron a dotar de estas características al aplicativo refactorizado.

3.1. Arquitectura. Los dos frameworks utilizados, tanto Angular como Spring tienen una característica en común, que es el manejo del patrón arquitectónico MVC, para el desarrollo de las aplicaciones.

Al usar los frameworks antes mencionados se segmentó la aplicación base separando la capa de presentación de la capa de servicio, dando cabida a la refactorización de la misma. En la figura 18 se muestra la arquitectura empleada para el desarrollo de la aplicación refactorizada.

Arquitectura aplicación refactorizada

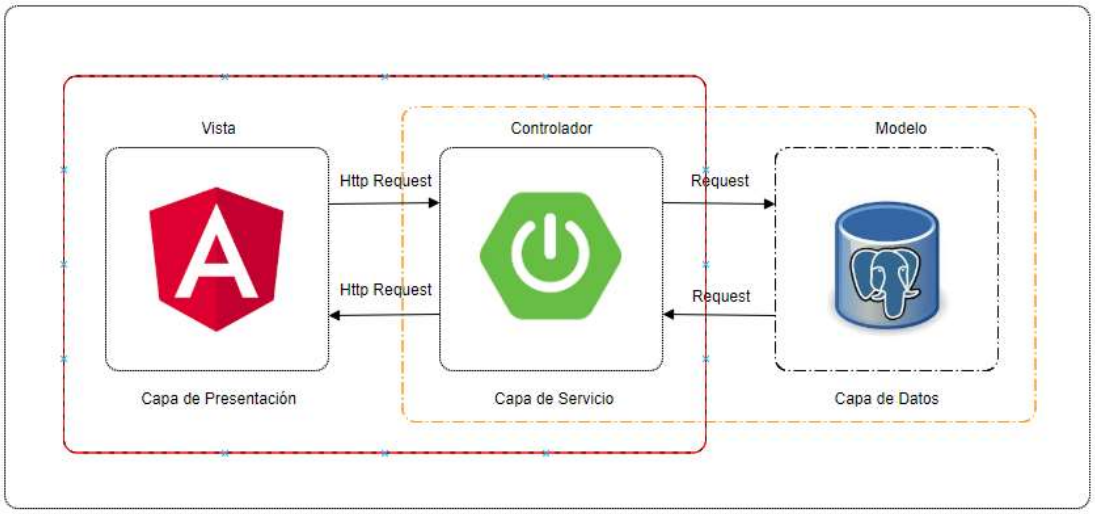


Figura 18. Arquitectura aplicación refactorizada.
Elaborado por: Diego Arias Y Roberth Proaño

3.2. Implementación de patrones de diseño de software. Al momento que se desarrolló la aplicación refactorizada se hizo uso de patrones de diseño que ayudaron a evitar y controlar duplicidad de código, logrando una estructura definida y ordenada.

3.2.1. Patrones de diseño en el front-end.

3.2.1.1. Singleton. Se crearon clases e interfaces al momento del desarrollo de la aplicación refactorizada ya que, facilitó el uso, reutilización e implementación de información dentro de la misma.

Al momento de refactorizar la aplicación se creó una variedad de clases que ayudaron a reutilizar el código haciendo la aplicación más eficiente y eficaz.

En las siguientes figuras se muestra la implementación e importación de una de las clases más importantes.

Clase “Tema”

```
export class Tema{
    idTem: number;
    persona: Persona;
    convocatoria: Convocatoria;
    temIdTipo: number;
    temIdEstado: number;
    temNombre: String;
    temDescripcion: String;
    temTema: String;
    temNumEst: number;
    temAutorA: String;
    temAutorB: String;
    temFechaPublicado: Date;
    temAuspiciante: String;
```

Figura 19. Clase “Tema”.
Elaborado por: Diego Arias y Roberth Proaño

En la figura 19, se muestran las variables que se utilizaron en el desarrollo e implementación de la clase “Tema”, esta clase contiene las variables que fueron implementadas en la base de datos, con lo cual se puede manipular información en cualquier componente desde el front-end.

Importación de la clase

```
import { Component } from '@angular/core';
import { Tema } from '../../../dto/tema';
import { Estaticos } from 'src/app/app.constants';
import { Asignado } from '../../../dto/asignado';
import { Sysusuario } from '../../../dto/sysusuario';
import { ActivatedRoute } from '@angular/router';

import { UserService,
         TemaService,
         AsignadoService,
         UtilService
} from '../../../services/services.index';


@Component({
  selector: 'app-estudiantetema',
  templateUrl: './estudiantetema.component.html',
  styles: [
  ]
})
export class EstudiantetemaComponent {
```

```
import { Component, OnInit} from '@angular/core';
import { Tema } from '../../../dto/tema';
import { Asignado } from '../../../dto/asignado';
import { Sysusuario } from '../../../dto/sysusuario';
import { Estaticos } from 'src/app/app.constants';

import { AsignadoService,
         UserService,
         TemaService,
         UtilService,
         DetalleService }
from '../../../services/services.index';


@Component({
  selector: 'app-estudianteevolucion',
  templateUrl: './estudianteevolucion.component.html'
})
export class EstudianteevolucionComponent implements OnInit {
```

Figura 20. Importación de la clase “Tema” en los componentes.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 20, la clase “Tema” fue utilizada en varios componentes entre ellos los componentes “EstudiantetemaComponent” y “EstudianteevolucionComponent” por lo que, al crear esta clase se puede usar la información solo con importar una línea de código evitando crear las variables en cada componente.

3.2.1.2. Constructor. Al generar un componente en Angular, crea por defecto un constructor, siendo una de las funciones inicializar variables, objetos o servicios que se requieran en el componente para funcionar correctamente. Al inicializar un servicio dentro del constructor este dará acceso a la información que proviene del back-end. En la figura 21, se muestra la inicialización de varios servicios entre ellos el servicio "UserService" el cual contiene información del usuario como información personal.

Constructor

```
constructor(  private userService: UserService,
              private temaService: TemaService,
              private asignadoService: AsignadoService,
              public detalleService: DetalleService,
              public utilService: UtilService
            ) {}
```

Figura 21. Constructor de "EstudiantehitoComponent".
Elaborado por: Diego Arias y Roberth Proaño

3.2.1.3. Observable. Los observables se utilizaron para subscribirse a eventos, dichos eventos contienen un flujo de información y al suscribirse a uno de estos se puede hacer uso de la información que está contenida en ellos. A continuación, en la figura 22, se muestra el uso del observable.

Observable Servicio "TemaService"

```
export class TemaService {

  private urlEndPoint: string = Lang.urlEndPoint.tema;
  private httpHeaders = new HttpHeaders({'Content-Type': 'application/json'});

  constructor(private http: HttpClient) { }

  create(tematipo: Tema) : Observable<Boolean> {
    return this.http.post<Boolean>(`${this.urlEndPoint}/create`, tematipo, {headers: this.httpHeaders})
  }

  update(tematipo: Tema): Observable<Tema>{
    return this.http.post<Tema>(`${this.urlEndPoint}/update`, tematipo, {headers: this.httpHeaders})
  }

  delete(id: number): Observable<Tema>{
    return this.http.delete<Tema>(`${this.urlEndPoint}/delete/${id}`, {headers: this.httpHeaders})
  }

}
```

Figura 22. Observables del servicio "TemaService".
Elaborado por: Diego Arias y Roberth Proaño

Como se visualiza en la figura 22, el patrón observable se implementó dentro del servicio "TemaService" ya que, dependiendo donde se use un observable este maneja diferentes flujos de información, en este caso se implementó para realizar peticiones HTTP desde el front-end hacia el back-end.

Suscripción al observable

```
public loadDialogDetalleDesarrollo( idTema: number ): void {
   this.loading = true ;
   if (idTema) {
     this.temaService.getById(idTema).subscribe(
       (tema) => {
        if (tema != null) {
          this.loading = false ;
          this.temaSeleccionado = tema;
           this.temaSeleccionado.asignados =
              this.temaSeleccionado.asignados.filter(s =>
                  ( s.asgIdTipo == Estaticos.TIPO_ID_ASIGNACION_ESTUDIANTE ||
                    s.asgIdTipo == Estaticos.TIPO_ID_ASIGNACION_LECTORPLAN ||
                    s.asgIdTipo == Estaticos.TIPO_ID_ASIGNACION_LECTORPROYECTO
                  )
              );
        }
       }
     )
   }
 }
```

Figura 23. Patrón Observable en el componente "estudianteevolucióncomponent".
Elaborado por: Diego Arias y Roberth Proaño

Como se visualiza en la figura 23, para obtener la información que se encuentra en el observable del servicio "TemaService" se subscribió al mismo. En este caso se subscribío a la petición get "getById" para tener información del usuario y su relación con el tema asignado dentro del componente "EstudianteevolucionComponent".

3.2.2. Patrones de diseño en el back-end. Al momento de desarrollar cualquier software existe variedad de patrones que indistintamente del tipo de lenguaje de programación son estándares que son implementados por los frameworks o lenguajes de programación.

3.2.2.1. Factory. Este patrón ayudó a centralizar la creación de objetos y utilizarlos en diferentes componentes.

Interface “AreaPersonaService”

```
@Service
public interface AreaPersonaService extends IParsable<AreaPersonaDTO, AreaPersona> {

    public boolean create(AreaPersonaDTO obj);

    public boolean update(AreaPersonaDTO obj);

    public boolean delete(Integer id);

    public List<AreaPersonaDTO> obtenerListadoAreaPersona(boolean activo);

    public AreaPersonaDTO obtenerAreaPersona(Integer id);

    public List<AreaPersonaDTO> obtenerListadoAreapersonaxPersona(Integer idper, boolean activo);

    public AreaPersonaDTO obtenerListadoAreapersonaxPersona(Integer idper, Integer idare, boolean activo);

    public List<AreaPersonaDTO> obtenerListadoAreapersonaxPersona(String ids, Integer idper, boolean activo);
}
```

Figura 24. Patrón de diseño Factory.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 24, se visualiza el patrón factory implementado en el servicio “AreaPersonaService” extendiendo de las clases “AreaPersona” y “AreaPersonaDTO” para encapsular y reutilizar código en diferentes componentes del back-end.

3.2.2.2. ***DTO.*** Este patrón se utilizó para mapear las variables utilizadas en las tablas de la base de datos con el fin de transferir información entre el cliente y el servidor o viceversa.

Clase “TemaDTO”

```
public class TemaDTO implements Serializable {
    private static final long serialVersionUID = 1L;
    private Integer idTem;
    private PersonaDTO persona;
    private ConvocatoriaDTO convocatoria;
    private Integer temIdTipo;
    private Integer temIdEstado;
    private String temNombre;
    private String temDescripcion;
    private String temTema;
    private Integer temNumEst;
    private String temAutorA;
    private String temAutorB;
    private Date temFechaPublicado;
    private String temAuspiciante;
    private String temLectorPlan;
    private String temLectorProyecto;
    private Boolean temActivo;
    private Date temFechaCreado;
    private Date temFechaEditado;
```

Figura 25. Patrón de diseño DTO en el componente “TemaDTO”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 25, la clase “TemaDTO” es utilizada para mapear la tabla “temas” con sus atributos de la base de datos.

3.3. Código relevante. En esta sección se comparó la aplicación base y la aplicación refactorizada donde se expuso los directorios y archivos para comprobar los cambios efectuados.

3.3.1. Código Back-End.

3.3.1.1. Árbol de directorios. Como se muestra en la figura 26, se visualiza los directorios de la aplicación base y la aplicación refactorizada, la primera diferencia que se destaca es la eliminación del directorio Web Pages (Interfaz de usuario) con esto se desacoplo el front-end del back-end.

Árbol de directorios

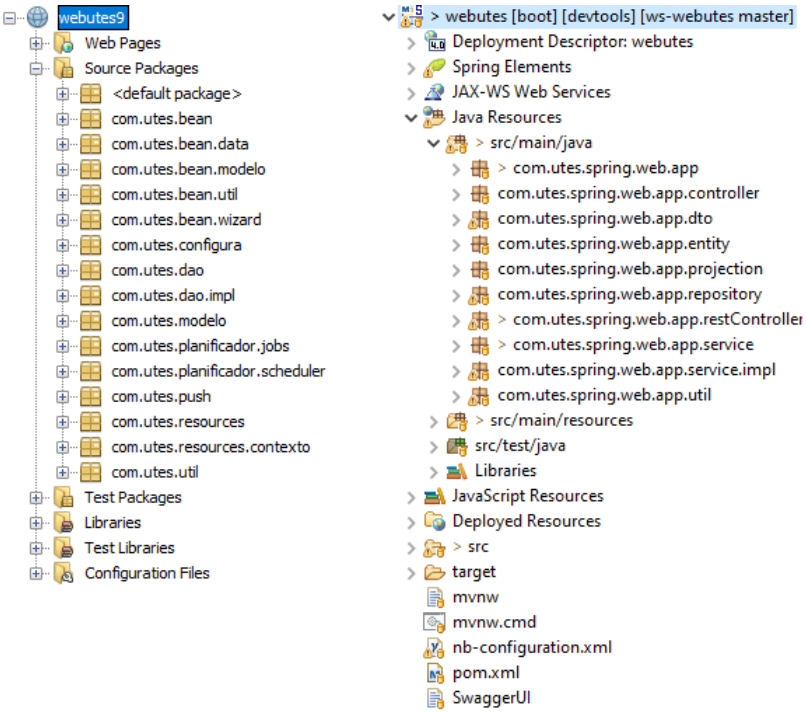


Figura 26. Árbol de directorios aplicación base vs aplicación refactorizada.
Elaborado por: Diego Arias y Roberth Proaño

Además de eso se redujo el número de directorios ya que, gracias al framework utilizado para desarrollar el back-end (Spring), optimizó las configuraciones que antes el desarrollador tenía que realizar manualmente.

A continuación, para nombrar el árbol de directorios omitiendo una parte del nombre del directorio para ser más eficaces:

- aplicación base: com.utes
- aplicación refactorizada: com.utes.spring.web

Tabla 8. Comparación del árbol de directorios

Aplicación Base	Aplicación Refactorizada
• Bean • Bean.data • Bean.modelo • Bean.util • Bean.wizard • Configura • Dao • Dao.impl • Modelo • Planificador.jobs • Planificador.scheduler • Push • Resources • Resources.contexto • Útil	• App • App.controller • Dto • Entity • Projection • Repository • restController • Service • Service.impl • Útil

Nota: Esta tabla contiene el árbol de directorios de la aplicación base y la aplicación refactoriza.

Como se observa en la tabla 8, se redujo el número de directorios, con lo cual no significa que se eliminaron directorios y sus archivos, lo que sucedió fue que ciertos directorios fueron agrupados o restructurados en nuevos directorios gracias al framework utilizado.

3.3.1.2. Comparación de directorios. Como se muestra en la tabla 8, al crear el back-end del aplicativo refactorizado se creó y reestructuro el directorio con lo cual se optimizo el back-end dejando solo los directorios que pertenecen a la capa de servicios. Algunos directorios fueron segmentados, agrupados y renombrados ya que, es la forma como se maneja el framework empleado.

Tabla 9.Arbol de directorio del aplicativo base y del aplicativo refactorizado

Aplicación Base	Aplicación Refactorizada
Bean	Repository
Bean.Modelo Modelo	Entity
Dao	Dao
Dao.impl	RestController Service Service.impl
Útil	Útil
Bean.util Bean.data Push Resources Resourses.contexto Planificador.jobs Planificador.scheduler Wizard	Front-End
	Projection
Configura	App App.controller

Nota: Esta tabla contiene el árbol de directorios de la aplicación base y la nueva estructura de la aplicación refactoriza.

Como se visualiza en la tabla 9, se muestra los directorios, su nueva organización y como fueron estructurados, creados, segmentados o reagrupados.

Uno de los directorios creados fue el directorio nombrado “projection” en el aplicativo refactorizado, este directorio se agregó para crear objetos personalizados en los cuales solo se manejan las propiedades que requieren ser utilizadas en lugar de llamar todas las variables de una clase, optimizando el tiempo de respuesta.

Directorio

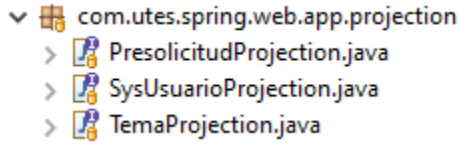


Figura 27. Directorio “projection”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 27, el directorio “projection” contiene tres clases personalizadas, con el fin de tener solo información necesaria y requerida evitando así un consumo de información innecesario.

Comparación de archivos “PresolicitudProjection” y “PresolicitudDTO”

```
public interface PresolicitudProjection {
    Integer getIdPsl();
    Integer getIdPer();
    Integer getIdIns();
    Integer getPslIdEstado();
    Integer getPslIdOpcion();
    Date getPslFecha();
    String getPslMensaje();
    Date getPslFechaRevision();
    String getPslObservacion();
    String getPslPrerevision();
    Date getPslFechaPrerevision();
    Integer getPslIdEstadoAnterior();
    Boolean getPslActivo();
}
```

```
public class PresolicitudDTO implements Serializable {
    private static final long serialVersionUID = 1L;
    private Integer idPsl;
    private PersonaDTO persona;
    private InscripcionDTO inscripcion;
    private Integer pslIdEstado;
    private Integer pslIdOpcion;
    private Date pslFecha;
    private String pslMensaje;
    private Date pslFechaRevision;
    private String pslObservacion;
    private String pslPrerevision;
    private Date pslFechaPrerevision;
    private Integer pslIdEstadoAnterior;
    private Boolean pslActivo;
    private String pslFiles;
    private Integer idPersona;
    private Integer idInscripcion;
    private String nombreEstado;
    private String nombreTipo;
    private List<RespuestaDTO> respuestas;

    public PresolicitudDTO() {
        this.respuestas = new ArrayList<RespuestaDTO>();
    }
}
```

Figura 28. Clase “PresolicitudProjection” y Clase “PresolicitudDTO”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 28, se observa que en la clase “PresolicitudProjection” contiene una interface que hace referencia a la clase “PresolicitudDao”, pero con la diferencia que en la clase “PresolicitudProjection” solo llama las propiedades que se requieren, reduciendo así la carga de información innecesaria.

3.3.1.3. Segmentación del directorio “dao.impl”. Otro de los directorios segmentados fue el directorio “dao.impl” ya que, al crear la API REST, esta ajustó y segmento aún más la lógica, encapsulando pequeñas partes de código, creando así una aplicación mantenible y escalable.

Segmentación de clases

```
@Service
public interface TemaService extends IParsable<TemaDTO, Tema> {

    public boolean create(TemaDTO obj);

    public boolean update(TemaDTO obj);

    public boolean delete(Integer id);

    public List<TemaDTO> obtenerTemas();
```

```
@Override
    @Transactional(readOnly = true)
    public List<TemaDTO> obtenerTemas() {
        final List<Tema> listTemasBD = this.temaRepository.findAll();
        final List<TemaDTO> resultado = new ArrayList<TemaDTO>();
        if (listTemasBD != null && !listTemasBD.isEmpty()) {
            for (final Tema tema : listTemasBD) {
                resultado.add(this.convertirEntityToDto(tema, true, false));
            }
        }
        return resultado;
    }
```

```
@RequestMapping(value = "/obtenerTemas", method = RequestMethod.GET, produces = {
            "application/json" + ";charset=utf-8" })
    public @ResponseBody List<TemaDTO> obtenerTemas() {
        return temaService.obtenerTemas();
    }
```

Figura 29. Clases “TemaService”, “TemaServiceImpl” y “TemaRestController”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 29, se desacopla la lógica del negocio, separando en diferentes clases. El primer archivo “TemaService” contiene solo el nombre de la función “obtenerTemas”, el segundo archivo “TemaServiceImp” contiene la implementación del archivo TemaService para poder hacer uso de la función “obtenerTemas” agregando toda la lógica que esta función debe tener para ser ejecutada.

El tercer archivo “TemaWSRestCntroller” hace uso de los dos anteriores encapsulando toda la lógica para poder ejecutar la petición HTTP que este caso es una petición “get” para listar un recurso.

3.3.1.4. Refactorización de la función “obtenerTemas”. La función “obtenerTemas” es una de las funciones que fue desarrolla en la aplicación base con el fin de listar todos los temas de titulación que existan en el aplicativo. Esta función fue refactorizada empleando Spring framework, como se muestra en la figura 30, la funcionalidad y lógica se mantuvo cambiando la sintaxis o forma como se ejecuta la función para generar el mismo resultado.

Función “obtenerTemas” aplicación base

```
@Override
public List<Tema> obtenerTemas() {
 List<Tema> listado = null;
 Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();
 String sql = tabla + " order by idTem asc";

 try {
  sesion.beginTransaction();
  listado = sesion.createQuery(sql).list();

  for (Tema object : listado) {
   Hibernate.initialize(object.getPersona());
   Hibernate.initialize(object.getConvocatoria());
   Hibernate.initialize(object.getAsignados());
   /**
    * add di3 miercoles
    */
  }

  sesion.beginTransaction().commit();
 } catch (HibernateException e) {
  sesion.beginTransaction().rollback();
  throw e;
 }

 return listado;
}
```

Figura 30. Función “obtenerTemas” aplicación base.
Elaborado por: Diego Arias y Roberth Proaño

Función “obtenerTemas” aplicación refactorizada

```
@Override
    @Transactional(readOnly = true)
    public List<TemaDTO> obtenerTemas() {
        final List<Tema> listTemasBD = this.temaRepository.findAll();
        final List<TemaDTO> resultado = new ArrayList<TemaDTO>();
        if (listTemasBD != null && !listTemasBD.isEmpty()) {
            for (final Tema tema : listTemasBD) {
                resultado.add(this.convertirEntityToDto(tema, true, false));
            }
        }
        return resultado;
    }
```

Figura 31. Función “obtenerTemas” aplicación refactorizada.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 31, hubo una reducción de líneas de código, pero dicha reducción no significa que su resultado se ha alterado de lo contrario, la funcionalidad permanece intacta al igual que su resultado.

3.3.2. Código Front-End. Para el desarrollo del front-end, se creó un árbol de directorios siguiendo una estructura y distribución ordenada logrando así mejorar su mantenibilidad y tener un mejor control sobre la aplicación como se muestra en la figura 32.

Árbol de directorio

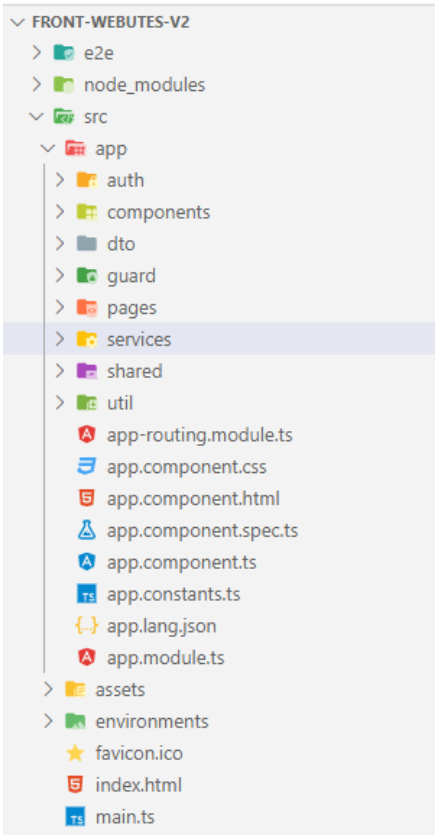


Figura 32. Árbol de directorio front-end.
Elaborado por: Diego Arias y Roberth Proaño.

Tabla 10. Descripción del árbol de directorios del front-end

Directorio	Descripción
• auth	Se creó un directorio solo para los componentes de login y registro de usuarios ya que, estos componentes no usan el mismo estilo que los demás componentes.
• components	Este directorio contiene componentes que se utilizan más de una vez en diferentes componentes con lo que, estos componentes pueden ser reutilizados varias veces sin crear duplicidad de código.
• dto	Este directorio contiene los objetos que manejan datos de las clases a ser usadas en el front-end.
• guards	Este directorio permite proteger las rutas, con lo cual solo usuarios autenticados podrán hacer uso de las rutas y del aplicativo.
• pages	Este directorio contiene todos los componentes que fueron utilizados y visualizados según su perfil

• services	Este directorio contiene todos los servicios que hacen de puente para la comunicación entre el front-end y el back-end.
• shared	Este directorio contiene los componentes estáticos de la aplicación los cuales son header, footer y menú.
• util	Este directorio contiene un servicio que maneja información que siempre debe estar presente en el proyecto, con lo cual se crea un solo punto de acceso para consumir información que se requiere en todo el aplicativo.

Nota: Esta tabla contiene la descripción del árbol de directorios del aplicativo refactorizado.

Como se muestra en la tabla 10, se detalla la estructura del front-end, con lo cual, se segmento el código para dotarlo de ciertas características como son: mantenibilidad, escalabilidad y modularidad. A continuación, se describe parte de los directorios más importantes.

3.3.2.1. *Directorio "services".* Angular emplea el uso de servicios para recibir y enviar información de y hacia el back-end, con el fin de centralizar estos servicios se creó un directorio que contiene todos los servicios que utiliza la aplicación refactorizada.

Directorio "services"

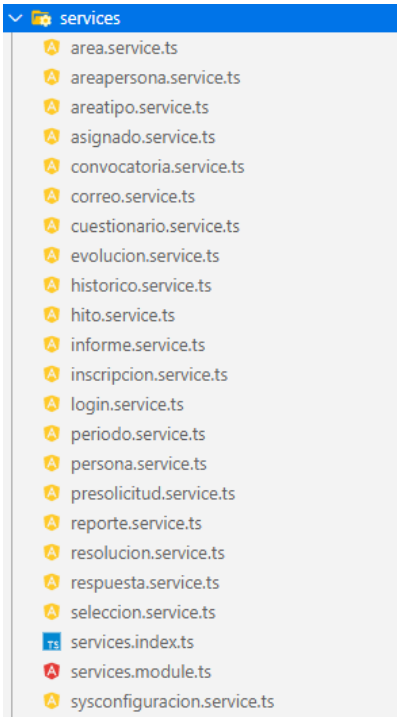


Figura 33. Directorio "services".
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 33, se visualizan los servicios creados para ser empleados en el consumo de información desde front-end. Hacia el back-end además, como buena práctica se creó dos archivos de configuración dentro del directorio “services” como se muestra en la figura 34.

Archivos de configuración

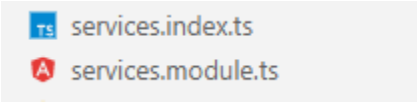


Figura 34. Archivos de configuración de los servicios.
Elaborado por: Diego Arias y Roberth Proaño.

El archivo “services.index.ts” contiene todos los servicios que utiliza la aplicación refactorizada con lo cual, se centraliza el llamamiento de los mismos disminuyendo las líneas de código al momento de usar en un componente más de un servicio, pero la característica de este archivo es que todos los servicios son exportados para poder ser utilizados o consumidos en cualquier lugar del aplicativo.

Importación de servicios

```
import { Component, OnInit} from '@angular/core';
import { Tema } from '../../../dto/tema';
import { Asignado } from '../../../dto/asignado';
import { Sysusuario } from '../../../dto/sysusuario';
import { Estaticos } from 'src/app/app.constants';

import { AsignadoService,
         UserService,
         TemaService,
         UtilService,
         DetalleService }
from '../../../services/services.index';


@Component({
  selector: 'app-estudianteevolucion',
  templateUrl: './estudianteevolucion.component.html'
})
export class EstudianteevolucionComponent implements OnInit {
```

Figura 35. Importación de servicios en el componente “EstudianteevolucionComponent”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 35, al implementar el archivo de configuración “services.index.ts” se centralizó y optimizó el llamamiento de servicios en los componentes, al momento de importar más de un servicio dentro de un componente el código puede ser de difícil mantenimiento ya que,

al tener los servicios en directorios diferentes las importaciones de cada servicio aumentan las líneas de código innecesarias.

El otro archivo creado fue el “service.module.ts”, esta implementación permite tener lo más limpio posible el módulo principal de la aplicación “app.module.ts”, este módulo solo debe tener las importaciones necesarias para el funcionamiento óptimo, correcto y eficiente del aplicativo.

Archivo “service.module.ts”

```
import {
  LoginService,
  UserService,
  AsignadoService
  EvolucionService
  HitoService
  InscripcionService
} from './services.index';

@NgModule({
  declarations: [],
  imports: [CommonModule, HttpClientModule],
  providers: [
    LoginService,
    UserService,
    AsignadoService
    EvolucionService
    HitoService
    InscripcionService
    ],
})
export class ServiceModule {}
```

Figura 36. Archivo “service.module.ts”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 36, se importa todos los servicios que van a ser consumidos y se los provee al aplicativo para ser utilizados.

Módulo principal

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { AppComponent } from './app.component';
//Modulos
import { AuthModule } from './auth/auth.module';
import { PagesModule } from './pages/pages.module';
import { ServiceModule } from './services/services.module'
//Rutas
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    CommonModule,
    AuthModule,
    AppRoutingModule,
    PagesModule,
    ServiceModule
  ],
  providers: [
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Figura 37. Módulo principal “app.mudule.ts”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 37, el módulo principal “app.module” fue optimizado al eliminar la importación de todos los servicios y solo agregar el módulo que los contiene “service.module.ts”.

3.3.2.1. Directorio “components”. Este directorio se creó con la finalidad de reutilizar código que es requerido más de una vez en diferentes componentes de la aplicación refactorizada.

Directorio “components”

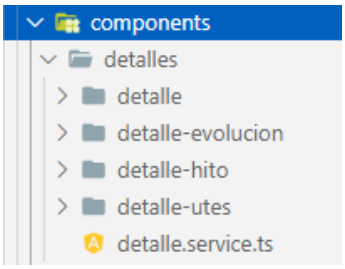


Figura 38. Directorio “components”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 38, se generaron tres componentes que fueron utilizados en diferentes lugares del aplicativo, lo que se implementó es una comunicación padre e hijo entre componentes

con lo cual, el componente hijo está esperando ser llamando por el componente padre para ser visualizado por el usuario.

Esta funcionalidad se realizó para visualizar los cuadros de diálogos ya que, estos componentes eran recurrentes con los diferentes perfiles por lo cual, se optó en implementar esta característica de Angular logrando reducir, optimizar y reutilizar código.

Componente “detalle.component.html”

```
<p-dialog header="Detalle"
          [(visible)]="detalleService.dialogDetalleTema"
          [modal]="true"
          [style]="{width: '60vw'}"
          [maximizable]="true"
          [draggable]="true"
          [resizable]="true">
        <p-tabView>
            <p-tabPanel header="Tema" leftIcon="pi pi-bookmark">
                <div class="col-xs-12">
                    <div class="row">
                        <div class="col-sm-6" *ngIf="tema.persona">
                            <div class="profile-info-name">Autor: </div>
                            <div class="profile-info-value">
                                <span >{{tema.persona.perApellido }} {{tema.persona.perNombre}}</span>
                            </div>
                        </div>
                        <div class="col-sm-6">
                            <div class="profile-info-name">Estado: </div>
                            <div class="profile-info-value" style="background-color: #b5d4fb;">
                                <span><b>{{getEstadoTema(tema.temIdEstado)}}</b></span>
                            </div>
                        </div>
                    </div>
                </div>
            </p-tabPanel>
        </p-tabView>
        <ng-template pTemplate="footer">
            <p-button icon="pi pi-check" (click)="detalleService.dialogDetalleTema=false" label="OK"
                styleClass="p-button-text"></p-button>
        </ng-template>
</p-dialog>
```

Figura 39. Componente “detalle.component.html”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 39, el archivo “detalle.component.html”, que contiene el componente hijo “detalle.component.html” siendo esta la interfaz gráfica que observa el usuario.

Importación del componente

```
<!-- Importacion del componente detalle -->
    <app-detalle *ngIf="temaSeleccionado" [tema]="temaSeleccionado"></app-detalle>
<!-- Importacion del componente detalle -->
```

Figura 40. Importación del componente hijo dentro del componente padre.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 40, con una sola línea de código, se implementa todo un componente hijo “detalle.component.html” para ser usando por el componente padre

“estudianteevolucion.component.html” según lo requiera. De esta manera se reduce, optimiza y reutiliza código de una manera adecuada.

3.3.2.2. Archivos de configuración. Endpoints, para poder usar los recursos credos en el back-end, debemos crear direcciones o endpoints con los cuales podremos realizar peticiones HTTP, de igual manera este archivo ayuda a centralizar los endpoints ya que, esta información es variable lo que permite cambiar o agregar nueva información en un solo archivo para que surta efecto en todo el proyecto.

Endpoints


```
{
    "urlEndPoint":{
        "areatipo":       "http://localhost:8090/api/areatipo",
        "area":           "http://localhost:8090/api/area",
        "areapersona":    "http://localhost:8090/api/areapersona",
        "asignado":       "http://localhost:8090/api/asignado",
        "convocatoria":   "http://localhost:8090/api/convocatoria",
        "correo":         "http://localhost:8090/api/correo",
        "cuestionario":   "http://localhost:8090/api/cuestionario",
        "evolucion":      "http://localhost:8090/api/evolucion",
        "historico":      "http://localhost:8090/api/historico",
        "hito":           "http://localhost:8090/api/hito",
        "informe":        "http://localhost:8090/api/informe",
        "inscripcion":    "http://localhost:8090/api/inscripcion",
        "periodo":        "http://localhost:8090/api/periodo",
        "persona":        "http://localhost:8090/api/persona",
        "presolicitud":   "http://localhost:8090/api/presolicitud",
        "resolucion":     "http://localhost:8090/api/resolucion",
        "respuesta":      "http://localhost:8090/api/respuesta",
        "seleccion":      "http://localhost:8090/api/seleccion",
        "sysconfiguracion": "http://localhost:8090/api/sysConfiguracion",
        "syspaginaperfil": "http://localhost:8090/api/sysPaginaPerfil"
    }
}
```


Figura 41. Endpoints “app.lang.json”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 41, se visualiza todos los endpoints que se emplearon en el archivo “app.lang.json”, creados para la conexión y consumo de los recursos del back-end.

Importación del archivo

```
import Lang from '../app.lang.json';

export class TemaService {

  private urlEndPoint: string = Lang.urlEndPoint.tema;
  private httpHeaders = new HttpHeaders({'Content-Type': 'application/json'});

  constructor(private http: HttpClient) { }

  create(tematipo: Tema) : Observable<Boolean> {
    return this.http.post<Boolean>(`${this.urlEndPoint}/create`, tematipo, {headers: this.httpHeaders})
  }
```

Figura 42. Importación del archivo “app.lang.json” dentro del servicio “TemaService”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 42, lo primero que se realizó es la importación del archivo de configuración “app.lang.json”, con lo cual ya se pudo usar el endpoint deseado e implementar las peticiones HTTP requeridas, en este caso el endpoint requerido fue el de crear un tema de titulación “create”.

Manejo de variables

En el archivo “app.constants.ts”, se asignan valores a todas las variables que se usan en el front-end provenientes del back-end, con lo cual, estas variables se crean una vez y se las implementan cuando y donde sea necesario dentro del aplicativo.

Manejo de variables

```
export class Estaticos {
    static readonly TIPO_ID_CUESTIONARIO_INSCRIPCION = 220;
    static readonly TIPO_ID_CUESTIONARIO_PREREVISION = 221;
    static readonly TIPO_ID_ASIGNACION_REVISOR = 210;
    static readonly TIPO_ID_ASIGNACION_LECTORPLAN = 212;
    static readonly TIPO_ID_ASIGNACION_LECTORPROYECTO = 213;
    static readonly TIPO_ID_ASIGNACION_ESTUDIANTE = 211;
    static readonly TIPO_LABEL_ASIGNACION_REVISOR = 'REVISOR TEMA';
    static readonly TIPO_LABEL_ASIGNACION_LECTORPLAN = 'LECTOR PLAN';
}
```

Figura 43. Manejo de variables.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 43, se creó variables estáticas y se les asignó un valor dependiendo el tipo de variable, las variables son de tipo entero o cadena de caracteres, en este caso las variables que se fueron asignadas con un valor numérico representan un estado que proviene de la base de datos. Este proceso se realizó con el fin de optimizar el proceso de desarrollo ya que, al asignar un significado a estos estados es más fácil su interpretación y uso.

Importación del archivo

Figura 44. Importación del archivo “app.constant.ts” en el componente “docenteasignacion.component.ts”
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 44, se hizo uso de las variables configuradas en el archivo “app.constant.ts” dentro de componente “docenteasigancio.component.ts”, al tener configuradas estas variables se hace más fácil el crear funciones sencillas para emitir resultados concretos.

3.4. Interfaces Aplicación Refactorizada. Una vez entendido el funcionamiento de la aplicación base se decidió mantener el mismo diseño de la interfaz de usuario ya que, este diseño contiene características importantes como navegabilidad, usabilidad, amigable e intuitiva para el usuario, pero desarrollado en Angular ya que, el aplicativo base fue desarrollado en JSF (JavaServer Faces).

A continuación, se han seleccionado las interfaces de los perfiles de usuarios que intervienen en el aplicativo para tener una idea clara de su diseño y funcionamiento.

3.4.1. Perfil estudiante.

Pantalla principal estudiante

Figura 45. Pantalla principal perfil estudiante.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 45, el usuario visualiza el tema de titulación al cual fue asignado con información resumida.

Visualización del cuadro dialogo

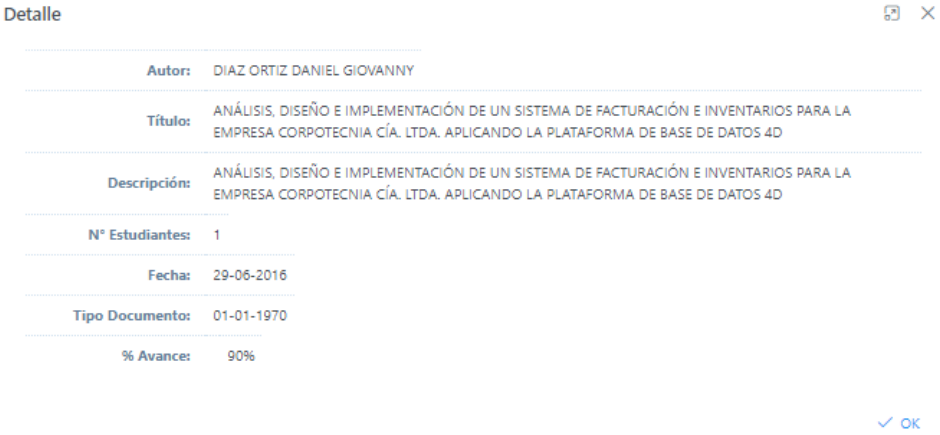


Figura 46. Visualización del cuadro dialogo “Detalle”.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 46, si el usuario necesita tener una información más específica este podrá observarlo en un cuadro de dialogo, en el cual consta toda la información relevante al tema de titulación asignado.

3.4.2. Perfil docente.

Pantalla principal docente

Figura 47 Pantalla principal perfil docente.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 47, pantalla Gestión de Temas del docente, en ella se visualiza los temas de titulación que el docente tiene en diferentes estados como creados sin validar y creados validados.

Cuadro de dialogo para crear tema

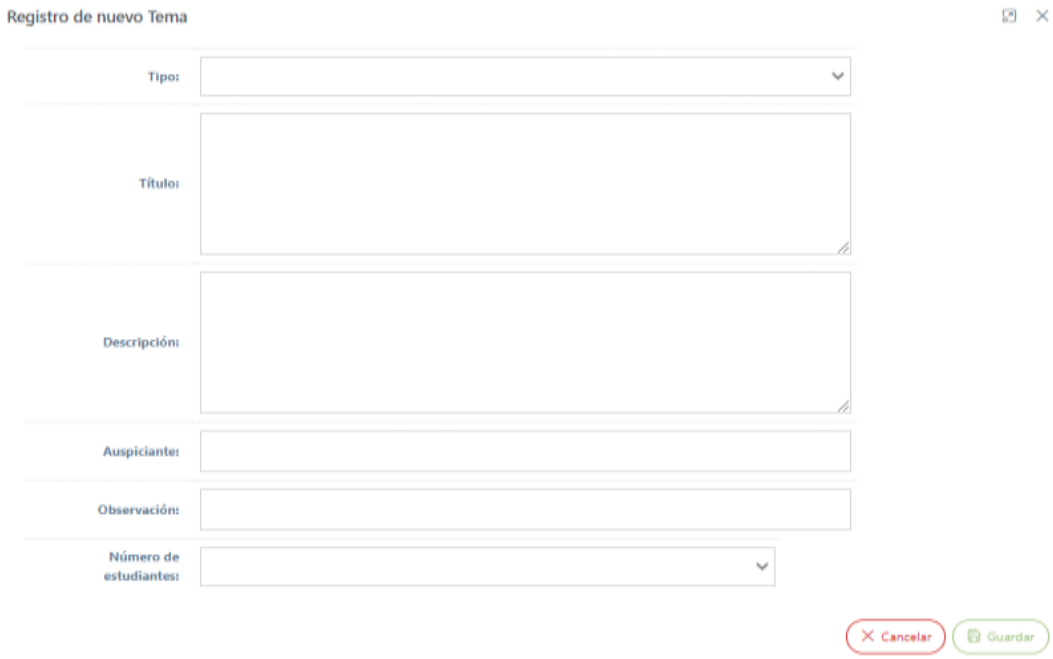


Figura 48. Cuadro de dialogo para crear un nuevo tema de titulación.
Elaborado por: Diego Arias y Roberth Proaño.

En la figura 48, se visualiza el cuadro de dialogo en el cual el docente podrá crear nuevos temas de titulación y agregarlos a la pantalla principal o dashboard.

3.4.3. Perfil CUTS.

Pantalla principal CUTS

Figura 49. Unidad de Titulación.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 49, se visualiza un listado de todos los estudiantes que se han inscrito a la Unidad de Titulación, existiendo dos estados uno al momento de que el estudiante se inscribe y espera la validación de su inscripción y el otro al momento donde su estado a cambiando ha negado o aprobado dependiendo de los requisitos enviados por el estudiante.

Detalle de presolicitud

Figura 50. Cuadro de dialogo visualización de requisitos para ser validado.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 50, se visualiza si el estudiante cumplió con los requisitos solicitados para ingresar a la Unidad de Titulación.

3.4.4. Perfil administrador.

Pantalla principal administrador

Figura 51. Lista de usuarios.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 51, el administrador puede listar todos los usuarios y ver el estado en que se encuentra un usuario en específico.

A su vez el administrador puede agregar un nuevo usuario al sistema como se muestra en la figura 52.

Cuadro de dialogo crear nuevo usuario

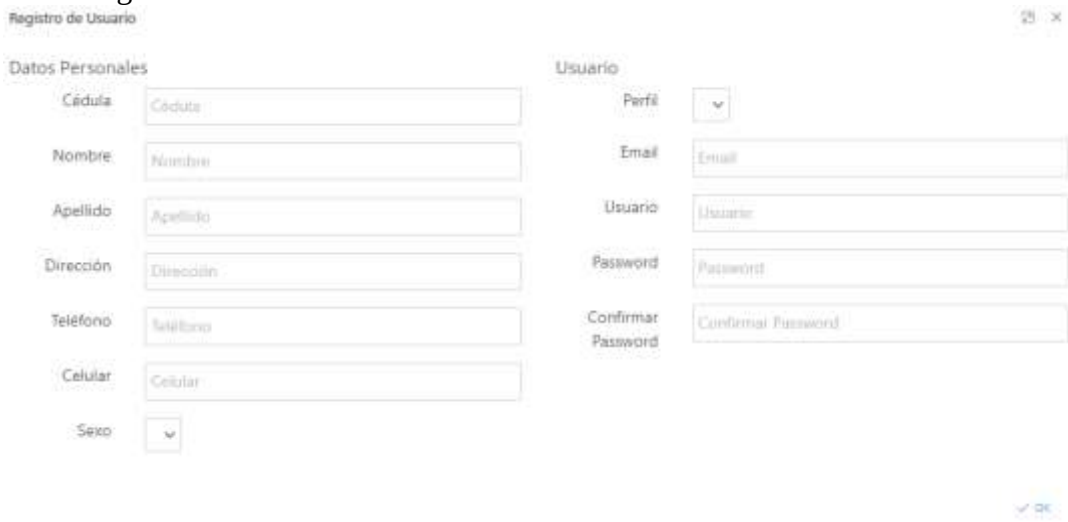


Figura 52. Creación de un nuevo usuario.
Elaborado por: Diego Arias y Roberth Proaño

3.5. Creación de portal de acceso. A continuación, se describe el proceso de diseño y desarrollo del portal de acceso unificado para ingreso a la suite de aplicaciones.

La creación del portal de acceso se desarrolló con el fin de gestionar y controlar una variedad de aplicaciones siendo el propósito agrupar e integrar futuras aplicaciones en un solo entorno, logrando tener un solo punto de acceso.

3.5.1. Arquitectura portal de acceso. Este aplicativo fue desarrollado bajo la arquitectura MVC mediante el uso del framework MEAN Stack, este framework se caracteriza por desarrollar aplicaciones bajo un mismo lenguaje de programación siendo este JavaScript. MEAN Stack utiliza los frameworks: Angular, NodeJS, ExpressJS y MongoDB.

Arquitectura portal de acceso

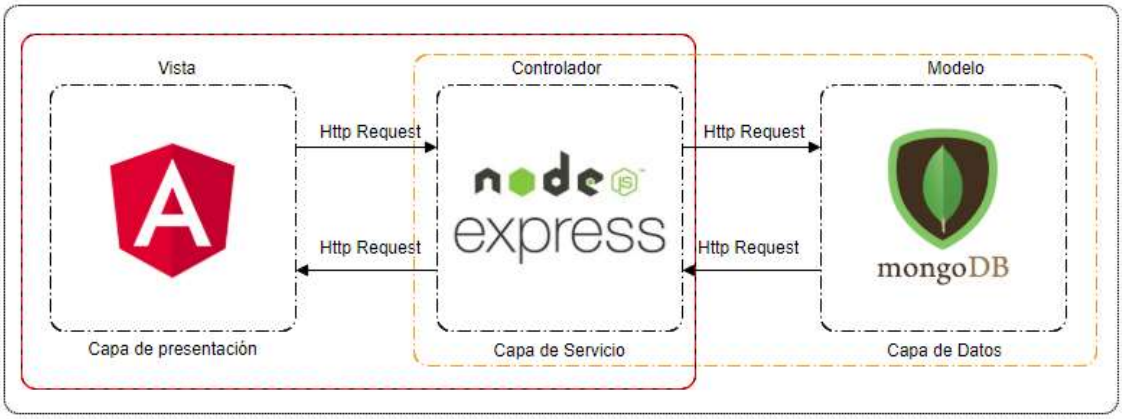


Figura 53. Arquitectura portal de acceso unificado.
Elaborado por: Diego Arias y Roberth Proaño

En la figura 53, se muestra la arquitectura empleada para en el desarrollo del portal de acceso.

3.5.2. Análisis del código Fuente. Back-end. Para la creación del back-end se utilizó NodeJsS y ExpressJS, los cuales facilitaron el desarrollo ya que, al trabajar bajo un mismo lenguaje de programación la sintaxis al programar se mantiene creando así aplicaciones robustas y distribuidas.

Árbol de directorios

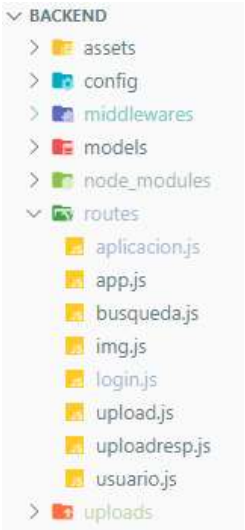


Figura 54. Árbol de directorios.
Elaborado por: Diego Arias y Roberth Proaño

Al desarrollar el back-end se creó un árbol de directorios como se muestra en la figura 54, el cual facilitó la ubicación de los diferentes archivos distribuyendo y encapsulando la lógica del negocio. Uno de estos directorios es “routes” que contiene las funciones de los endpoints que fueron implementadas en el aplicativo.

Función "app.get"

```
app.get('/', (req, res, next) => {
    var desde = req.query.desde || 0;
    desde = Number(desde);
    Usuario.find({}, 'nombre email img role')
        .skip(desde)
        .limit(5)
        .exec(
            (err, usuarios) => {
                if (err) {
                    return res.status(500).json({
                        ok: false,
                        mensaje: 'Error en la carga de Usuarios',
                        errors: err
                    });
                }
                Usuario.count({}, (err, conteo) => {
                    res.status(200).json({
                        ok: true,
                        usuarios: usuarios,
                        total: conteo
                    });
                });
            });
});
```

Figura 55. Función "app.get" obtiene todos los usuarios.
Elaborado por: Diego Arias y Roberth Proaño

En la figura 55, se muestra la función “app.get” que se utilizó para obtener todos los usuarios registrados en el aplicativo

Front-end

Para el desarrollo del front-end se usó Angular que facilitó el desarrollo del aplicativo, de igual manera se creó un árbol de directorios como se muestra en la figura 56, en el cual se segmentó y estructuró todos los archivos según la información que maneja cada uno de ellos.

Árbol de directorios portal de acceso

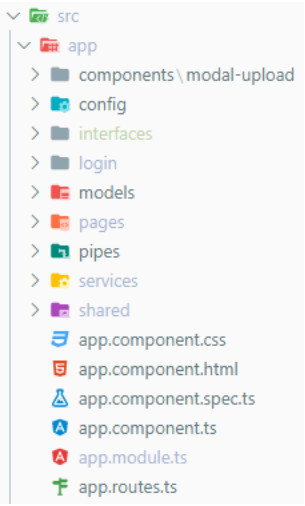


Figura 56. Árbol de directorios portal de acceso.
Elaborado por: Diego Arias y Roberth Proaño

3.5.3. Maquetación de Interfaces. Para la interfaz de usuario se usó una plantilla la cual agilizó el desarrollo y la implementación de las funcionalidades, reduciendo el tiempo de construcción del aplicativo.

A continuación, en la figura 57, se visualiza la pantalla principal de un usuario con perfil administrador. En la pantalla principal se muestran las aplicaciones que fueron agregadas.

Pantalla principal administrador

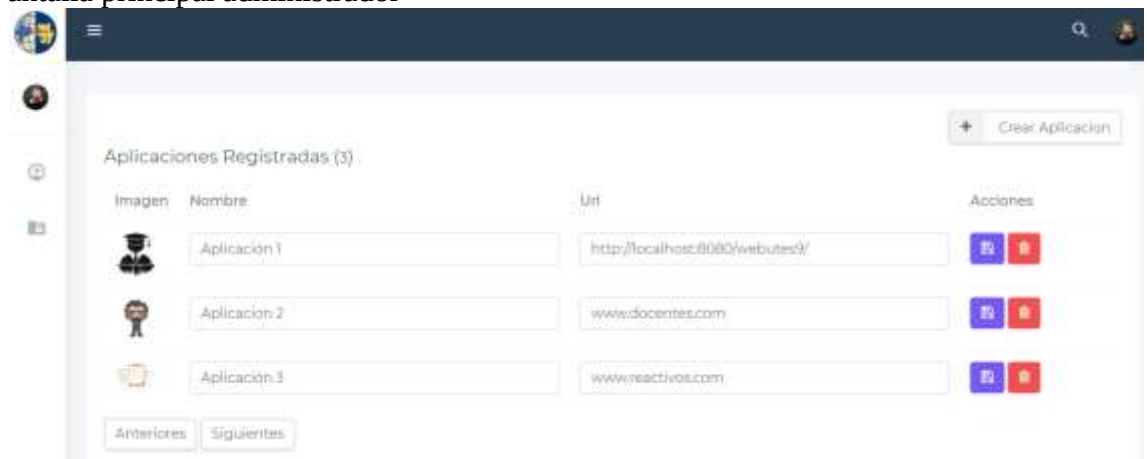


Figura 57. Pantalla principal perfil administrador.
Elaborado por: Diego Arias y Roberth Proaño.

El perfil administrador puede agregar nuevas aplicaciones al igual que nuevos usuarios, mientras que el perfil usuario solo puede hacer uso de las aplicaciones que hayan sido cargadas previamente.

Pantalla principal

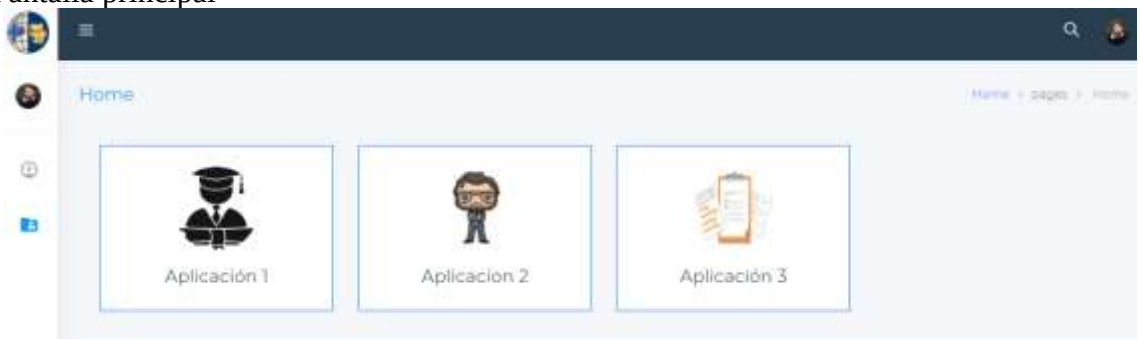


Figura 58. Pantalla principal perfil usuario
Elaborado por: Diego Arias y Roberth Proaño

3.5.4. Implementación de hojas de estilos CSS. Al utilizar el framework Angular este tiene una característica importante que se puede añadir diferentes hojas de estilos ya se de forma general o ir personalizando cada componente dependiendo el uso o tipo de información que este vaya a publicar.

Como se muestra en la figura #, se visualiza el árbol de directorios de las hojas de estilos implementadas para el desarrollo de la refactorización del aplicativo base.

Árbol de directorios de las hojas de estilos.

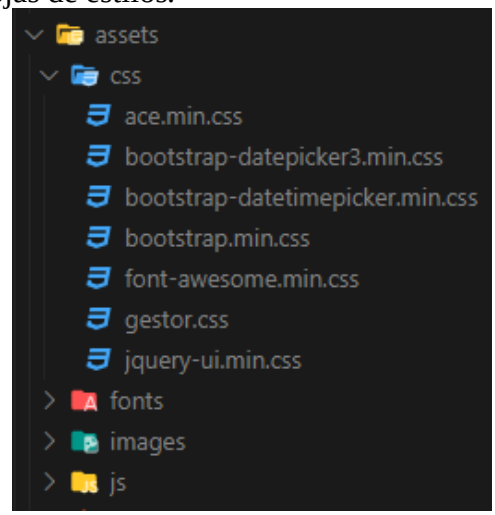


Figura 59. Árbol de directorios de las hojas de estilos.
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 59, en el directorio “assets” se agregaron todas las hojas de estilos que fueron aplicadas en el aplicativo refactorizado para personalizar ciertos componentes, además

se implementó “Bootstrap”, el cual gestiona de mejor manera el cómo se visualiza todos los elementos añadiendo una mejor interfaz hacia el usuario final.

Al implementar Bootstrap esté define y estandariza ciertos aspectos de los componentes como el color, tamaño, tipo de letra y ubicación por nombrar en este caso el estilo de un botón.

De esta manera se reduce el tiempo de creación de los componentes al tener ya definidas las hojas de estilos CSS.

Como se muestra en la figura 60, la implementación de Bootstrap hace que la interfaz de usuario sea aplicada de una manera más fácil el cualquier componente que se desea utilizar y que la interacción con el usuario se mas interactiva.

Archivo "estudiantetema.html"

```
<p-tabView>
    <p-tabPanel header="{{titulo}}" leftIcon="pi pi-bookmark">
        <p-table [value]="listComisionPublicadoEstudiante" #dt dataKey="idPsl"
                 [paginator]="true" [rows]="10"
                 [showCurrentPageReport]="true"
                 [rowsPerPageOptions]="[10,25,50]"
                 styleClass="p-datatable-responsive-demo">
            <ng-template pTemplate="header">
                <tr>
                    <th style="width: 30px;">#</th>
                    <th class="center" style="width: 100px">Acciones</th>
                    <th>Autor</th>
                    <th style="width: 400px;">Titulo</th>
                    <th>Fec Enviado</th>
                </tr>
            </ng-template>
            <ng-template pTemplate="body" let-item let-rowIndex="rowIndex">
                <tr>
                    <td style="width: 30px;">{{ rowIndex + 1 }}</td>
                    <td class="actions center" style="width: 50px">
                        <button class="p-button-raised"
                                pButton type="button"
                                icon="pi pi-search"
                                name="detalle"
                                (click)='openDialog(item)'
                                title="Detalle Registro">
                        </button>
                    </td>
                    <td>{{item.persona.perApellido}} {{item.persona.perNombre}}</td>
                    <td style="width: 400px;">{{item.temNombre}}</td>
                    <td>{{getFecha(item.temFechaEnviado)}}</td>
                </tr>
            </ng-template>
        </p-table>
    </p-tabPanel>
</p-tabView>
```

Figura 60. Archivo "estudianteTema.html".
Elaborado por: Diego Arias y Roberth Proaño

A continuación, en la figura 61, se visualiza el cambio que ha tenido la aplicación refactorizada al implementar Bootstrap, ayudando a mejorar la interfaz de usuario facilitando el ver de una mejor manera los distintos componentes que interactúan en dicha pantalla.

Visualización del archivo “estudianteTema.html”.

Figura 61. Visualización del archivo “estudianteTema.html”.
Elaborado por: Diego Arias y Roberth Proaño

De esta forma quedan establecidos los estilos que fueron utilizados en la aplicación refactorizada y que pueden ser reutilizados para diferentes aplicaciones que se planeen desarrollar a futuro.

3.6. Catálogo de servicios. A continuación, se expone un listado de los servicios más usados por la aplicación; todos los servicios se encuentran en el directorio “restController” del back-end, y un listado completo se puede ver en el (Anexo C).

Tabla 11. Catálogo de servicios aplicación refactorizada

Método Http	**Uri**	**Función**
Post	/api/convocatoria/create	Crea una nueva convocatoria
Get	/api/convocatoria/obtenerConvocatoriaPorId/{id}	Obtiene convocatoria por id
Post	/api/convocatoria/update	Actualiza convocatoria.
Post	/api/hito/create	Crea un nuevo Hito.
Get	/api/hito/obtenerEntidadaHitoxTema/{idtema}	Obtiene entidad hito por tema.
Get	/api/hito/obtenerHito/{id}	Obtiene hito por id.
Get	/api/hito/obtenerHitoxTema/{idtema}	Obtiene hito por tema.
Get	/api/hito/obtenerListadoHito	Obtiene listado de hitos.
Get	/api/hito/obtenerSecuencialHito/{idtema}	Obtienen secuencial hito.
Post	/api/hito/update	Actualizar Histórico.
Get	/api/inscripcion/obtenerInscripcionPorId/{id}	Obtiene inscripción por id.
Get	/api/inscripcion/obtenerInscripcionPorPeriodo/{idperiodo}	Obtiene inscripción por periodo.
Get	/api/inscripcion/obtenerListadoInscripcion	Obtiene listado de inscripciones.
Get	/api/inscripcion/obtenerListadoInscripcionPorEstado	Obtiene listado de inscripción por estado.
Get	/api/inscripcion/obtenerSecuencialInscripcion	Obtiene secuencial de inscripción.
Get	/api/inscripcion/obtenerUltimoRegistroInscripcion	Obtiene ultimo registro de inscripción.
Post	/api/inscripcion/update	Actualiza la inscripción.
Post	/api/persona/créate	Crea una nueva persona
Delete	/api/persona/delete/{id}	Elimina una persona por id.
Get	/api/persona/obtenerListadoPersona	Obtiene listado de persona.
Get	/api/persona/obtenerPersonaPorCedula/{cedula}	Obtiene persona por cedula.
Get	/api/persona/obtenerPersonaPorId/{id}	Obtiene persona por id.
Post	/api/persona/update	Actualizar entidad persona.
Post	/api/tema/consultaDocente	Consulta temas de docente.
Post	/api/tema/consultaDocenteEstado	Consulta docente estado.
Post	/api/tema/create	Crea nuevo tema.

Get	/api/tema/obtenerConsultaTemas/{parametro}	Consulta temas
Get	/api/tema/obtenerTemas	Obtiene temas
Get	/api/tema/obtenerTemasEstados/{codigos}	Obtiene temas estados por codigo.
Get	/api/tema/obtenerTemasxConvocatoria/{fkconv}	Obtiene temas por convocatoria.
Get	/api/tema/obtenerTemasxEstado/{idestado}	Obtiene temas por estado.
Get	/api/tema/obtenerTemasxId/{pktema}	Obtiene temas por pk de tema.
Get	/api/tema/obtenerTemasxIdstemas/{codigos}	Obtiene temas por código.
Get	/api/tema/obtenerTemasxIdstemasEstados/{codigos}/{estados}	Obtiene temas por id tema y estados.
Get	/api/tema/obtenerTemasxIdstemasEstadosPageable/{codigos}/{estados}	Obtiene temas por ids temas y Estados paginable.
Get	/api/tema/obtenerTemasxPk/{idtem}	Obtiene temas por id temas.
Get	/api/tema/obtenerTemasxUsuario/{nameuser}	Obtiene temas por usuario.
Get	/api/tema/obtenerTemasxUsuarioEstadoConvocatoria/{nameuser}/{estados} /{idconvocatoria}/{fresini}/{fresfin}/{finiini}/{finifin}	Obtiene temas por usuario, estado, convocatoria mediante usuario, estado, convocatoria, fechas inicio – fin.
Get	/api/tema/obtenerTemasxUsuarioEstadosIn/{nameuser}/{estados}	Obtiene temas por usuario estado mediante usuario, estado.
Get	/api/tema/obtenerTemaxNombre/{nombretema}	Obtiene tema por nombre de tema.
Post	/api/tema/update/{idTem}	Actualiza temas por id

Nota: Esta tabla contiene todos los servicios de la aplicación refactorizado

CAPÍTULO 4

PLAN DE PRUEBAS

Este plan de pruebas fue desarrollado con el fin de comprobar y verificar que la refactorización realizada a la aplicación base fue realizada correctamente y que cumpla con los requerimientos del dueño del producto.

4. Objetivo del plan de pruebas

Demostrar la calidad de la aplicación refactorizada, mediante la utilización de varias herramientas que permiten encontrar e identificar los errores y solucionarlos para que el aplicativo cumpla los requerimientos establecidos.

4.1. Alcance de las pruebas. Evaluar el aplicativo refactorizado y cada uno de sus módulos para comprobar su correcto funcionamiento, los módulos a evaluar son:

- Módulo estudiante
- Módulo docente
- Módulo CUTS

4.2. Enfoque de pruebas.

4.2.1. Criterios de Entrada. Para empezar la fase de pruebas el aplicativo refactorizado debe cumplir con los siguientes requerimientos.

- Aplicación refactorizada completamente

4.2.2. Criterios de Salida. Los criterios tomados en cuenta serán:

- Mensajes de confirmación
- Mensajes de error
- Botones deshabilitados
- Redireccionamiento

4.3. Ejecución de pruebas. Las pruebas fueron realizadas con diferentes herramientas que permitieron alcanzar los objetivos planteados anteriormente, así mismo se usaron casos pruebas para cada una de las pruebas seleccionadas y así registrar los resultados conseguidos después de la ejecución de cada una de ellas.

4.3.1. Pruebas funcionales.

4.3.1.1. Pruebas de funcionalidad. Para la ejecución de las pruebas de funcionalidad se evaluaron los módulos descritos en el alcance de las pruebas, desarrollando todos los casos pruebas que se encuentran detallados en el (Anexo E), en el cual figuran todas las variantes y los resultados obtenidos de una forma más detallada.

A continuación, se visualiza las pruebas más importantes para comprobar el correcto funcionamiento del aplicativo refactorizado y los resultados obtenidos.

Tabla 12. Caso de pruebas inscripción estudiante

Proceso	**Actor**	**Escenario**	**Resultados**	**Resultados Esperados**
Inscripción	Estudiante	Registro al sistema	Guarda correctamente	OK
		Ingreso al Sistema	Ingreso al sistema	OK
		Inscripción a una opción de titulación (Proyecto, Articulo o Examen Complexivo)	Guarda y visualiza la inscripción realizada	OK
	CUTS	Validación inscripción	Visualiza la inscripción y valida la misma	OK
	Estudiante	Verificación Estado inscripción	Visualiza el estado de la inscripción realizada	OK

Nota: Esta tabla contiene los resultados obtenidos al ejecutar las pruebas funcionales sobre dicho proceso

Como se muestra en la tabla 12, se visualiza el proceso que un estudiante debe seguir para registrarse e inscribirse a la Unidad de Titulación y los resultados obtenidos al ejecutar dicha prueba.

Tabla 13. Caso prueba publicación tema

Proceso	Actor	Escenario	Resultados	Resultados Esperados
Publicación Tema	Docente autor	Crear tema	Guarda correctamente el tema creado ya se proyectó o artículo académico	OK
		Enviar tema	Se visualiza el cambio de estado al enviar tema	OK
	CUTS	Asignación revisor tema	Cambia de estado "Asignado"	OK
	Docente revisor	Revisión tema	Asignar estado: Revisado en revisión	OK
	CUTS	Valida tema	Se visualiza los temas publicados	OK
	Docente autor	Publica tema	Visualización del cambio de estado tanto en CUTS y docente autor	OK

Nota: Esta tabla contiene los resultados obtenidos al ejecutar las pruebas funcionales sobre dicho.

Como se muestra en la tabla 13, se visualiza el proceso que para la publicación de temas y los resultados obtenidos al ejecutar las pruebas.

4.3.2. Pruebas no funcionales.

4.3.2.1. Pruebas de usabilidad. Para evaluar la usabilidad del aplicativo refactorizado se utilizó la herramienta Sirius, que ayudó a comprobar el correcto funcionamiento mediante la aplicación de criterios los cuales se encuentra detallados en el (Anexo E).

Porcentaje de usabilidad

CÁLCULO DEL PORCENTAJE DE USABILIDAD

Porcentaje de usabilidad*	91,88

Datos para el cálculo:

Nº de criterios evaluados	63	
El sumatorio de los valores de relevancia de los criterios evaluados es:	165	*A los NA no se les aplica relevancia
Sumatorio de la columna i de cada criterio	9,1878787879	*Los NA no se tienen en cuenta
Sumatorio de la columna h *10 de cada criterio	10,0000000000	*Los NA no se tienen en cuenta

Figura 62. Porcentaje de usabilidad de la aplicación refactorizada.
Elaborado por: Diego Arias y Roberth Proaño.

Como se muestra en la figura 62, el porcentaje alcanzando es de 91.88% para obtener este resultado se utilizó una muestra de quince personas, las cuales manipularon la aplicación refactorizada y brindaron su calificación. De esta manera se obtuvo el porcentaje antes mencionado.

A comparación con el aplicativo base, el resultado fue mayor ya que, se agregaron funcionalidades al realizar la búsqueda y visualización de información siendo esto un plus al aplicativo refactorizado.

4.3.2.2. Pruebas de rendimiento y estrés. Para la ejecución de las pruebas tanto de rendimiento como de estrés se utilizó la herramienta JMeter ya que, esta facilita la puesta a prueba de un aplicativo generando una simulación de trabajo sobre el software al crear una prueba para cada módulo, en este caso las pruebas se realizaron a los módulos mencionados en el alcance de las pruebas.

Para la ejecución de estas pruebas se tomaron en cuenta los siguientes datos y nomenclatura:

Tabla 14. Datos y nomenclatura a usar para la ejecución de pruebas

Datos			
Grupos de hilos	Estudiante	Docente	CUTS
Carga inicial	100	40	10
Carga máxima	200	120	40
Carga estrés	650	250	170
Nomenclatura			
# Usuarios	Número de usuarios		
Grupos de hilos	Módulos sujetos a la ejecución de pruebas.		
CI	Carga inicial: número de usuarios que interactuaran en el aplicativo.		
CM	Carga máxima: número máximo de usuarios que interactuaran en el aplicativo.		
CE	Carga estrés: número de usuarios punto de quiebre en el aplicativo.		
Muestra	Número de peticiones realizadas		
Media	Tiempo promedio de ejecución en milisegundos		
Rendimiento	Peticiones realizadas segundo/minuto/hora		
m/s	Milisegundo		
S	Segundos		

Nota: Esta tabla contiene los datos y la nomenclatura a utilizar en la ejecución de las pruebas de rendimiento.

Pruebas de rendimiento

Para la ejecución de estas pruebas se tomó los datos de la tabla 14.

Tabla 15. Los resultados obtenidos en la prueba de rendimiento

	Estudiante		Docente		CUTS	
	CI	**CM**	**CI**	**CM**	**CI**	**CM**
N° Usuarios	100	200	40	120	10	40
Muestra	4800	6000	1200	3600	517	945
Media	3327 ms	4036 ms	1232 ms	3007 ms	605 ms	1086 ms
Rendimiento	25.7 s	32.6 s	18.1 s	22.3 s	15.9 s	18.7 s

Nota: Esta tabla contiene los resultados de la ejecución de las pruebas de rendimiento.

En la tabla 15, se visualiza los resultados de las pruebas de rendimiento obtenidos. Al ejecutar las pruebas de rendimiento y realizar una comparación con los resultados obtenidos del aplicativo base con el aplicativo refactorizado se logró visualizar una mejora y reducción en el tiempo promedio de ejecución de peticiones dicho tiempo no sobrepasa los cinco segundos, por lo que el aplicativo refactorizado mejoró su tiempo de respuesta.

Gráfica en barras de los resultados

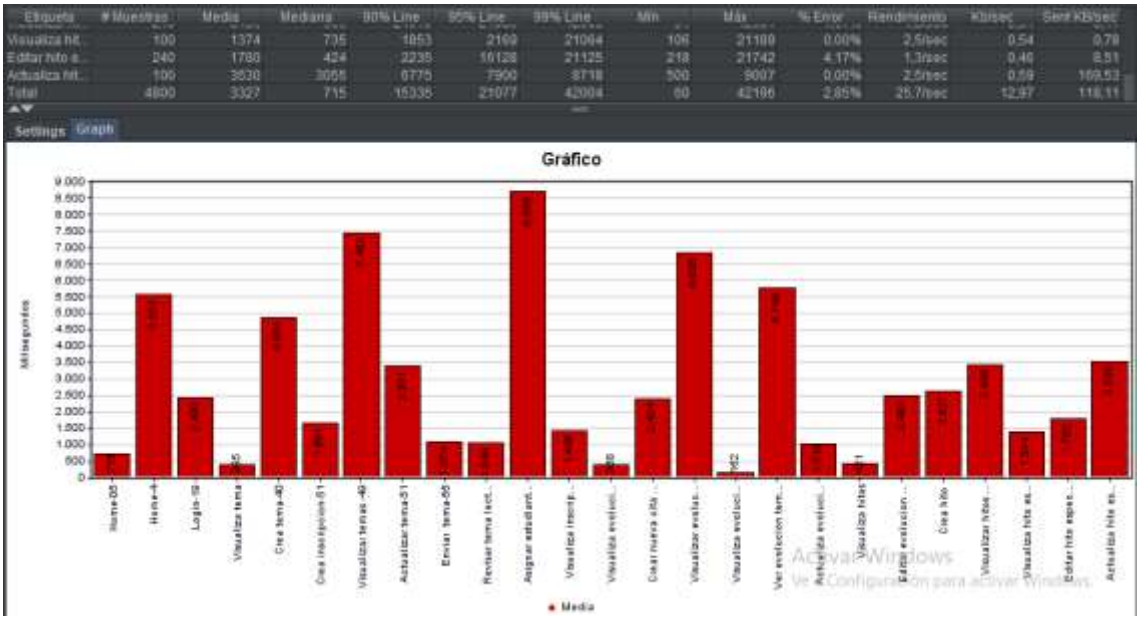


Figura 63. Gráfica en barras de los resultados obtenidos de las pruebas módulo estudiante con cien usuarios
Elaborado por: Diego Arias y Roberth Proaño

Como se muestra en la figura 63, se visualizan los valores obtenidos al ejecutar las pruebas de rendimiento con una carga de cien usuarios, en este gráfico se visualiza el tiempo que se demora en ejecutar cada petición el eje x representa el tiempo en milisegundos y el eje y las peticiones ejecutadas, entre las que se puede nombrar se encuentra: home, login, visualización de temas, etc., siendo el módulo del perfil estudiante.

La comparación de los resultados y un detalle más completo se puede observar en el (Anexo F).

Pruebas de estrés

La ejecución de estas pruebas se realizó con el objetivo de buscar e identificar el punto de quiebre del aplicativo refactorizado, para la ejecución de estos casos prueba se tomó los datos de la tabla 14.

Para ejecutar las pruebas de estrés se aumentó el número de usuarios en un 8.33% al valor que fue tomado de referencia del aplicativo base dando un aumento de 50 usuarios a cada grupo de hilos para la ejecución de dichas pruebas. Este aumento de usuarios se realizó con el fin de comprobar el correcto funcionamiento del aplicativo ante una concurrencia alta de usuarios.

Tabla 16. Los resultados obtenidos en la prueba de estrés.

	Estudiante		**Docente**		**CUTS**	
	CI	CE	CI	CE	CI	CE
N° Usuarios	100	600	40	200	10	170
Muestra	4800	10800	1200	6000	520	3120
Media	3327 ms	7326 ms	1232 ms	4414 ms	603 ms	2166 ms
Rendimiento	25.7 s	58.8 s	18.1 s	35.4 s	15.6	47.9 s

Nota: Esta tabla contiene los resultados de la ejecución de las pruebas de estrés.

Al ejecutar las pruebas de estrés y realizar una comparación con los resultados obtenidos del aplicativo base vs el aplicativo refactorizado se logró visualizar una reducción en el tiempo promedio de ejecución de peticiones, dicho tiempo no sobrepasa los diez segundos. El rendimiento aumentó ya que el número de muestras se incrementó de igual manera en este caso el rendimiento hace referencia al tiempo que se demoró en procesar el total de muestras.

El punto de quiebre del aplicativo refactorizado no se logró, pero se puedo comprobar que al aumentar el número de usuarios el aplicativo siguió trabajando de forma correcta.

La comparación de los resultados y un detalle más completo se puede observar en el (Anexo F).

CONCLUSIONES

- Se realizó el análisis de los estilos, funcionalidad y estructura de datos de la aplicación base, lo cual facilitó el proceso de refactorización ya que, sin realizar este análisis previo no hubiera sido posible la refactorización.
- Al desarrollar la refactorización de la aplicación base se estableció un estilo de diseño (CSS), el cual puede ser utilizado para el desarrollo de futuras aplicaciones, logrando tener un estilo unificado para todas, evitando así tener varias aplicaciones con estilos y diseños diferentes.
- Se creó el portal de acceso el cual permite el ingreso unificado a un grupo de aplicaciones en si se creó un solo punto de acceso para agrupar aplicaciones que estén dispersas.
- Se realizó la refactorización de la aplicación web Registro y Seguimiento de Opciones de Titulación, obteniendo una nueva aplicación con características importantes como la capacidad de interconectarse con otras aplicaciones y la facilidad de agregar nuevas funcionalidades además de otras características como la modularidad, escalabilidad y mantenibilidad mejorando así su estructura sin afectar las funcionalidades originales.
- Se diseño y ejecutó el plan de pruebas que comprobó y validó el correcto funcionamiento del aplicativo refactorizado con lo cual, se determina que al aplicar los cambios en la estructura del aplicativo se mejora considerablemente el tiempo de espera lo cual se logró con los frameworks utilizados.

RECOMENDACIONES

- Antes de empezar un proceso de refactorización se debe tener claro el concepto de del mismo ya que, este proceso se lo realiza con el objetivo de repotenciar un software más no cambiar su funcionamiento.
- Para ejecutar el aplicativo se recomienda revisar el manual de instalación ya que, en este documento se detalla los pasos que se deben seguir para ejecutar el aplicativo de forma correcta.
- Se recomienda sacar respaldos de la base de datos periódicamente ya que, con esto se podrá tener un mejor control de la información que se va añadiendo y los cambios que se van ejecutando.
- Para ejecutar el proyecto se debe tener mínimo la versión 10 de Angular ya que, si esta versión es menor causara problemas al ejecutar el aplicativo por lo que utiliza librerías actualizadas y optimizadas a la versión antes mencionada.

LISTA DE REFERENCIAS

Angular.io. (Enero de 2020). *www.angular.io*. Obtenido de https://angular.io/docs

Apache Software Foundation. (Enero de 2020, p. 07). *http://tomcat.apache.org/*. Obtenido de http://tomcat.apache.org/

Atlassian. (2020, p. 26). *Atlassian*. Obtenido de Jira Software: https://www.atlassian.com/es/software/jira

Bbvaapimarket. (Enero de 2020, p. 08). *https://bbvaopen4u.com/*. Obtenido de https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

Cáceres Alvarez, M. A. (2010, p. 37). Modelo de programación asíncrona para Web transaccionales. *Ingeniare*, 37. Obtenido de https://scielo.conicyt.cl/scielo.php?pid=S0718-33052011000100004&script=sci_arttext&tlng=en

Caicedo, S. M. (2008, p. 06). Process Integration Making Use of Service Oriented Architecture SOA. *Scientia et Technica*, 6.

Caso, I. N. (Septiembre de 2004). *http://apit.wdfiles.com/.* Obtenido de http://apit.wdfiles.com/local--files/start/02_apit_scrum.pdf

Developers, G. (Enero de 2020, p. 12). *https://developers.google.com*. Obtenido de https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications

Díaz, S. M. (2004, p.23). *Ingeniería de la web y patrones de diseño.* Madrid: Pearson Educación.

Flores, M., & Quisupangui, D. (2016). *Desarrollo de una aplicación web para el registro y seguimiento de opciones de titulación* . Quito.

Fowler, B. (2018, p.13). *Refactoring.* Melrose, Massachusetts: Pearson Addison.

Franco. (Septiembre de 2010). *https://pdfs.semanticscholar.org/.* Obtenido de https://pdfs.semanticscholar.org/8658/38b05d87b9b2f306af6d677385b0357785aa.pdf

Gallego, M. T. (2012). *http://openaccess.uoc.edu/.* Obtenido de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf

genbeta. (18 de 07 de 2020). *https://www.genbeta.com*. Obtenido de https://www.genbeta.com/desarrollo/spring-framework-el-patrn-dao

GitHub. (Enero de 2020). *https://github.com*. Obtenido de https://github.com/git/git

Hilliard, R. (Noviembre de 2000, p. 11). *https://pdfs.semanticscholar.org.* Obtenido de https://pdfs.semanticscholar.org/e74f/0b588e95d5e3c5c5176617026f48a63655d5.pdf

https://www.ionos.es/. (12 de 06 de 2020). *https://www.ionos.es/*. Obtenido de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-la-refactorizacion/

Kinsta. (Enero de 2020, p. 30). *https://kinsta.com/*. Obtenido de https://kinsta.com/es/base-de-conocimiento/que-es-github/

Marini, I. E. (Octubre de 2012, p. 28). *https://www.linuxito.com.* Obtenido de https://www.linuxito.com/docs/el-modelo-cliente-servidor.pdf

Medina, J. (2014). *Pruebas de rendimiento TIC.* Murcia: Ediciòn Kindle. Obtenido de Empezando con Apache JMeter: https://riptutorial.com/es/jmeter

Mikowski, M., & Powell, J. (2014). *Single Page Web Applications.* New York, US: Manning Publications Co.

Nextech Education Center. (Febrero de 2020, p. 34). *Nextech Education Center*. Obtenido de https://nextech.pe/que-es-bpmn-y-para-que-sirve/

NodeJs. (03 de Enero de 2020, p. 09). *Acerca*. Obtenido de nodejs.org: https://nodejs.org/es/about/

Paz, J. A. (2016, p. 13). *https://repository.ucc.edu.co/.* Obtenido de https://repository.ucc.edu.co/bitstream/20.500.12494/962/1/Pruebas.pdf

PostgreSQL, T. (Enero de 2020, p. 05). *https://www.postgresql.org*. Obtenido de https://www.postgresql.org/about/

Presedo Varela, N. R. (2010). Accediendo a Base de Datos desde aplicaciones Web desarrolladas con J2EE: patrones de diseño. 7.

Profile. (2020, p. 12). *https://profile.es/*. Obtenido de https://profile.es/blog/patrones-de-diseno-de-software/

Qualitydevs. (enero de 2020). *https://www.qualitydevs.com/*. Obtenido de https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

RJCodeAdvance. (2020, p. 11). *https://rjcodeadvance.com/*. Obtenido de https://rjcodeadvance.com/patrones-de-software-que-es-patron-de-diseno-parte-2/

S.L.U., I. E. (Enero de 2020). *https://www.ionos.es/.* Obtenido de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/spring-boot-tutorial/

Shvets. (08 de 2020, p.10). *https://feedback.refactoring.guru*. Obtenido de https://refactoring.guru/es/design-patterns/singleton

SmartBear, S. (02 de 2020, p. 02). *https://swagger.io*. Obtenido de https://swagger.io

Torrente, M. d. (2011). *SIRIUS: Sistema de evaluación de la usabilidad web orientado al usuario y basado en la determinación de tareas críticas.* Oviedo.

UNIVERSIDAD POLlTÉCNICA SALESIANA. (2015). *https://www.ups.edu.ec/.* Obtenido de https://www.ups.edu.ec/normativa/-/document_library_display/u8OILw1nqXw9/viewf/9845597

Visual Paradigm. (12 de 06 de 2020). *https://online.visual-paradigm.com*. Obtenido de https://online.visual-paradigm.com/es/about-us/

VMware. (Enero de 2020, p. 15). *https://spring.io/.* Obtenido de https://spring.io/web-applications

ANEXOS

Anexo A: Spring y grafica de seguimiento de tareas.

Anexo B: Análisis de estilos CSS.

Anexo C: Catálogo de servicios.

Anexo D: Plan de pruebas funcionales.

Anexo E: Casos de pruebas funcionales

Anexo F: Casos de pruebas no funcionales

Anexo G: Manual de instalación

Anexo H: Manual de usuario

Para revisar los anexos de este trabajo, diríjase al CD.