RESUMEN

APLICACIÓN WEB PARA LA GESTIÓN DEL PLAN ESTRATÉGICO DE LA UNIVERSIDAD DE MONTEMORELOS. CASO DE ESTUDIO: FACULTAD DE INGENIERÍA Y TECNOLOGÍA

por

Kevin Arody Aguilar Ruiz

Asesor principal: Germán Harvey Alférez Salinas

RESUMEN DE TESIS DE MAESTRÍA

Universidad de Montemorelos

Facultad de Ingeniería y Tecnología

Título: APLICACIÓN WEB PARA LA GESTIÓN DEL PLAN ESTRATÉGICO DE LA UNIVERSIDAD DE MONTEMORELOS. CASO DE ESTUDIO: FACULTAD DE INGENIERÍA Y TECNOLOGÍA

Nombre del investigador: Kevin Arody Aguilar Ruiz

Nombre y título del asesor principal: Germán Harvey Alférez Salinas, Doctor en Informática

Fecha de terminación: diciembre de 2021

Problema

En la Universidad de Montemorelos como en varias organizaciones se trabaja a través de un plan estratégico que determina los objetivos a alcanzar para los siguientes años de la institución. A pesar de la importancia que tiene la planeación estratégica en el futuro de la institución, la Universidad de Montemorelos no cuenta con una herramienta para la gestión de su plan estratégico.

Actualmente, existen herramientas que ayudan a las organizaciones a gestionar proyectos, planes estratégicos, planes de negocios y proyecciones financieras, entre otros aspectos. Sin embargo, estas herramientas ofrecen recursos insuficientes para el contexto institucional tales como la intervención en el análisis de la misión, la visión,

el análisis FODA (fortalezas, oportunidades, debilidades y amenazas) por parte del equipo, la declaración de objetivos y actividades, y las reuniones de rendición de cuenta.

Método

El desarrollo de este proyecto está basado en una metodología organizada en siete etapas que siguen un proceso iterativo incremental, compuesto de la siguiente manera: conceptualización de la idea, análisis de requisitos, diseño de software, desarrollo de la aplicación, pruebas previas, despliegue de la herramienta y mantenimiento.

Resultados

Siguiendo la metodología descrita, se obtuvo un producto de software, esto es una aplicación web para la gestión del plan estratégico de la Universidad de Montemorelos. La aplicación web contiene las secciones de intervención en la misión, la visión, el análisis FODA, la declaración de objetivos y actividades, así como el tablero de control.

Conclusiones

Por medio de la aplicación web es posible realizar la gestión del plan estratégico de la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos con base en la declaración de misión y visión, el análisis FODA, la declaración de objetivos y actividades y el seguimiento por medio de un tablero de control.

Universidad de Montemorelos

Facultad de Ingeniería y Tecnología

APLICACIÓN WEB PARA LA GESTIÓN DEL PLAN ESTRATÉGICO DE LA UNIVERSIDAD DE MONTEMORELOS. CASO DE ESTUDIO: FACULTAD DE INGENIERÍA Y TECNOLOGÍA

Tesis
presentada en cumplimiento parcial
de los requisitos para el título de
Maestría en Ciencias Computacionales

por

Kevin Aguilar Ruiz

Diciembre de 2021

APLICACIÓN WEB PARA LA GESTIÓN DEL PLAN ESTRATÉGICO DE LA UNIVERSIDAD DE MONTEMORELOS. CASO DE ESTUDIO: FACULTAD DE INGENIERÍA Y TECNOLOGÍA

Proyecto

presentado en cumplimiento parcial de
los requisitos para el grado de
Maestría en Ciencias
Computacionales

por

Kevin Arody Aguilar Ruiz

APROBADO POR LA COMISIÓN:

Germán Harvey Alférez

Dr. Germán Harvey Alférez Salinas
Asesor principal

Mtro. Jaime Alcántara Quiroga
Asesor externo

Mtro. Alejandro Walterio García Mendoza
Miembro

Dr. Ramón Andrés Díaz Valladares
Director de Posgrado e Investigación

Gutiérrez Colorado

Mtro. Daniel Arturo Gutiérrez Colorado
Miembro

Fecha de aprobación
22 de abril de 2022

DEDICATORIA

Agradezco a Dios, por su infinito amor y sabiduría.

A mi familia, por su apoyo incondicional en todo momento.

A mis amigos, por mostrarme la alegría de compartir momentos juntos.

A mis maestros y mentores en la Universidad de Montemorelos, por creer en mí y mostrarme que soy capaz de hacer más de lo que creo posible.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RECONOCIMIENTOS

A Dios, por su compañía, dirección, instrucción y amor constantes.

Al Dr. Germán Harvey Alférez Salinas, asesor principal y mentor, por su dedicación, apoyo continuo, dirección y paciencia en la realización de este trabajo y porque creyó en mi en todo momento.

CAPÍTULO I

INTRODUCCIÓN

Planteamiento del problema

Toda institución tiene una identidad y un objetivo en común a lograr. Cada una de las actividades que se realicen durante las diferentes etapas deben estar enfocadas en el cumplimiento de la misión que la institución tiene, trabajando de manera colaborativa. Esto implica que en la mente de todos los que laboran en alguna institución deben estar claros, tanto los objetivos como las estrategias para alcanzarlos.

Según la investigación descrita en el artículo *The office of strategy management* de Kaplan y Norton (2005) en la *Harvard business review*, el 85% de los equipos de liderazgo ejecutivo dedican menos de una hora al mes a discutir la estrategia, y el 50% no dedica nada de tiempo. La investigación también revela que, en promedio, el 95% de los empleados de una empresa no comprenden su estrategia. No es de extrañar, entonces, que el 90% de las empresas no logren sus objetivos estratégicos.

Por otro lado, en el libro *The 4 disciplines of execution*, los autores McChesney, Covey y Huling (2012) presentan cuatro razones principales, descritas a continuación, por las que la estrategia falla.

1. Los empleados no comprenden la estrategia corporativa. Si los empleados no conocen o no comprenden la estrategia de su empresa, no se puede esperar que sepan cómo pueden contribuir a su ejecución a través de su trabajo diario.

2. La organización no sabe cómo ejecutar la estrategia. La mayoría de las personas no conocen las actividades que generarán resultados.

3. Las organizaciones no realizan un seguimiento del progreso. Si no se implementan medidas para rastrear el progreso, la ejecución de la estrategia corporativa decae.

4. Las personas no son responsables de realizar actividades estratégicas. Las medidas utilizadas para responsabilizar a los empleados (y líderes) deben estar conectadas con el éxito estratégico.

En la Universidad de Montemorelos (UM), como en varias organizaciones, se trabaja a través de un plan estratégico que determina los objetivos a alcanzar para los siguientes años de la institución. A pesar de esto, en la UM no existe una herramienta en la cual se pueda gestionar el plan estratégico. Actualmente, existen herramientas que ayudan a las organizaciones con la administración de proyectos, planes de negocios y proyección financiera, entre otras. Sin embargo, estas herramientas son insuficientes para el contexto institucional por sus funcionalidades limitadas.

En esta investigación se plantea la creación de una aplicación web que ayude a gestionar el plan estratégico de la Universidad de Montemorelos. Este proyecto está basado en los principios de planeación estratégica de Brantley, Jackson y Cauley (2015) y de McChesney, Covey y Huling (2012).

Declaración del problema

En nuestro mundo altamente conectado, la mayoría de los administradores deben tratar con la tecnología para obtener una ventaja competitiva en sus proyectos. Prácticamente todos los segmentos de la industria y el gobierno intentan aprovechar

la tecnología para mejorar la efectividad, el valor y la velocidad. Los procesos de trabajo tradicionales ya no son suficientes, sino que se están reemplazando gradualmente por diseños alternativos de organización y nuevas técnicas de gestión más ágiles (Harold, 2017).

Hoy en día existen varias herramientas que ayudan a los equipos de trabajo de una empresa a ser más productivos en su labor a través de la comunicación entre sus integrantes. Estos sistemas son utilizados en el mundo laboral tecnológico de este tiempo para ayudar a los empleados en su labor diaria. Sin embargo, esto puede llevar a la ejecución de actividades fútiles o que carecen de importancia a futuro. Es por esto que los proyectos centrados en iniciativas estratégicas son fundamentales para el éxito a largo plazo de una organización (Meredith, Shafer y Mantel, 2018).

Otro punto importante que se suele pasar por alto es la diversidad de operación de las instituciones. Los sistemas de planificación estratégica deben estar diseñados para adaptarse a las características únicas de cada organización. Como cada organización difiere en algunos aspectos de todas las demás, se deduce que los sistemas de planificación de las organizaciones difieren entre sí (Steiner, 1979).

A pesar de la importancia de la planeación estratégica, la Universidad de Montemorelos no cuenta con una herramienta para la gestión de su plan estratégico. Aunque existen aplicaciones web para la gestión de proyectos y para la gestión del plan estratégico, estas herramientas no son adecuadas para las necesidades de la institución ya que ofrecen únicamente características básicas de gestión, altos precios por servicios más completos y recursos insuficientes para el contexto institucional tales como la intervención en el análisis de la misión, la visión, el análisis FODA (fortalezas,

oportunidades, debilidades y amenazas) por parte del equipo, la declaración de objetivos y actividades y las reuniones de rendición de cuenta.

Justificación

Tener un plan estratégico es esencial para una organización exitosa. La planificación estratégica responde a las siguientes preguntas: ¿dónde se encuentra la organización ahora? ¿hacia dónde va? ¿cómo llegará allí? En pocas palabras, un plan estratégico es el mapa formalizado que describe la forma en la que una organización ejecuta la estrategia elegida. Un plan detalla a dónde irá una organización a futuro y cómo se llegará allí. También se puede considerar al plan estratégico como una herramienta de gestión que sirve para ayudar a una organización a hacer un mejor trabajo al enfocar la energía, los recursos y el tiempo de todos los empleados en la misma dirección.

A pesar de la importancia de la planeación estratégica, la Universidad de Montemorelos no tiene una herramienta informática para gestionar dicho plan. Además, aunque existen herramientas informáticas comerciales para la gestión del plan estratégico, no son adecuadas para las necesidades específicas de la institución. Por ejemplo, para la intervención en el análisis de la misión, la visión, el análisis FODA por parte del equipo, la declaración de objetivos y actividades y las reuniones de rendición de cuenta.

Debido a la particularidad de la Universidad de Montemorelos como institución educativa adventista, se han adoptado los conceptos de planeación estratégica de Brantley et al. (2015) y de McChesney et al. (2012) para la creación de la aplicación web.

Objetivos

Objetivo general

Crear una aplicación web para la creación y gestión del plan estratégico de la Universidad de Montemorelos.

Objetivos específicos

1. Construir una aplicación web para la definición y gestión del plan estratégico de la Universidad de Montemorelos basada en los conceptos de Brantley et al. (2015) y de McChesney et al. (2012).

2. Elaborar un caso de estudio para evaluar la aplicación web implementándolo a la gestión del plan estratégico de la Facultad de Ingeniería y Tecnología.

Hipótesis

La definición y gestión de objetivos y actividades del plan estratégico mediante una aplicación web tiene un efecto positivo en su cumplimiento en el contexto de la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos.

Limitaciones

Como limitaciones se encuentran temas de cultura institucional. De manera específica, en el compromiso e involucramiento de los líderes directivos, la facilitación en el desarrollo de las habilidades a todos los niveles del personal de forma que el aprendizaje forme parte de la actividad habitual de los empleados y el fomento en la cooperación con los compañeros pueden afectar el seguimiento del plan estratégico.

Delimitaciones

Se delimitará la creación de la aplicación web para la administración de los objetivos y las actividades del plan estratégico de la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos.

Definición de términos

Es común encontrar en la literatura palabras o términos desconocidos. A continuación, se han seleccionado términos no comunes presentes en este trabajo con la finalidad de contextualizar al lector.

API: es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.

Backend: es la parte de un sistema informático o una aplicación a la que el usuario no accede directamente, normalmente responsable de almacenar y manipular los datos.

Framework: es una plataforma de desarrollo de software que al proporcionar un estándar de creación e implementación facilita el desarrollo de aplicaciones, productos y soluciones de software.

Frontend: es la parte de un sistema informático o aplicación con la que el usuario interactúa directamente.

HyperText Markup Language (HTML): el lenguaje de marcado de hipertexto es un estándar que define una estructura básica para la elaboración de páginas web.

Java: es un lenguaje de programación diseñado para producir programas que se ejecuten en cualquier sistema informático.

Software: es el conjunto de instrucciones y programas que permiten a la computadora realizar determinadas tareas.

Unified Modeling Language (UML): el lenguaje unificado de modelado es un lenguaje de modelado de sistemas que utiliza notaciones gráficas para expresar el diseño de proyectos de software.

CAPÍTULO II

MARCO TEÓRICO

Introducción

En esta sección se presenta el marco teórico en donde se explican tanto los conceptos de planeación estratégica, como de ingeniería de software que se han utilizado para el desarrollo de la presente investigación.

Conceptos de planeación estratégica

En la Figura 1 se muestran los conceptos del proceso de planeación estratégica que se utilizaron en esta investigación. Entre ellos se encuentran los siguientes conceptos: misión, visión, análisis FODA, objetivos, actividades, indicadores clave de rendimiento (KPI) y tablero de control. Cada uno de los conceptos está relacionado entre sí y forma parte de lo que es la planeación estratégica.

Planeación estratégica

La planeación estratégica es un enfoque deliberativo y disciplinado para producir decisiones y acciones fundamentales que dan forma y guían lo que es una organización, lo que hace y por qué lo hace. De esta manera, la planificación estratégica es un conjunto de conceptos, procedimientos y herramientas que deben adaptarse cuidadosamente a las situaciones para lograr los resultados deseados (Bryson, 2015).

También se puede definir la planeación estratégica como un proceso formal diseñado para ayudar a una organización a identificar y mantener una alineación óptima

con los elementos más importantes de su entorno (Rowley, Lujan y Dolence, 1997). Finalmente, Steiner (1979) define la planeación estratégica como la identificación sistemática de las oportunidades y amenazas que se podrían presentar en el futuro, los cuales junto con otros datos relevantes proporcionan la base para que en las organizaciones se tomen mejores decisiones en el presente.

Declaración de la misión

La declaración de la misión es una expresión general e intemporal del propósito y aspiración de la organización que aborda, tanto lo que se busca lograr como la manera en que se logrará. En otras palabras, es una declaración de por qué existe la organización (McChesney et al., 2012; Steiner, 1979).

Declaración de la visión

La declaración de la visión es una breve y concisa declaración del futuro de la organización que responde a la pregunta: cómo será la empresa dentro de cinco años o más (McChesney et al., 2012; Steiner, 1979).

Análisis FODA

El análisis FODA (fortalezas, oportunidades, debilidades y amenazas) es una vista resumida de la posición actual de una empresa o persona, específicamente se enfoca en el estudio de las fortalezas, debilidades, oportunidades y amenazas (Riquelme Leiva, 2016; Steiner, 1979). Además, el análisis FODA puede verse como una herramienta que posibilita conocer y evaluar las condiciones de operación reales de una organización, a partir del análisis de esas cuatro variables principales, con el fin de proponer acciones y estrategias para su beneficio (Ramírez Rojas, 2017).

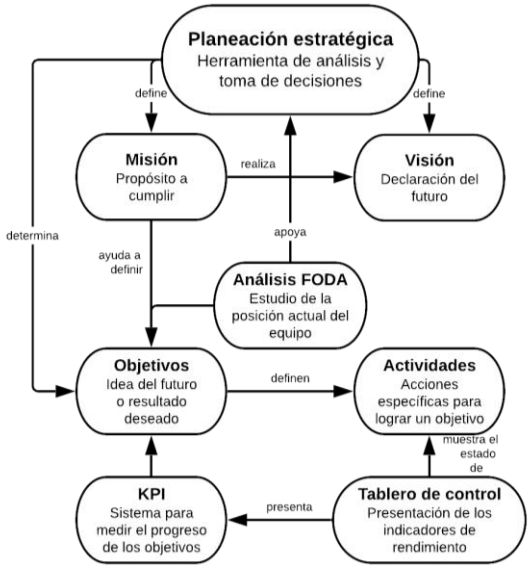

Figura 1. Mapa conceptual de la planeación estratégica.

Objetivos

Un objetivo es una idea del futuro o resultado deseado que una persona o un grupo de personas visualiza, planifica y se compromete a lograr (Locke y Latham, 1990). También puede entenderse como objetivo estratégico importante a la idea futura que, al enfocar todas las energías en su realización, genere el mayor impacto en el desarrollo de la organización (McChesney et al., 2012).

Actividades

Las actividades son declaraciones específicas que explican cómo se logrará un objetivo. Son las áreas que mueven la estrategia a las operaciones y son ejecutadas por equipos o individuos (Steiner, 1979). Estas actividades deben ser las de mayor apalancamiento en las que un equipo puede participar y rastrear constantemente los resultados para garantizar la ejecución y el logro de un objetivo estratégico (McChesney et al., 2012).

Key Performance Indicator (KPI)

Un indicador clave de rendimiento (key performance indicator, KPI) es un sistema que se utiliza para lograr los principales objetivos de cualquier negocio como atraer y retener clientes o aumentar los ingresos y reducir los costos (Klochkov, 2010). Los KPIs reflejan y derivan de los objetivos organizacionales. Estos indicadores miden el progreso y el logro de los objetivos (Shahin y Mahbod, 2007).

Tablero de control

Un tablero de control se utiliza para informar los KPIs y realizar un seguimiento de su rendimiento con respecto a los objetivos planeados (Allio, 2012; Sarikaya, Correll, Bartram, Tory y Fisher, 2018). Este tablero funciona como el marcador en un juego. Esto ayuda a los equipos a saber en todo momento si están ganando o no. Un tablero de control convincente le dice al equipo dónde están y dónde deberían estar, lo cual es información esencial para la resolución de problemas y la toma de decisiones del equipo. Es por eso que un equipo no puede funcionar sin un tablero de control que motive a la acción (McChesney et al., 2012).

Conceptos de ingeniería de software

En la Figura 2 se muestran los conceptos de ingeniería de software que se utilizaron en esta investigación. Entre ellos se encuentran los siguientes conceptos: arquitectura de software, patrón arquitectónico, modelo vista-controlador, spring framework, spring boot, spring security, thymeleaf, JPA, mapeo objeto-relacional e hibernate. Cada uno de los conceptos está relacionado entre sí y forman parte del desarrollo de la aplicación web.

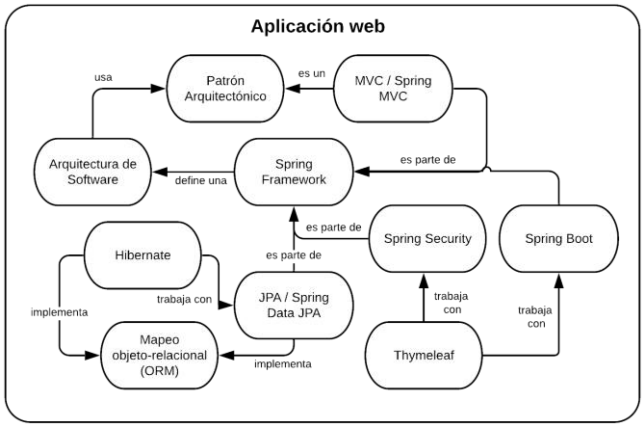

Figura 2. Mapa conceptual de ingeniería de software.

Aplicación web

Una aplicación web es una aplicación de software que depende de la web para su correcta ejecución (Gellersen y Gaedke, 1999). Esto quiere decir que la manera de acceder y utilizar una aplicación web es por medio de un navegador web y una conexión a internet. Además, las aplicaciones web aprovechan las facilidades de interacción de las páginas HTML para proporcionar y solicitar información a los usuarios que las utilizan (Ricca y Tonella, 2001).

La estructura típica de una aplicación web consiste en páginas HTML que se muestran al usuario, la ejecución de los comandos del lado del cliente que interactúan con el usuario, y los comandos del lado del servidor que realizan el procesamiento de los datos y que normalmente interactúan con bases de datos (Jazayeri, 2007).

Arquitectura de software

La arquitectura de software de un programa o sistema informático es la estructura del sistema que comprenden elementos de software, las propiedades externas visibles de esos elementos y las relaciones entre ellos. Se desarrolla una arquitectura

de software como el primer paso hacia el diseño de un sistema que tiene una colección de propiedades deseadas (Bass, Clements y Kazman, 2003). Dicho de otra manera, la arquitectura de sistemas de software es el conjunto de decisiones de diseño principales que se toman sobre el sistema (Taylor, Medvidović y Dashofy, 2010).

Patrón arquitectónico

Un patrón arquitectónico es una solución general reutilizable para un problema común en la arquitectura de software dentro de un contexto dado (Taylor et al., 2010). La idea general detrás de estos patrones consiste en describir una solución de diseño sólida en una forma explícita que se aplique en un contexto determinado y analizar los beneficios y los inconvenientes de la solución (Paakki, Karhinen, Gustafsson, Nenonen y Verkamo, 2000).

Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador (MVC) surge con el objetivo de reducir el esfuerzo de programación necesario en la implementación de sistemas a partir de estandarizar el diseño de las aplicaciones. El patrón MVC divide las partes que conforman una aplicación en el modelo, las vistas y los controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo (Díaz González y Fernández Romero, 2012).

El patrón MVC fue descrito por primera vez por Reenskaug (1979) de la siguiente manera:

1. Modelo: representa la funcionalidad principal y los datos.

2. Vista: representación visual del modelo. Muestra la información al usuario.

3. Controlador: enlace entre un usuario y el sistema. Maneja la entrada del usuario.

Spring

Spring es un framework de código abierto que facilita la creación de aplicaciones empresariales Java. Proporciona todo lo que se necesita para adoptar el lenguaje Java en un entorno empresarial con la flexibilidad de crear muchos tipos de arquitecturas según las necesidades de una aplicación (Johnson et al., 2021).

Spring se creó para abordar la complejidad del desarrollo de aplicaciones empresariales. La utilidad de spring no se limita al desarrollo del lado del servidor. Cualquier aplicación Java puede beneficiarse de spring en términos de simplicidad, capacidad de prueba y acoplamiento flexible (Walls, 2019).

Spring hace que la programación de Java sea más rápida, fácil y segura. El enfoque de spring en la velocidad, la simplicidad y la productividad lo ha convertido en el framework Java más popular del mundo (Binstock y Maple, 2019; JetBrains, 2020; Stack-Overflow, 2021).

Spring framework

El spring framework contiene una gran cantidad de clases y paquetes y está diseñado como un framework modular que puede introducirse gradualmente en un proyecto utilizando solo las funciones necesarias (Hemrajani, 2006).

Los desarrolladores pueden elegir qué módulos necesita la aplicación. Más allá de eso, el spring framework proporciona soporte básico para diferentes arquitecturas

de aplicaciones, incluyendo mensajería, datos transaccionales y persistencia y web. También incluye el framework web spring MVC basado en servlets de Java (Johnson et al., 2021).

Spring MVC

Spring MVC es el módulo del framework spring que se ocupa del modelo-vista-controlador o patrón MVC. Spring MVC implementa el patrón MVC utilizando un controlador frontal llamado DispatcherServlet. El DispatcherServlet se encarga principalmente de interceptar las solicitudes entrantes, convertir la carga útil de la solicitud en la estructura interna de los datos, enviar los datos al modelo para su posterior procesamiento y obtener los datos procesados con el fin de enviarlos a la vista para su representación. A su vez, DispatcherServlet utiliza la configuración de spring para descubrir los componentes delegados que necesita para la asignación de solicitudes, la resolución de vistas y el manejo de excepciones, entre otros (Johnson et al., 2021; Walls, 2019)

Spring boot

El spring framework ha existido durante más de una década y se ha convertido en el framework estándar de facto para desarrollar aplicaciones Java. En la actualidad, con más de 75 millones de descargas por mes, spring boot es el framework de Java más utilizado. Spring boot ofrece un nuevo paradigma para desarrollar aplicaciones spring ágilmente al pasar por alto el conocimiento de configuración inicial necesaria del framework, esto permite al desarrollador concentrarse en abordar las necesidades de funcionalidad de su aplicación (Heckler, 2021; Walls, 2016; Webb et al., 2021).

Spring security

Spring security es un framework de autenticación y control de acceso potente y altamente personalizable. Es la opción principal para implementar seguridad en aplicaciones spring. El framework se centra en proporcionar autenticación (verificar la identidad de un usuario), autorización (especificar los privilegios de acceso a los recursos) y protección contra ataques comunes a las aplicaciones Java (Alex et al., 2021; Spilca, 2020).

Thymeleaf

Thymeleaf es un moderno motor de plantillas Java del lado del servidor para entornos web. Thymeleaf es ideal para el desarrollo web HTML5 JVM moderno pues cuenta con módulos para spring framework, con una gran cantidad de integraciones con diversas herramientas y con la capacidad de conectar funcionalidades personalizadas (Deblauwe, 2021; Good, 2018; Thymeleaf, 2021).

JPA

La API de persistencia de Java (Java Persistence API, JPA) se ocupa de la forma en que los datos relacionales se asignan a objetos Java (entidades persistentes), la forma en que estos objetos se almacenan en una base de datos relacional para que se pueda acceder a ellos en un momento posterior y la existencia continua de un estado de la entidad, incluso después de que finalice la aplicación que la utiliza. Además de simplificar el modelo de persistencia de entidad, JPA estandariza el mapeo objeto-relacional (Biswas y Ort, 2006).

Mapeo objeto-relacional

El mapeo objeto-relacional (object-relational mapping, ORM) proporciona una metodología y un mecanismo para que los sistemas orientados a objetos mantengan sus datos a largo plazo de forma segura en una base de datos (O'Neil, 2008). Es decir, ORM es la persistencia automatizada de los objetos en una aplicación Java a las tablas en una base de datos SQL, utilizando metadatos que describen el mapeo entre las clases de la aplicación y el esquema de la base de datos SQL. En esencia, ORM funciona transformando datos de una representación a otra (Bauer, King y Gregory, 2015; Gracia del Busto y Yanes Enríquez, 2012).

Spring data JPA

El spring data JPA es un módulo del proyecto de spring data que facilita la implementación de repositorios basados en JPA. Este módulo se ocupa del soporte mejorado para capas de acceso a datos basadas en JPA. El spring data JPA facilita la creación de aplicaciones hechas con spring que utilizan tecnologías de acceso a datos. El objetivo de la abstracción del repositorio de spring data es reducir significativamente la cantidad de código estándar requerido para implementar capas de acceso a datos para varias fuentes de almacenamiento de persistencia (Gierke, Darimont, Strobl, Paluch y Bryant, 2021).

Hibernate

Hibernate es un framework de persistencia ORM para Java. Hibernate es una implementación de la especificación JPA que se puede usar fácilmente en cualquier entorno compatible con este (Hibernate, 2021). Como framework ORM, hibernate se

ocupa de la persistencia de los datos al aplicarse a las bases de datos relacionales, automatizando muchas tareas de codificación repetitivas (Bauer et al., 2015). Hibernate es quizás el ORM más utilizado actualmente en el mundo de los desarrolladores de Java (Hemrajani, 2006).

Trabajo relacionado

Metodologías de planeación estratégica

Dentro de la planeación estratégica existen varias metodologías populares que son utilizadas por equipos dentro de instituciones de diferentes indoles (Aldehayyat y Anchor, 2008; Kalkan y Bozkurt, 2013; Stonehouse y Pemberton, 2002; Webster, Reif y Bracker, 1989). A continuación se mencionan las más populares.

Balanced scorecard

El balanced scorecard ha despertado gran interés entre directivos y empresarios, hasta el punto donde se considera como uno de los más importantes modelos de planificación y gestión estratégica. El balanced scorecard es un modelo de gestión que traduce la estrategia en objetivos relacionados, medidos a través de indicadores y ligados a unos planes de acción que permiten alinear el comportamiento de los miembros de la organización (Fernández, 2001).

El balanced scorecard utiliza una herramienta visual llamada mapa estratégico para comunicar claramente un plan estratégico. El mapa estratégico ayuda a facilitar las discusiones entre ejecutivos sobre los vínculos en las cuatro perspectivas (la financiera, la del cliente, la interna y la de aprendizaje y crecimiento) del balanced scorecard lo que lo hace importante al ofrecer una manera de comunicar la información de alto

nivel en toda la organización en un formato fácilmente digerible (Kaplan y Norton, 1992; Kaplan y Norton, 1996; Kaplan y Norton, 2004).

Análisis FODA

El análisis ambiental es una parte fundamental del proceso de planificación de la gestión estratégica. El análisis FODA (fortalezas, oportunidades, debilidades y amenazas), elogiado por su sencillez y practicidad, implica la recopilación y representación de información sobre factores internos y externos que tienen o pueden tener un impacto en el negocio (Pickton y Wright, 1998). Stacey (1993) describe el análisis FODA como:

> una lista de las fortalezas y debilidades de una organización según lo indica un análisis de sus recursos y capacidades, además de una lista de las amenazas y oportunidades que identifica un análisis de su entorno. Obviamente, la lógica estratégica requiere que el patrón futuro de acciones a tomar debe combinar las fortalezas con las oportunidades, protegerse de las amenazas y buscar superar las debilidades. (pág.52)

Análisis PEST

El análisis PEST (político, económico, social y tecnológico) es una herramienta poderosa y ampliamente utilizada para comprender el riesgo estratégico. Identifica los cambios y los efectos del entorno macroeconómico externo en la posición competitiva de una empresa (Sammut-Bonnici y Galea, 2015).

El análisis PEST examina el impacto de cada uno de los factores en el negocio. Los resultados se pueden utilizar para aprovechar oportunidades y para hacer planes de contingencia para amenazas al preparar planes comerciales y estratégicos (Byars, 1991; Cooper, 2000).

Kotler (1997) afirma que el análisis PEST es una herramienta estratégica útil para comprender el crecimiento o declive del mercado, la posición comercial, el potencial y la dirección de las operaciones. El uso del análisis PEST puede considerarse eficaz para la planificación empresarial y estratégica, la planificación del mercadeo, el desarrollo de productos y negocios y los informes de investigación. PEST también asegura que el desempeño de la empresa esté alineado positivamente con las poderosas fuerzas de cambio que están afectando el entorno empresarial (Porter, 1985).

Hoshin Kanri

El sistema de planificación Hoshin Kanri, que tiene su origen en Japón, se puede interpretar de su traducción literal como una metodología para establecer una dirección estratégica. En el contexto del Hoshin Kanri, esto se resume en transmitir una prioridad estratégica, los medios para lograrla y los indicadores para medir el éxito. La aplicación del Hoshin Kanri traduce la intención estratégica en acciones y comportamientos requeridos del día a día. Los principales objetivos del Hoshin Kanri se pueden resumir en: (a) identificar áreas importantes de oportunidad para que la organización cambie o mejore, (b) determinar las acciones más rentables en toda la organización para lograr estos cambios, (c) crear un plan de implementación detallado y (d) proporcionar un mecanismo de revisión para identificar acciones correctivas y aprendizaje (Bechtell, 1996; Tennant y Roberts, 2000; Tennant y Roberts, 2001).

Objectives and Key-Results (OKRs)

La metodología denominada objetivos y resultados clave (objectives and key-results, OKRs) es un protocolo colaborativo de establecimiento de objetivos para

empresas, equipos e individuos. Los objetivos manifiestan aquello que se debe lograr. Específicamente, un objetivo es una declaración concisa que describe una meta cualitativa amplia diseñada para impulsar a la organización en la dirección deseada. Por otro lado, los resultados clave evalúan y monitorean cómo se logrará el objetivo. Concretamente, un resultado clave es una declaración cuantitativa que mide el logro de un objetivo determinado. La metodología de gestión basada en OKRs ayuda a asegurar que la empresa, o bien, que los empleados concentren sus esfuerzos en hacer contribuciones medibles en los temas importantes de la organización (Doerr, 2018; Niven y Lamorte, 2016).

The 4 disciplines of execution

El libro *The 4 disciplines of execution* (McChesney et al., 2012), habla acerca de cómo ejecutar una estrategia efectiva en medio de la operación del día a día. Las cuatro disciplinas son reglas precisas para traducir la estrategia en acción en todos los niveles de una organización. Estas disciplinas se enfocan en establecer lo realmente importante dejando a un lado lo urgente del momento. A continuación, se describen las disciplinas contenidas en la obra:

1. Enfocarse en lo crucialmente importante. Esta es la disciplina del enfoque. La ejecución excepcional comienza con la reducción del enfoque, identificando claramente lo que se debe hacer. De lo contrario, se lograrán objetivos que no serán realmente importantes.

2. Actuar sobre las medidas de predicción. Esta es la disciplina del apalancamiento. El 80% de los resultados provendrán del 20% de las actividades. Es importante identificar y actuar sobre las actividades con mayor apalancamiento.

3. Llevar un tablero de control convincente. Esta es la disciplina del compromiso. Las personas y los equipos juegan de manera diferente cuando llevan la puntuación, y el tipo correcto de marcadores motiva a los jugadores a ganar.

4. Rendición de cuentas. Esta es la disciplina de la responsabilidad. Cada equipo participa en un proceso semanal simple que resalta los éxitos, analiza los fracasos y corrige el curso, según sea necesario.

Becoming a mission-driven church

El libro *Becoming a mission-driven church* (Brantley et al., 2015), publicado y escrito por administradores de la División Norteamericana, denota lo importante que es tener una estrategia basada en la misión para así cumplir objetivos más audaces. Para ejecutar una estrategia basada en la misión, los autores identifican una serie de hábitos a seguir. Estos hábitos se describen a continuación:

1. Crear equipo. Establecer un ambiente de calidez, unidad y confianza en el equipo. Dejar de lado el ego (Filipenses 2) y trabajar hacia un objetivo común (Hechos 2).

2. Acordar la misión. Determinar la misión como equipo. Buscar la dirección en su declaración de misión, visión y objetivos anuales (Nehemías 4:6; 1 Reyes 8; Mateo 28:19, 20; Hechos 17:6).

3. Estrategia en la escritura. Trabajar en estrecha colaboración para llevar a cabo la estrategia (1 Corintios 12). Cada miembro del equipo debe tener fácil acceso a la estrategia que describe los planes futuros. La estrategia guía la planificación, el presupuesto y el funcionamiento en general (Lucas 14:28).

4. Implementación de la estrategia. Actuar sobre las cosas que obtienen un resultado directo en el cumplimiento de la misión. Monitorear y graficar el progreso (Mateo 7:21–23; Santiago 1:22–25).

5. Aprendizaje continuo. Evaluar constantemente las acciones realizadas para el crecimiento continuo del equipo y el mejoramiento de la forma en que se cumple la misión. Informar los resultados y aprender a ser más eficaces al ejecutar las tareas necesarias (Mateo 5:14–30; 1 Tesalonicenses 5:21).

Los hábitos declarados en la obra *Becoming a mission-driven church* (Brantley et al., 2015) están inspirados en las disciplinas establecidas en el libro *The 4 disciplines of execution* (McChesney et al., 2012). Especialmente en la declaración de la misión, la visión, los objetivos y las actividades, la medición del progreso a través de un tablero de control y las reuniones de rendición de cuenta.

Herramientas tecnológicas de gestión empresarial

Actualmente existen varias soluciones de software de gestión de proyectos tales como monday.com, un software de gestión de proyectos y equipos en la nube (Monday, 2021). Por otra parte, Trello es una aplicación web de gestión de proyectos que permite a los usuarios organizar sus tareas a través de tableros kanban, listas, tarjetas y otras herramientas de productividad (Trello, 2021). Además, Slack es una aplicación de chat en equipo (Slack, 2021) que fomenta la comunicación y el intercambio de conocimientos entre miembros de una organización, permitiendo la colaboración y coordinación del trabajo.

También existe Asana, una herramienta de gestión de proyectos que permite a los equipos organizar y seguir sus tareas y el progreso de los proyectos en tiempo real

(Asana, 2021). Finalmente, se menciona Wrike, una aplicación en línea que permite a los equipos administrar proyectos a través de listas y pantallas tipo hoja de cálculo, diagramas de Gantt y calendario (Wrike, 2021).

Por lo general, este tipo de software ofrece características como seguimiento del tiempo, diferentes vistas e informes, registro de actividades, tableros de control personales, gestión de trabajo, listas de tareas y proyectos. Los precios de estas herramientas oscilan entre los $5 a los $25 dólares americanos por mes y por usuario y también existen precios personalizados (ver Tabla 1).

Por otro lado, existen gestores del plan estratégico los cuales ofrecen diferentes aproximaciones para el manejo de un plan estratégico. Se pueden mencionar algunos de ellos como lo son Envisio que es una plataforma en la nube que une al usuario con los planes estratégicos del departamento (Envisio, 2021). WorkBoard proporciona soluciones de gestión de objetivos en tiempo real para amplificar los resultados comerciales (WorkBoard, 2021). Cascade incorpora la planificación, la ejecución, los paneles y la gestión de personas en un sistema integrado con el objetivo de mejorar el rendimiento empresarial (Cascade, 2021). Estos gestores ofrecen una mejor representación del trabajo que debe realizarse al tener un plan estratégico.

Estas herramientas ofrecen características tales como administración de presupuesto, establecer y gestionar objetivos, definición de las prioridades, seguimiento del progreso, visualización de datos, administración de recursos, tablero de control de actividades, evaluación de desempeño, informes y estadísticas. Los precios de estas herramientas oscilan entre los $10 a los $60 dólares americanos por mes, por usuario y también otras opciones con precios personalizados (ver Tabla 2).

Tabla 1

Lista de herramientas tecnológicas de gestión de proyectos

Nombre	Características	Precios
monday.com	Registro de actividad, búsqueda y filtrado, tableros privados, seguimiento del tiempo, vista Kanban, entre otras.	De $29 a $5,851+ al mes.
Trello	Tableros personalizados, colaboración de equipo entre tableros, adjuntar archivos, entre otras.	Gratuito y de $5 a $20.83+ por usuario al mes.
Slack	Chats públicos y privados, conversaciones en hilos, mensajes directos, compartir archivos, notificaciones personalizadas, entre otras.	Gratuito y de $8 a $15 por usuario al mes.
Asana	Listas de tareas, tableros Kanban, vistas e informes, soporte y control, gestión de trabajo, tareas y proyectos, entre otras.	Gratuito, personalizado y de $11.99 a $23.99 por usuario al mes.
Wrike	Gestión de proyectos y tareas en colaboración. Ver proyectos como una hoja de cálculo, lista, diagrama de Gantt y tablero Kanban. Ver y compartir información en tableros personalizables, seguimiento del tiempo, entre otras.	Gratuito y de $117.60 a $415.20 al año por usuario.

Tabla 2

Lista de herramientas tecnológicas de gestión del plan estratégico

Nombre	Características	Precios
Envisio	Informes y estadísticas, informes de rendimiento, mapeo estratégico, chat, notificaciones automáticas y en tiempo real, planificación de acciones, entre otras.	Suscripción. Plan personalizado.
Workboard	Establecer objetivos y prioridades, progreso del trabajo, delegar y compartir elementos de acción, rastrear elementos de acción, obtener informes de estado semanales automáticos, establecer objetivos, definir prioridades, medir el progreso, entre otras.	De $9 a $50+ por usuario al mes
Cascade	Administración de presupuesto, gestión de objetivos, visualización de datos, seguimiento del progreso, administración de recursos, tablero de actividades, importación/exportación de datos, establecimiento de objetivos/seguimiento, entre otras.	Plan personalizado y de $9 a $58+ por usuario al mes

A pesar de que estas herramientas son utilizadas por diferentes organizaciones ofreciéndoles la capacidad de manejar sus proyectos y actividades diarias, ninguna de ellas es adecuada para las necesidades de la Universidad de Montemorelos. Específicamente, estas herramientas ofrecen únicamente características básicas de gestión, altos precios por servicios más completos y recursos insuficientes para el contexto institucional tales como la intervención en el análisis de la misión, la visión, el análisis FODA por parte del equipo, la declaración de objetivos y actividades y las reuniones de rendición de cuentas a nivel de objetivos y actividades.

CAPÍTULO III

METODOLOGÍA

Introducción

En este capítulo se presenta la descripción de los pasos seguidos en esta investigación para el desarrollo de la aplicación web para la gestión del plan estratégico de la Universidad de Montemorelos. En la Figura 3 se muestra la metodología de la investigación, en la cual se siguió un proceso iterativo incremental. Esto quiere decir que el proyecto estuvo compuesto por varios ciclos completos de etapas repetitivas donde se generó un producto final de software.

La primera etapa es la de conceptualización de la idea en donde se establece una vista descriptiva del proyecto previsto, de dónde surge, cuál es su propósito y los objetivos. En la etapa de análisis de requisitos se especifican los objetivos del proyecto en términos de casos de uso. En la etapa de diseño de software se utiliza la información recopilada en la fase de requisitos para diseñar el sistema, describiendo en detalle las características y operaciones deseadas.

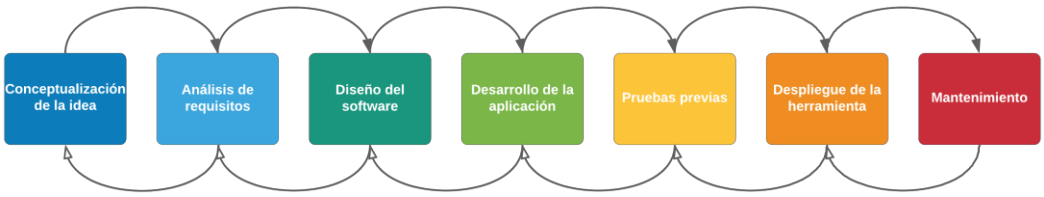

Figura 3. Diagrama de la metodología de la investigación.

En la etapa de desarrollo de la aplicación se tomaron los requisitos y las especificaciones del diseño del proyecto y se programó la aplicación web. En la etapa de pruebas previas se verificó que el sistema funcionaba según los requisitos establecidos, verificando errores según los métodos de prueba de caja negra. En la etapa de despliegue de la herramienta se desplegó la aplicación web creada en un servidor de la Universidad de Montemorelos para su uso por parte de los usuarios finales. En la etapa de mantenimiento se hicieron diversos ajustes a la aplicación web de acuerdo con los comentarios de los usuarios finales de la aplicación. En las siguientes subsecciones se describen cada una de estas etapas.

Conceptualización de la idea

En el año 2018 se le extendió una invitación al Dr. Harvey Alférez, director del Instituto de Ciencia de Datos de la Universidad de Montemorelos, por Paul Hopkins, director del Departamento de Social Media and Big Data Services de la División Norteamericana de la Iglesia Adventista del Séptimo Día para explorar posibilidades de colaboración en el contexto de ciencia de datos. Dos años después, se invitó nuevamente al Dr. Alférez a colaborar con el departamento de Social Media and Big Data Services, esta vez con la intención de trabajar en un proyecto de estrategia de datos para la División Norteamericana. De esta colaboración nacieron varias ideas sobre estrategia de datos para la división. Uno de estos proyectos fue la creación de una aplicación web para la gestión del plan estratégico de la División Norteamericana.

Como resultado de esta etapa, el Dr. Alférez presentó un prototipo de la aplicación web para la gestión del plan estratégico de la división ante los administradores de esa organización. Este prototipo fue creado con Balsamiq que es una herramienta de

diseño de interfaz de usuario para crear wireframes, también llamados maquetas o prototipos (ver Figura 4). Para la creación de este prototipo se combinaron las ideas presentadas en los libros Becoming a mission-driven church (Brantley et al., 2015) y The 4 disciplines of execution (McChesney et al., 2012) para el desarrollo de un plan estratégico organizacional efectivo.

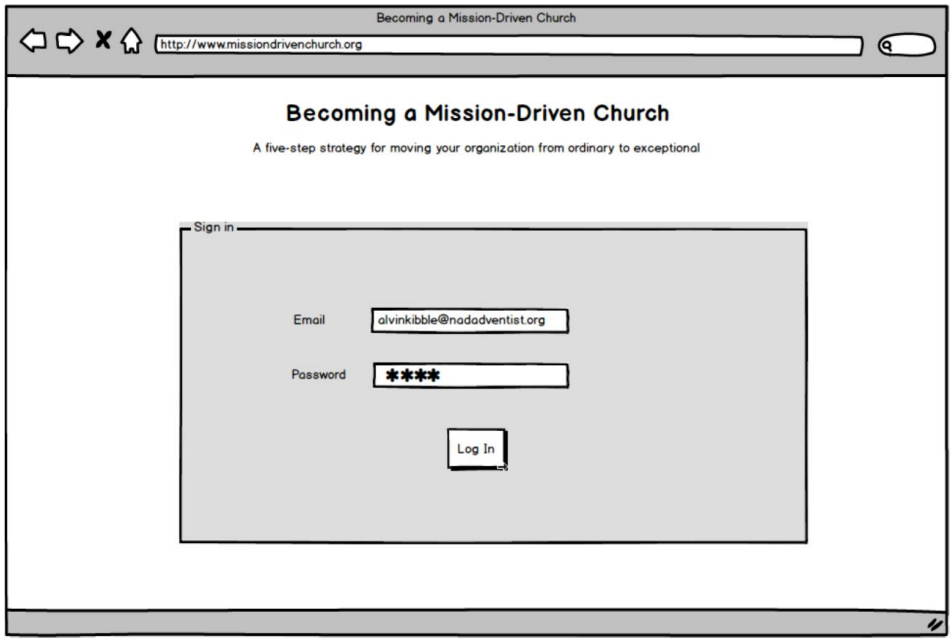

Figura 4. Pantalla inicial del prototipo de la aplicación web para la gestión del plan estratégico de la División Norteamericana.

Análisis de requisitos

En una reunión del Dr. Alférez con el Dr. Ismael Castillo, rector de la Universidad de Montemorelos, se llegó a la conclusión de continuar el proyecto de la creación de una aplicación web para la gestión del plan estratégico para ser aplicado en la UM. De esta forma, se continuó el trabajo avanzando a una nueva etapa donde se analizaron

los requisitos para el desarrollo de la aplicación web destinada a la gestión del plan estratégico de la Universidad de Montemorelos. Se espera que a futuro el proyecto también pueda ser desplegado en la División Norteamericana.

Por medio de una serie de diagramas de caso de uso del Lenguaje Unificado de Modelado (Unified Modeling Language, UML) durante esta etapa, se determinaron dos actores principales y las acciones que estos llegarían a realizar con la aplicación web. En el Apéndice A se describe la documentación de los casos de uso como requisitos funcionales del software, especificando las interacciones de los actores con el sistema.

A continuación, se describen dos diagramas de caso de uso para una mejor comprensión del funcionamiento del software desarrollado en este trabajo de investigación. En el diagrama de casos de uso de la Figura 5 se muestra al actor administrador. Este actor es la persona que se encarga de gestionar los departamentos y los usuarios de la institución dentro de la aplicación web.

A continuación se describen brevemente los casos de uso que realiza el actor administrador:

1. Crear/actualizar/borrar departamentos: el actor administrador es capaz de crear, modificar y/o borrar un departamento específico de la institución.

2. Crear/actualizar/borrar usuarios: el actor administrador es el responsable de crear, editar y/o borrar las cuentas de los usuarios dentro de la institución.

En el diagrama de casos de uso de la Figura 6 se muestra al actor usuario. Este actor representa todo líder que gestiona el plan estratégico de su departamento dentro de la aplicación web.

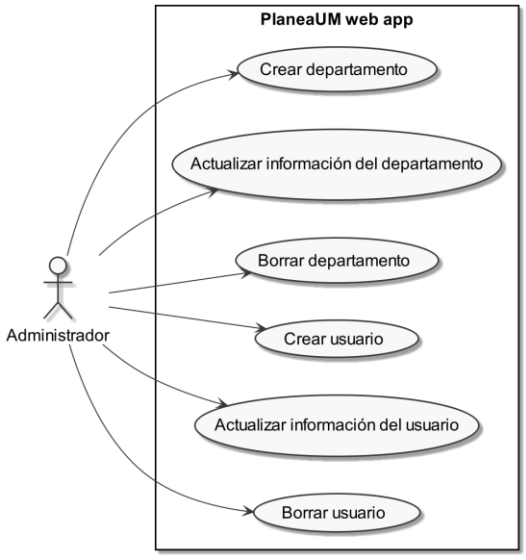

Figura 5. Diagrama de casos de uso del administrador.

A continuación se describen brevemente los casos de uso que lleva a cabo el actor usuario:

1. Actualizar la misión/visión del departamento: el actor usuario expone la misión y visión del departamento al que pertenece.

2. Agregar/actualizar/borrar las fortalezas, debilidades, amenazas u oportunidades del departamento: el actor usuario añade en esta sección los resultados del análisis FODA que ha hecho en su departamento.

3. Crear/actualizar/borrar objetivos: el actor usuario declara y administra los objetivos del departamento.

4. Actualizar KPI del objetivo: el actor usuario actualiza el KPI del objetivo con la finalidad de medir el avance.

5. Crear/actualizar/borrar actividades: el actor usuario declara y administra las actividades específicas para el cumplimiento de los objetivos del departamento.

6. Actualizar el progreso de la actividad: el actor usuario actualiza el progreso de la actividad con la finalidad de medir el cumplimiento.

7. Crear/actualizar/borrar elementos del presupuesto de la actividad: el actor usuario declara y administra los detalles del presupuesto de las actividades.

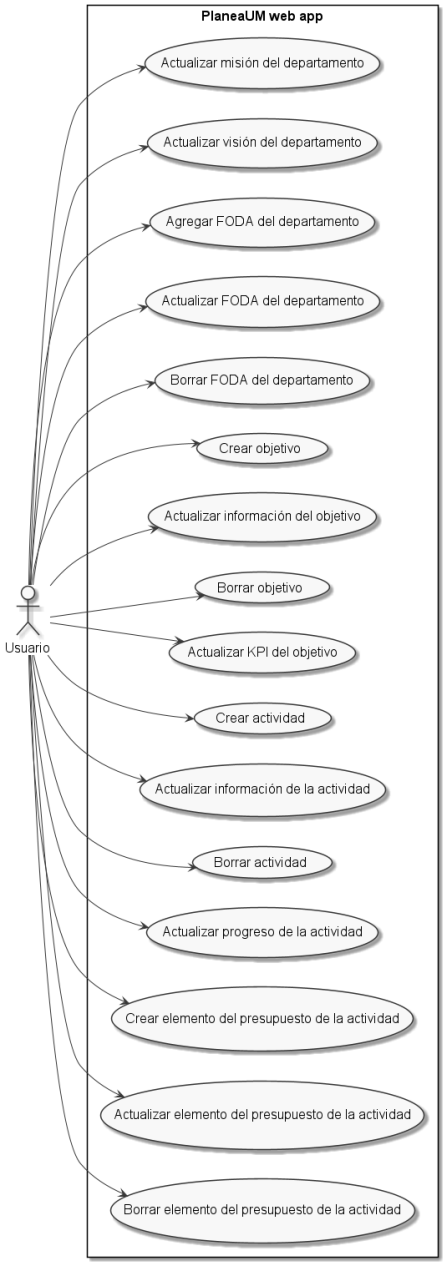

Figura 6. Diagrama de casos de uso del usuario.

Diseño del software

En esta sección se creó un conjunto de diagramas de los elementos que conforman el diseño del software desarrollado para una mejor comprensión de su funcionamiento. El primer paso consistió en especificar los datos que eran necesarios en la interacción de los actores con el sistema. Como resultado, se diseñó una base de datos con cada una de las tablas y atributos necesarios para el funcionamiento de la herramienta. La Figura 7 presenta el diagrama entidad-relación del proyecto.

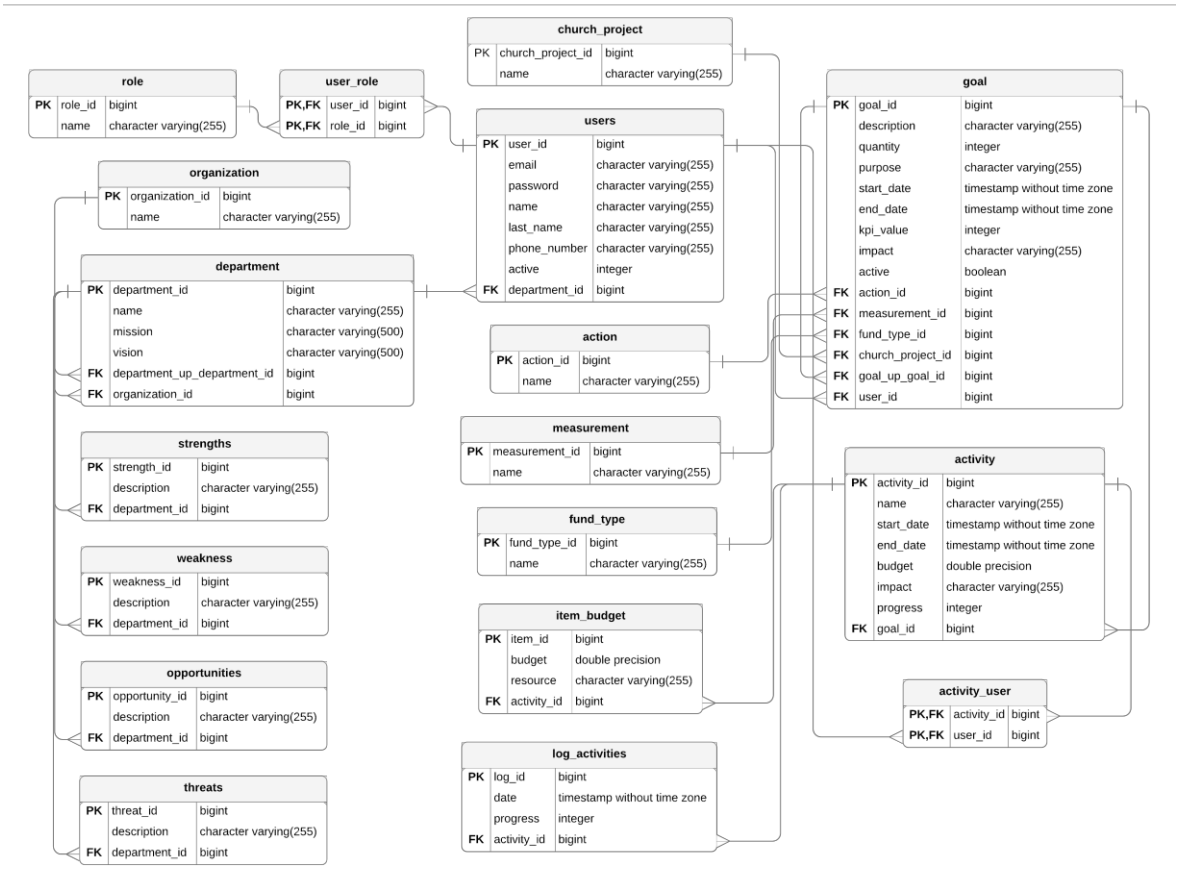

Figura 7. Diagrama entidad-relación.

Una vez establecida la información a recolectar y las relaciones entre las tablas, se empezó a trabajar en las clases correspondientes para poder utilizar la información dentro de la aplicación web. Como resultado se obtuvo el diagrama de clases de la Figura 8 que describe la estructura del software. Este diagrama muestra, principalmente, cómo las clases se relacionan entre sí dentro del sistema.

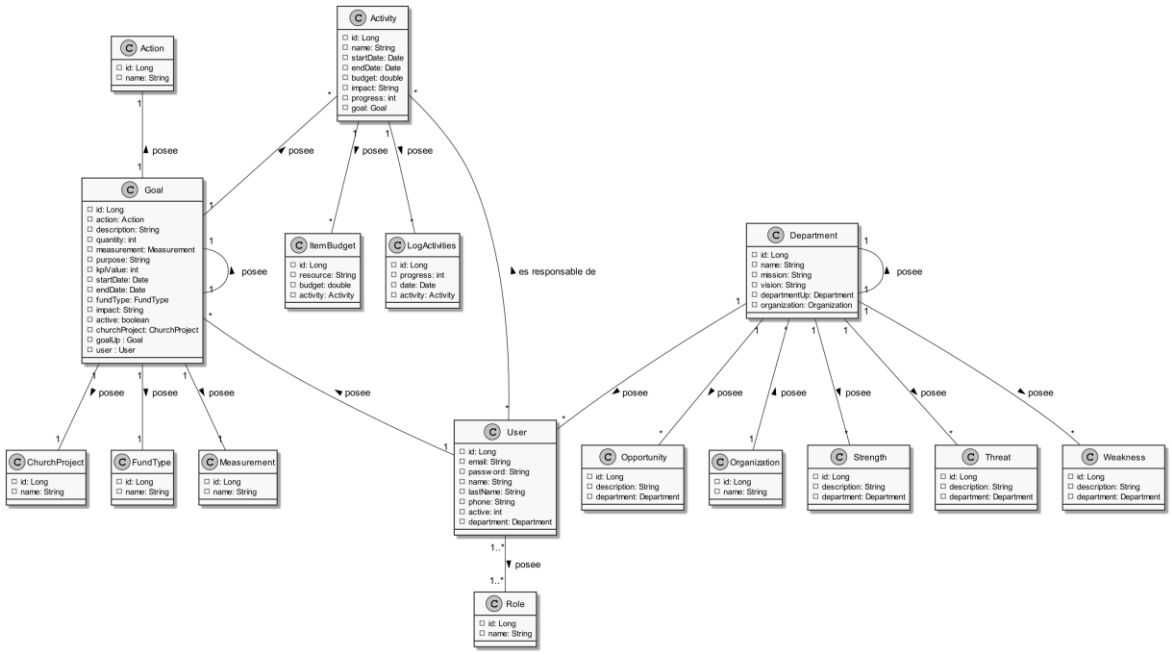

Figura 8. Diagrama de clases.

En este paso se utilizó JPA para crear las entidades asociadas a las tablas en la base de datos. Esto quiere decir que, por medio de la persistencia establecida con hibernate se automatizó el mapeo de las clases, sus atributos y las relaciones entre éstas al esquema de la base de datos, creando así el modelo. Esto generó una relación uno a uno entre las clases de la aplicación y las tablas dentro de la base de datos. Entre estas entidades se encuentran representados los departamentos, los usuarios,

los objetivos, las actividades y demás como clases dentro del software. En el Apéndice B se encuentra el diccionario de datos que describe detalladamente las relaciones entre las entidades/clases y sus atributos.

Diagramas de secuencia

Una vez establecido el diseño de los datos se empezó a trabajar en el flujo de trabajo que se iba a seguir dentro del sistema para el cumplimiento de cada uno de los casos de uso. Para esto, se crearon diagramas de secuencia UML que describen la interacción entre los objetos dentro del sistema para así cumplir con los casos de uso.

A continuación se describen los diagramas de secuencia que corresponden a cada uno de los casos de uso presentados en la Figura 5 y en la Figura 6 para una mejor comprensión del funcionamiento del software desarrollado en este trabajo de investigación.

Crear departamento

En la Figura 9 se muestra el diagrama de secuencia para la creación de departamentos dentro de la aplicación web. El actor administrador le da clic al botón de agregar nuevo departamento en la vista de departamentos (departments.html). Enseguida el sistema muestra una ventana donde el actor administrador introduce la información necesaria.

Una vez ingresada la información, al hacer clic en guardar, se envían los datos del departamento (DepartmentController.java), se valida que los datos de todos los campos se hayan ingresado y se inicia el proceso para guardar la información. DepartmentController.java ejecuta el método save de DepartmentService.java que así mismo

utiliza el DepartmentRepository.java para conectarse con la tabla Department de la base de datos y guardar el departamento. Después de ejecutar la operación de persistencia, DepartmentRepository.java regresa a DepartmentController.java que a continuación redirecciona hacia la vista departments.html que muestra al actor administrador la información de los departamentos creados.

Actualizar información del departamento

En la Figura 10 se muestra el diagrama de secuencia para actualizar departamentos dentro de la aplicación web. El actor administrador le da clic al botón de actualizar departamento en la vista de departamentos (departments.html). Enseguida el sistema muestra una ventana donde el actor administrador modifica la información necesaria. Los campos que el actor administrador puede modificar son el nombre del departamento y el departamento arriba.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos del departamento a DepartmentController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. DepartmentController.java ejecuta el método save de DepartmentService.java que así mismo utiliza el DepartmentRepository.java para conectarse con la tabla Department de la base de datos y actualizar la información del departamento.

Después de ejecutar la operación de persistencia, DepartmentRepository.java regresa a DepartmentController.java que a continuación redirecciona hacia la vista departments.html que le muestra al actor administrador la información actualizada de los departamentos.

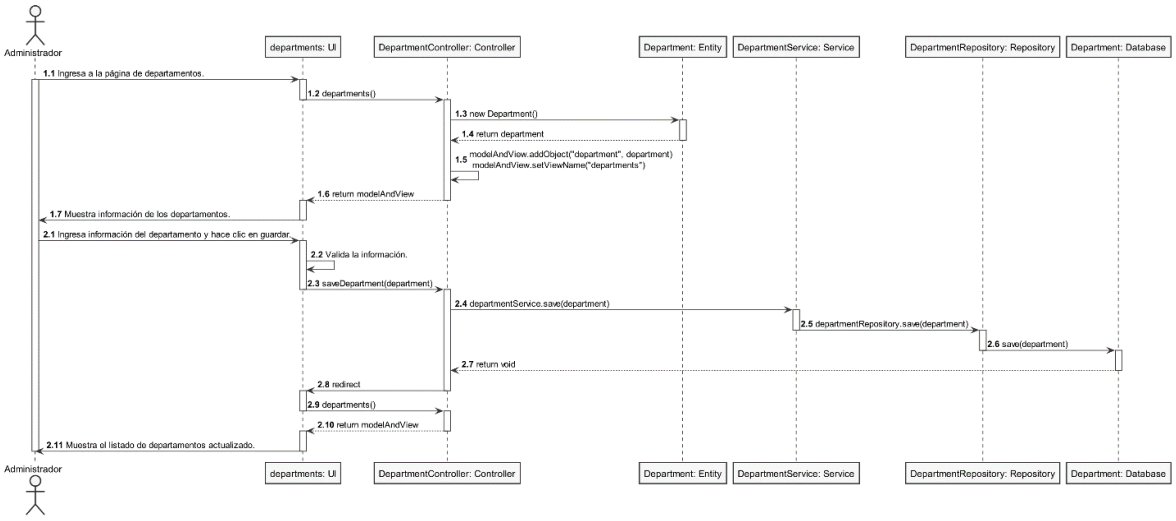

Figura 9. Diagrama de secuencia para agregar departamentos.

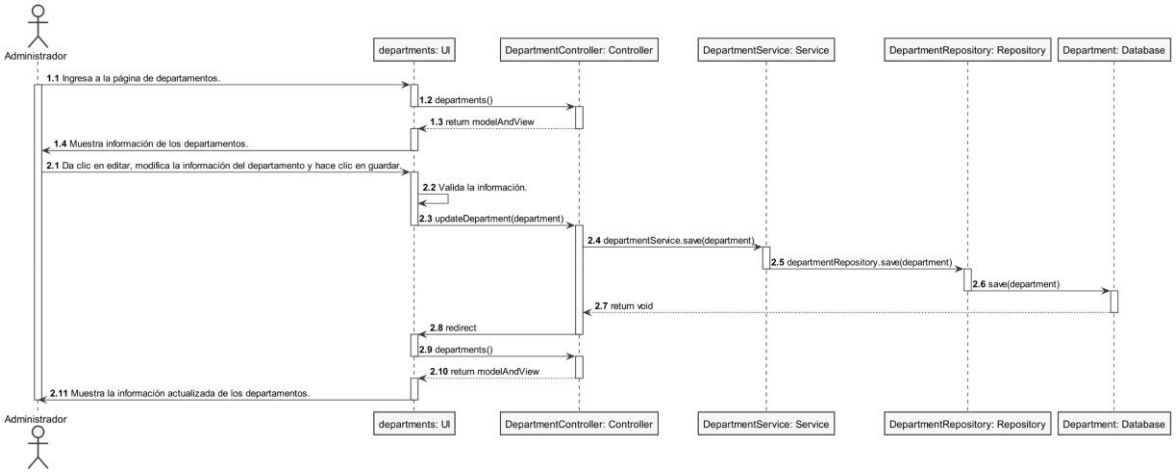

Figura 10. Diagrama de secuencia para actualizar departamentos.

Borrar departamento

En la Figura 11 se muestra el diagrama de secuencia para borrar departamentos dentro de la aplicación web. El actor administrador le da clic al botón de borrar departamento en la vista de departamentos (departments.html). Enseguida el sistema muestra una ventana donde se le pide al actor administrador confirmar la acción.

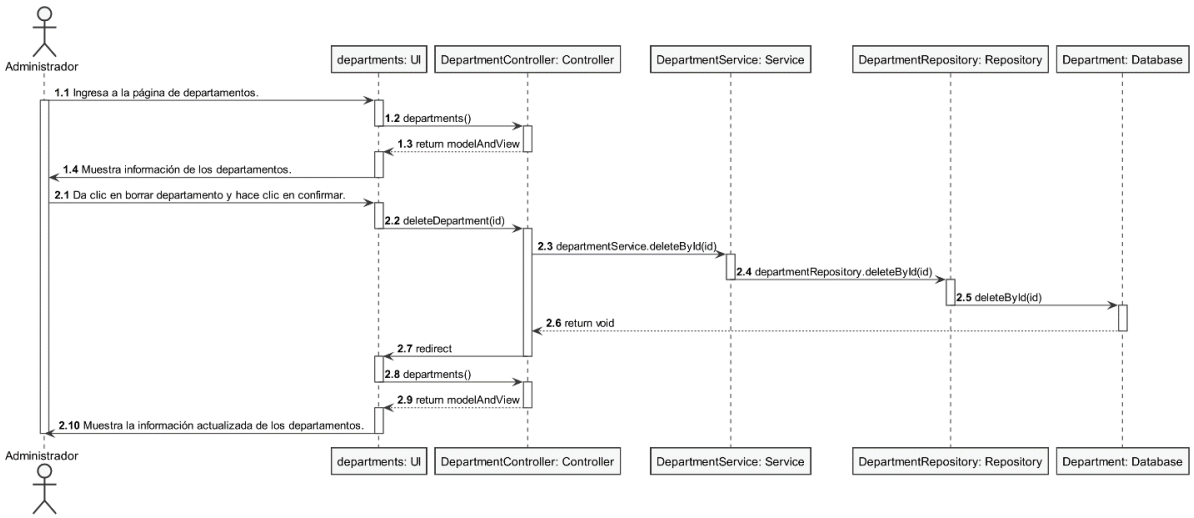

Figura 11. Diagrama de secuencia para borrar departamentos.

Al hacer clic en borrar, se envía como parámetro el identificador del departamento a DepartmentController.java, el cual inicia el proceso para borrar el departamento. DepartmentController.java ejecuta el método deleteById de DepartmentService.java que así mismo utiliza el DepartmentRepository.java para conectarse con la tabla Department de la base de datos y borrar el departamento. Después de ejecutar la operación de persistencia, DepartmentRepository.java regresa a DepartmentController.java que a continuación redirecciona hacia la vista departments.html que muestra al actor administrador la información actualizada de los departamentos.

Crear usuario

En la Figura 12 se muestra el diagrama de secuencia para la creación de usuarios dentro de la aplicación web. El actor administrador le da clic al botón de agregar nuevo usuario en la vista de usuarios (users.html). Enseguida el sistema muestra una ventana donde el actor administrador introduce la información necesaria. Los campos

que el actor administrador debe introducir son el nombre, el apellido, el correo electrónico, la contraseña, el departamento al que pertenecerá el usuario y el rol que tendrá. Una vez ingresada la información, al hacer clic en guardar, se envían los datos del usuario a UserController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información.

UserController.java ejecuta el método save de UserService.java que así mismo utiliza el UserRepository.java para conectarse con la tabla User de la base de datos y guardar el usuario. Después de ejecutar la operación de persistencia, UserRepository.java regresa a UserController.java que a continuación redirecciona hacia la vista users.html que muestra al actor administrador la información de los usuarios creados.

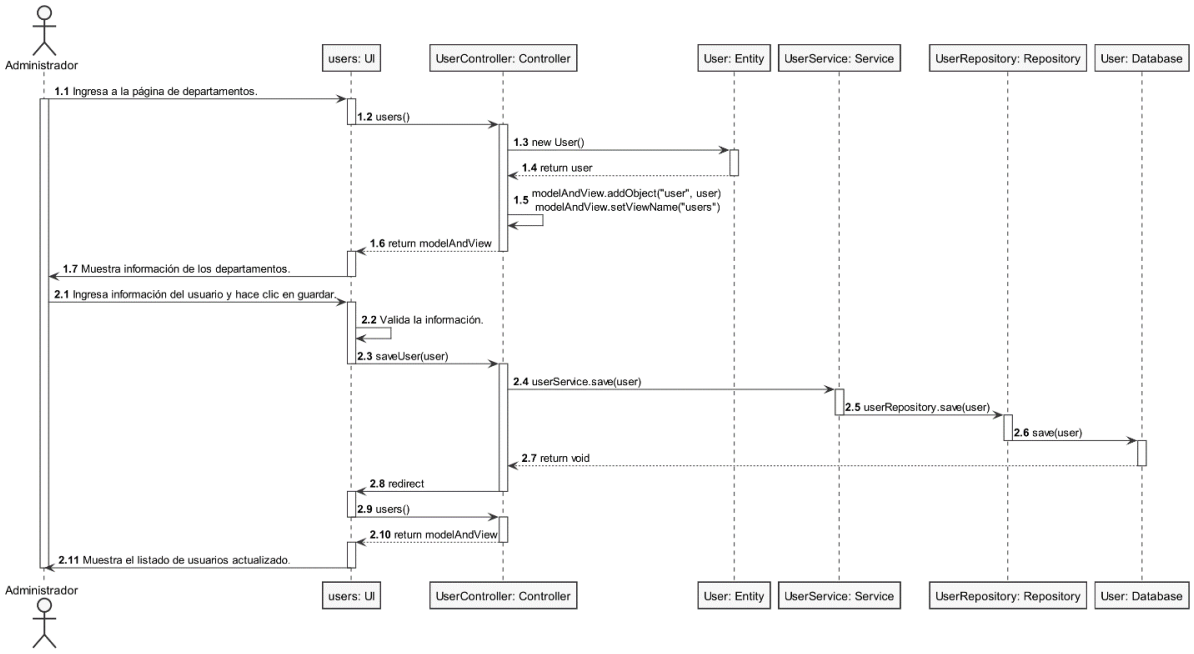

Figura 12. Diagrama de secuencia para agregar usuarios.

Actualizar información del usuario

En la Figura 13 se muestra el diagrama de secuencia para la actualización de usuarios dentro de la aplicación web. El actor administrador le da clic al botón de actualizar usuario en la vista de usuarios (users.html). Enseguida el sistema muestra una ventana donde el actor administrador modifica la información necesaria. Los campos que el actor administrador puede modificar son el nombre, el apellido, el correo electrónico, la contraseña, el departamento al que pertenece el usuario y el rol que tendrá.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos del usuario a UserController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. UserController.java ejecuta el método save de UserService.java que así mismo utiliza el UserRepository.java para conectarse con la tabla User de la base de datos y actualizar la información del usuario. Después de ejecutar la operación de persistencia, UserRepository.java regresa a UserController.java que a continuación redirecciona hacia la vista users.html que muestra al actor administrador la información actualizada de los usuarios.

Borrar usuario

En la Figura 14 se muestra el diagrama de secuencia para borrar usuarios dentro de la aplicación web. El actor administrador le da clic al botón de borrar usuario en la vista de usuarios (users.html). Enseguida el sistema muestra una ventana donde se le pide al actor administrador confirmar la acción. Al hacer clic en borrar, se envía como parámetro el identificador del usuario a UserController.java, el cual inicia el proceso para borrar el usuario.

UserController.java ejecuta el método deleteById de UserService.java que así mismo utiliza el UserRepository.java para conectarse con la tabla User de la base de datos y borrar el usuario. Después de ejecutar la operación de persistencia, UserRepository.java regresa a UserController.java que a continuación redirecciona hacia la vista users.html que muestra al actor administrador la información actualizada de los usuarios.

Actualizar misión del departamento

En la Figura 15 se muestra el diagrama de secuencia para la actualización de la misión del departamento dentro de la aplicación web. El actor usuario le da clic al botón de editar misión en la vista de misión (mission.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. El campo que el actor usuario debe introducir es la descripción de la misión.

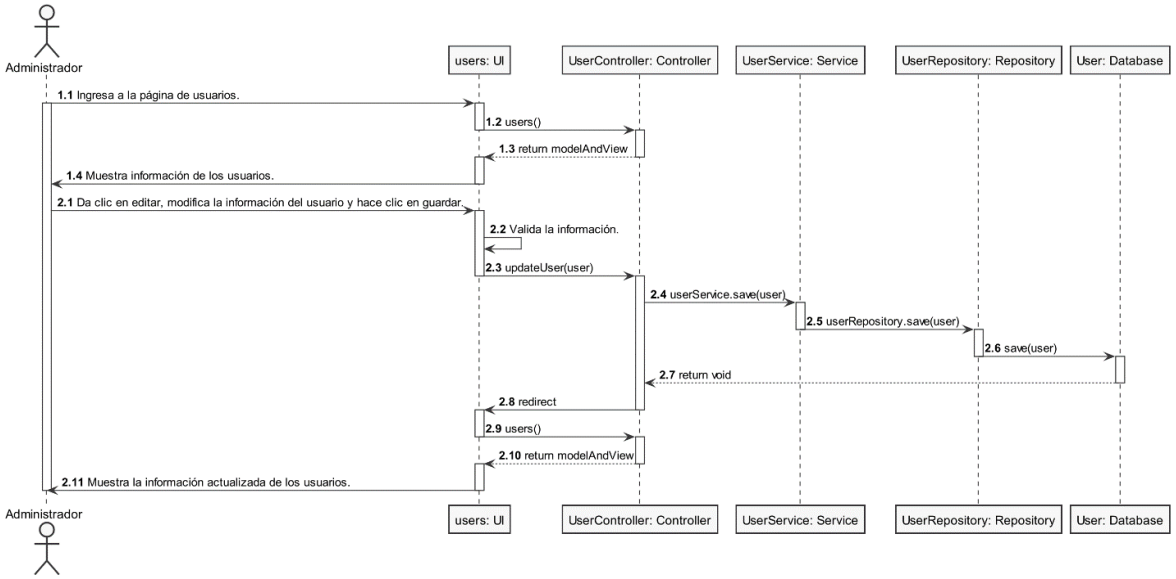

Figura 13. Diagrama de secuencia para actualizar usuarios.

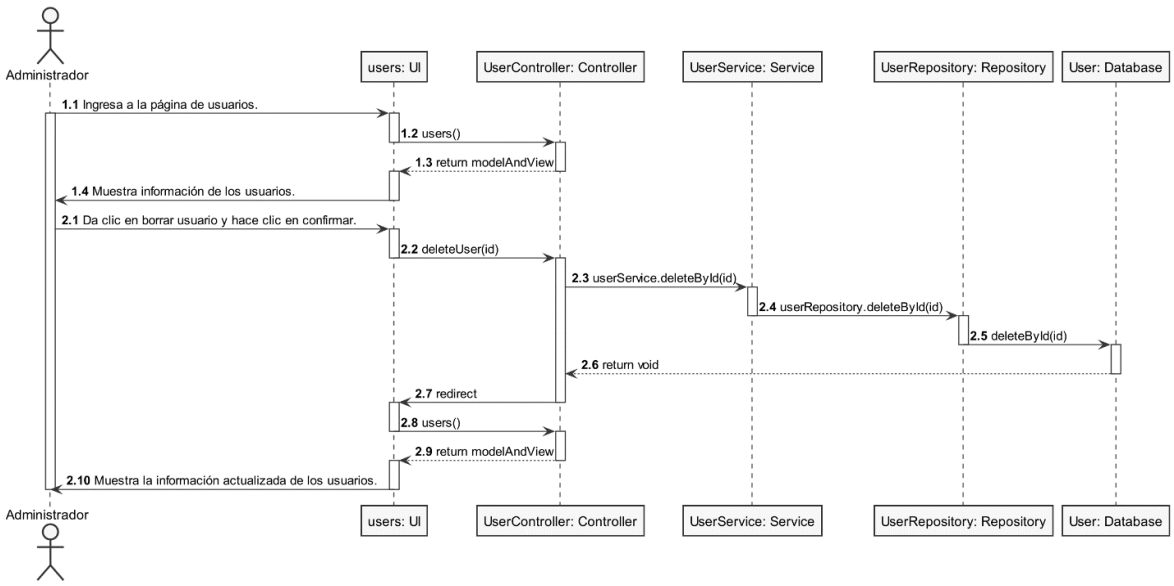

Figura 14. Diagrama de secuencia para borrar usuarios.

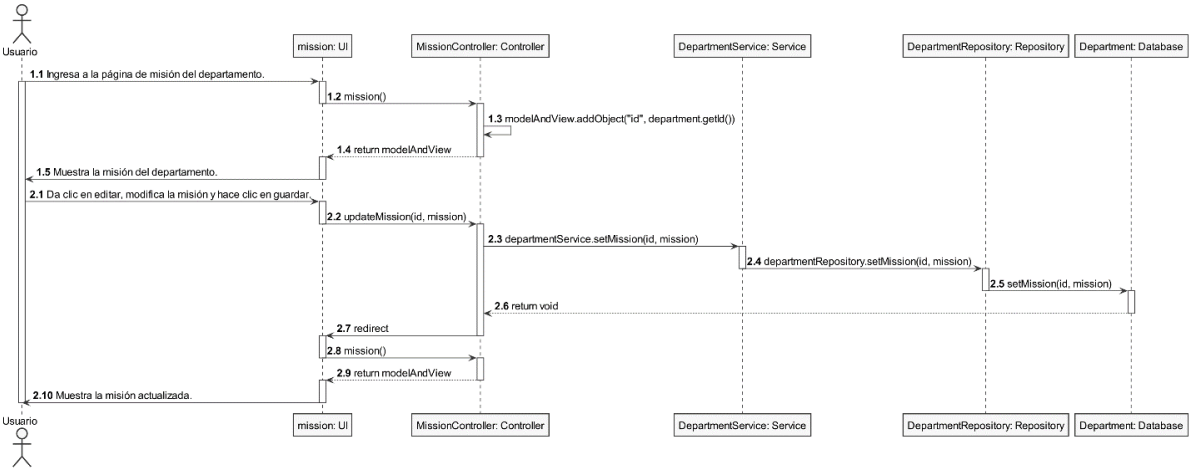

Figura 15. Diagrama de secuencia para actualizar la misión del departamento.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían como parámetro el identificador del departamento al que pertenece el actor usuario y el texto de la misión a MissionController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información.

MissionController.java ejecuta el método setMission de DepartmentService.java que así mismo utiliza el DepartmentRepository.java para conectarse con la tabla Department de la base de datos y actualizar la misión del departamento. Después de ejecutar la operación de persistencia, DepartmentRepository.java regresa a MissionController.java que a continuación redirecciona hacia la vista mission.html que muestra al actor usuario la información actualizada de la misión.

Actualizar visión del departamento

En la Figura 16 se muestra el diagrama de secuencia para la actualización de la visión del departamento dentro de la aplicación web. El actor usuario le da clic al botón de editar visión en la vista de visión (vision.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. El campo que el actor usuario debe introducir es la descripción de la visión.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían como parámetro el identificador del departamento al que pertenece el actor usuario y el texto de la visión a VisionController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información.

VisionController.java ejecuta el método setVision de DepartmentService.java que así mismo utiliza el DepartmentRepository.java para conectarse con la tabla Department de la base de datos y actualizar la visión del departamento. Después de ejecutar la operación de persistencia, DepartmentRepository.java regresa a VisionController.java que a continuación redirecciona hacia la vista vision.html que muestra al actor usuario la información actualizada de la visión.

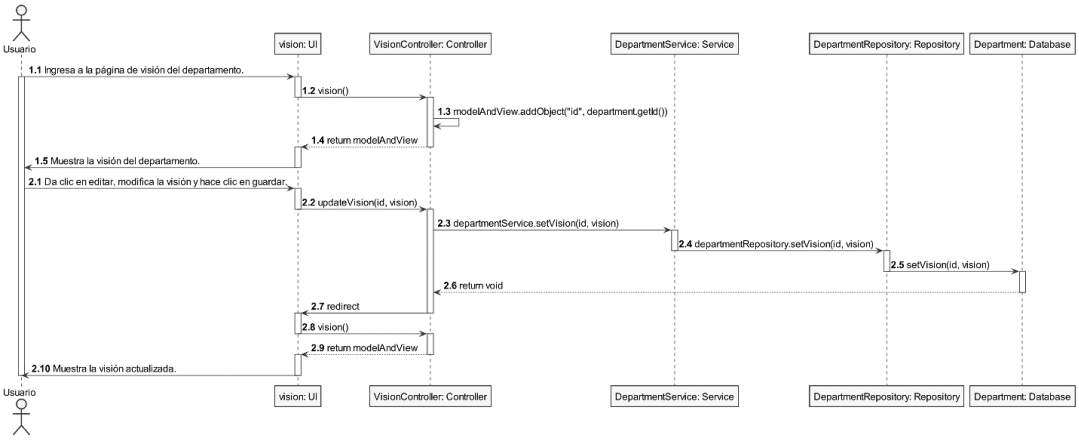

Figura 16. Diagrama de secuencia para actualizar la visión del departamento.

Agregar FODA del departamento

A continuación, se muestran los diagramas de secuencia para agregar las fortalezas/debilidades/oportunidades/amenazas del departamento dentro de la aplicación web (ver Figura 17, Figura 18, Figura 19 y Figura 20). El actor usuario le da clic al botón de agregar nueva fortaleza/debilidad/oportunidad/amenaza en la vista del análisis FODA (swot.html).

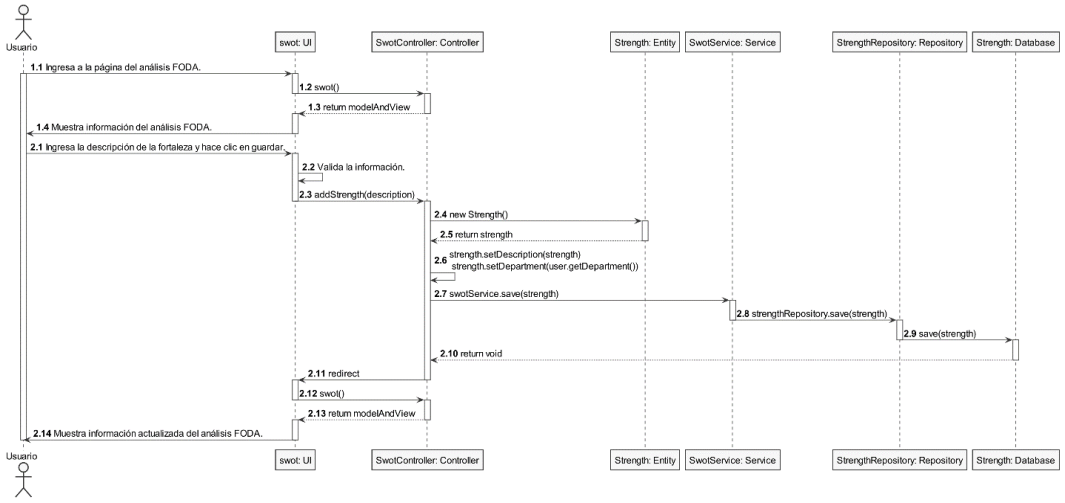

Figura 17. Diagrama de secuencia para agregar fortalezas del departamento.

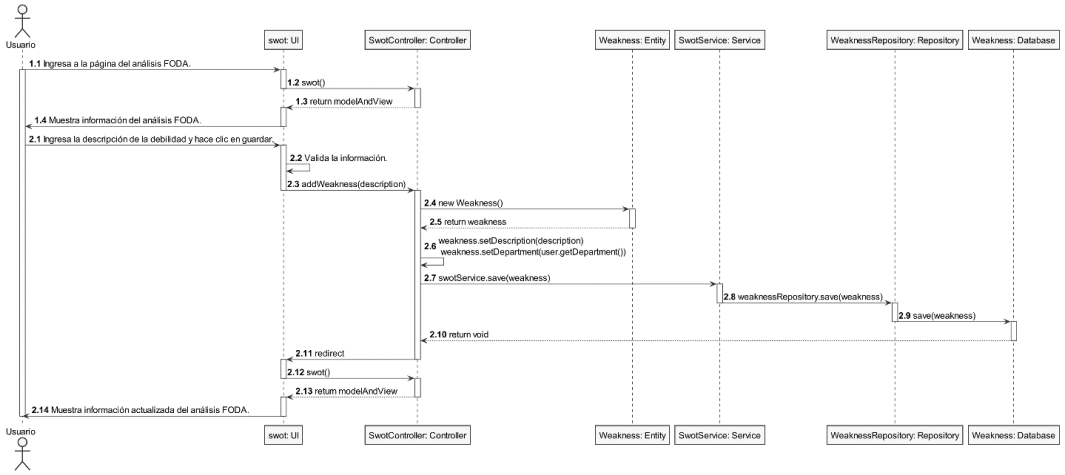

Figura 18. Diagrama de secuencia para agregar debilidades del departamento.

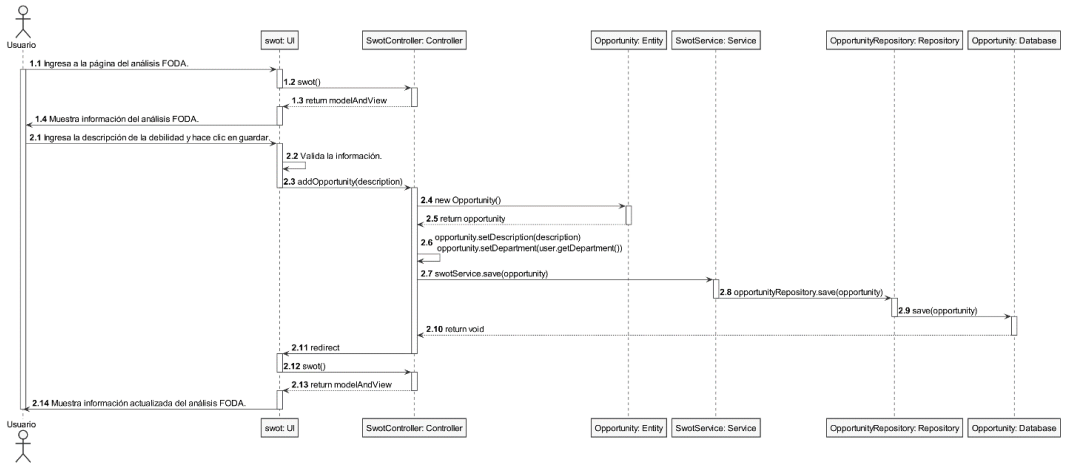

Figura 19. Diagrama de secuencia para agregar oportunidades del departamento.

Enseguida el sistema muestra una ventana donde el actor usuario introduce la información necesaria. El campo que el actor usuario debe introducir es la descripción de la fortaleza/debilidad/oportunidad/amenaza. Una vez ingresada la información, al hacer clic en guardar, se envían los datos de la fortaleza/debilidad/oportunidad/amenaza a SwotController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información.

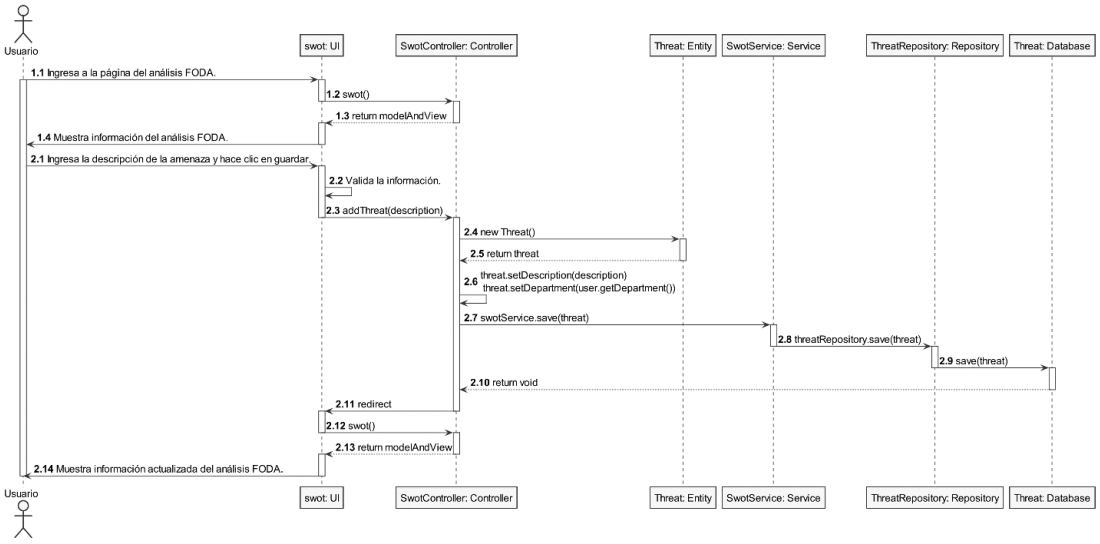

Figura 20. Diagrama de secuencia para agregar amenazas del departamento.

SwotController.java asigna el departamento al que pertenece el actor usuario a la fortaleza/debilidad/oportunidad/amenaza y ejecuta el método save de SwotService.java que así mismo utiliza el StrengthRepository.java/WeaknessRepository.java/ OpportunityRepository.java/ThreatRepository.java para conectarse con la tabla Strength/Weakness/Opportunity/Threat de la base de datos y guardar la fortaleza/debilidad/oportunidad/amenaza. Después de ejecutar la operación de persistencia, StrengthRepository.java/WeaknessRepository.java/OpportunityRepository.java/ThreatRepository.java regresa a SwotController.java que a continuación redirecciona hacia la vista swot.html que muestra al actor usuario la información de la fortaleza/debilidad/oportunidad/amenaza creada.

Actualizar FODA del departamento

A continuación, se muestran los diagramas de secuencia para la actualización de las fortalezas/debilidades/oportunidades/amenazas del departamento dentro de la

aplicación web (ver Figura 21, Figura 22, Figura 23 y Figura 24). El actor usuario le da clic al botón de editar fortaleza/debilidad/oportunidad/amenaza en la vista del análisis FODA (swot.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. El campo que el actor usuario puede modificar es la descripción de la fortaleza/debilidad/oportunidad/amenaza.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían como parámetro el identificador de la fortaleza/debilidad/oportunidad/amenaza y el texto de la fortaleza/debilidad/oportunidad/amenaza a SwotController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información. SwotController.java ejecuta el método setStrengthDescription/setWeaknessDescription/setOpportunityDescription/setThreatDescription de SwotService.java que así mismo utiliza el StrengthRepository.java/WeaknessRepository.java/OpportunityRepository.java/ThreatRepository.java para conectarse con la tabla Strength/Weakness/Opportunity/Threat de la base de datos y actualizar la fortaleza/debilidad/oportunidad/amenaza del departamento.

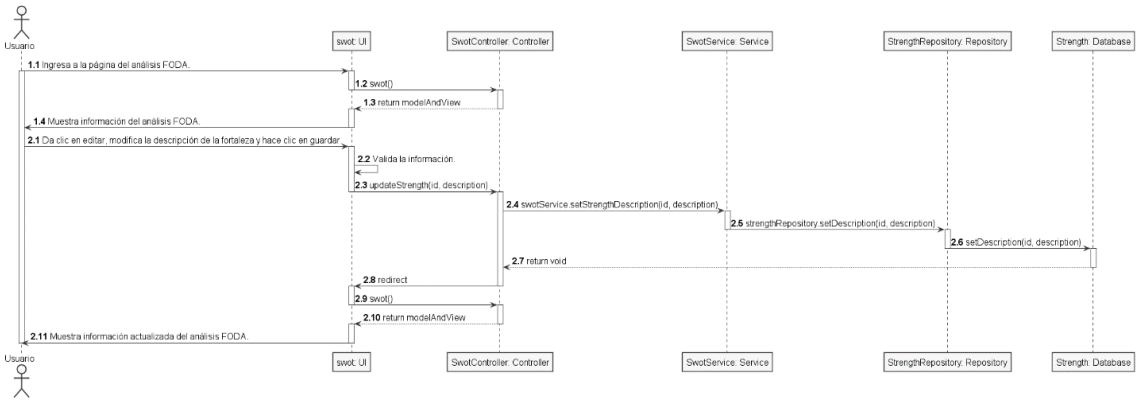

Figura 21. Diagrama de secuencia para actualizar fortalezas del departamento.

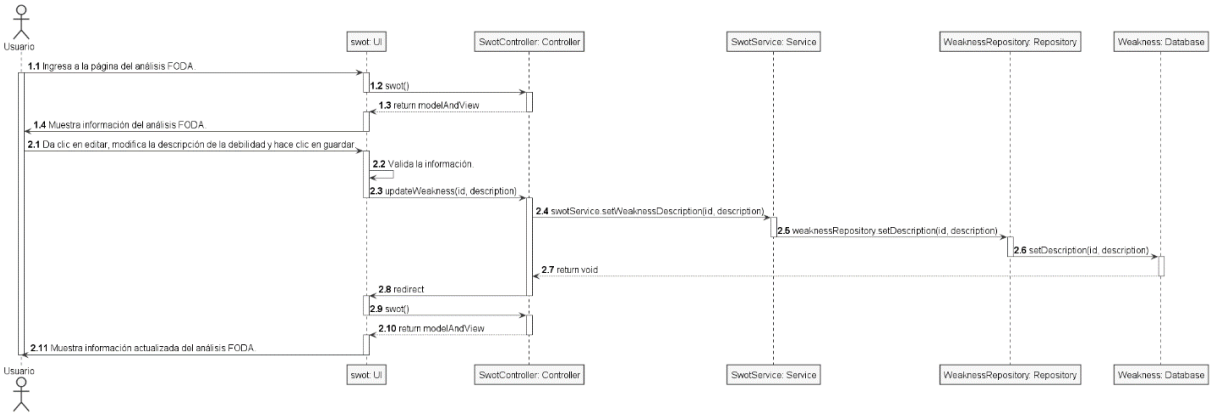

Figura 22. Diagrama de secuencia para actualizar debilidades del departamento.

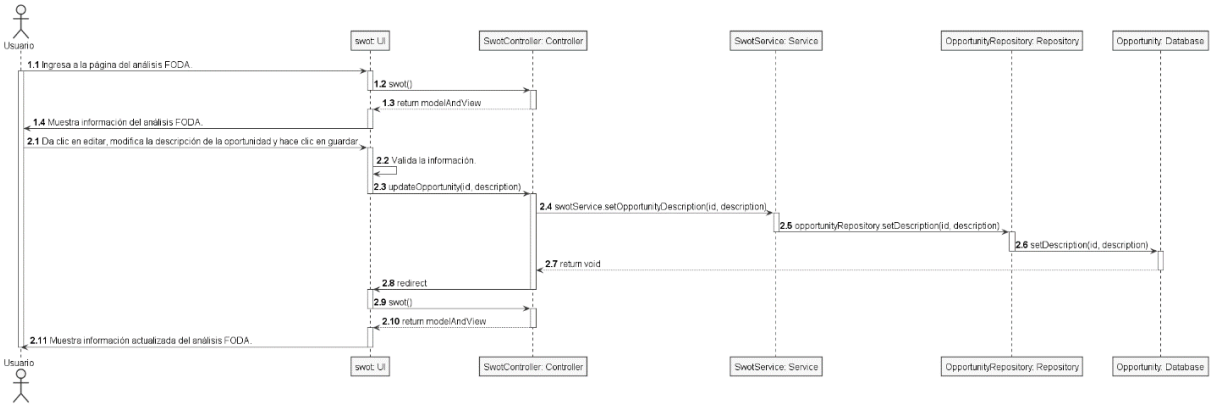

Figura 23. Diagrama de secuencia para actualizar oportunidades del departamento.

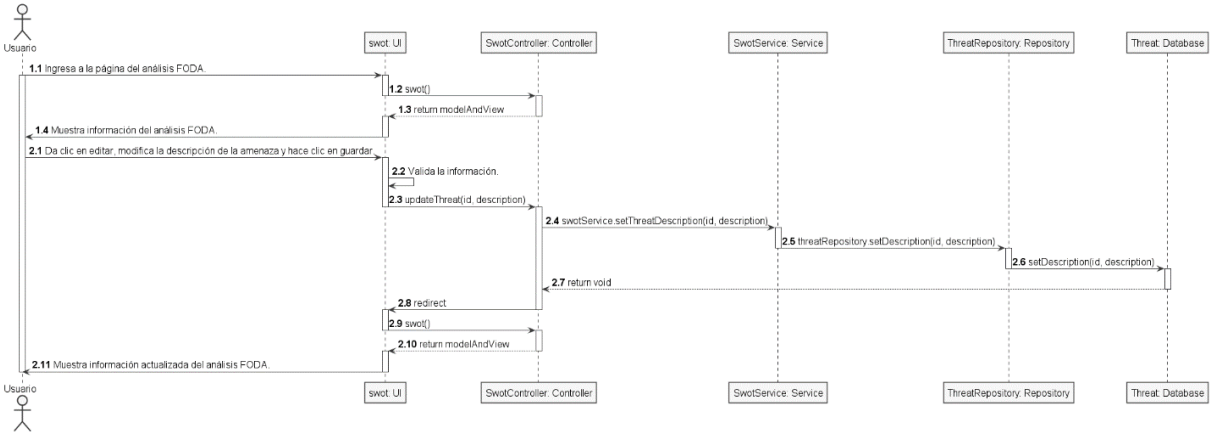

Figura 24. Diagrama de secuencia para actualizar amenazas del departamento.

Después de ejecutar la operación de persistencia, StrengthRepository.java/ WeaknessRepository.java/OpportunityRepository.java/ThreatRepository.java regresa a SwotController.java que a continuación redirecciona hacia la vista swot.html que muestra al actor usuario la información actualizada de la fortaleza/debilidad/oportunidad/amenaza.

Borrar FODA del departamento

A continuación, se muestran los diagramas de secuencia para borrar fortalezas/debilidades/oportunidades/amenazas dentro de la aplicación web (ver Figura 25, Figura 26, Figura 27 y Figura 28). El actor usuario le da clic al botón de borrar fortaleza/debilidad/oportunidad/amenaza en la vista del análisis FODA (swot.html).

Enseguida el sistema muestra una ventana donde se le pide al actor usuario confirmar la acción. Al hacer clic en borrar, se envía como parámetro el identificador de la fortaleza/debilidad/oportunidad/amenaza a SwotController.java, el cual inicia el proceso para borrar la fortaleza/debilidad/oportunidad/amenaza

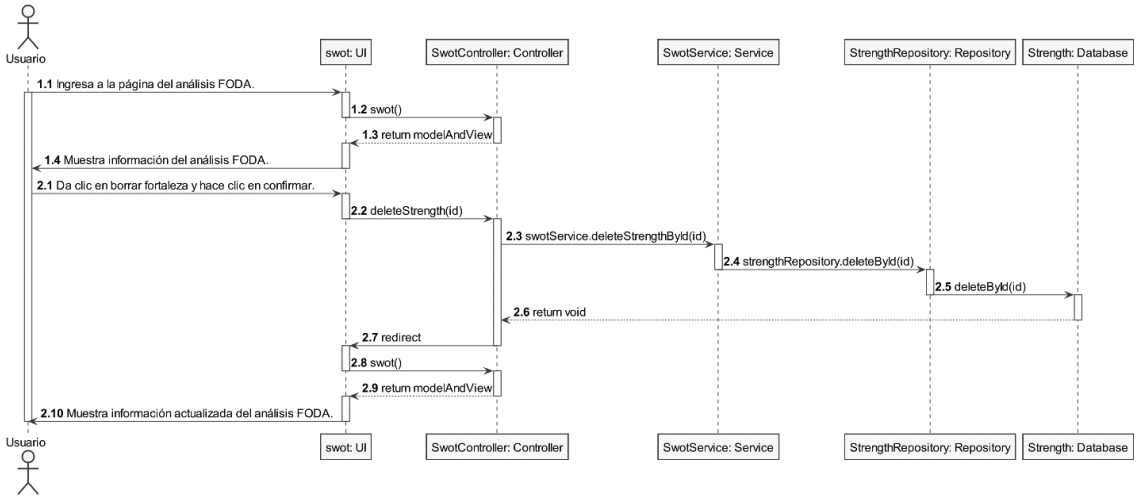

Figura 25. Diagrama de secuencia para borrar fortalezas del departamento.

SwotController.java ejecuta el método deleteStrengthById/deleteWeakness-ById/deleteOpportunityById/deleteThreatById de SwotService.java que así mismo utiliza el StrengthRepository.java/WeaknessRepository.java/OpportunityRepository.java/ ThreatRepository.java para conectarse con la tabla Strength/Weakness/Opportunity/ Threat de la base de datos y borrar la fortaleza/debilidad/oportunidad/amenaza.

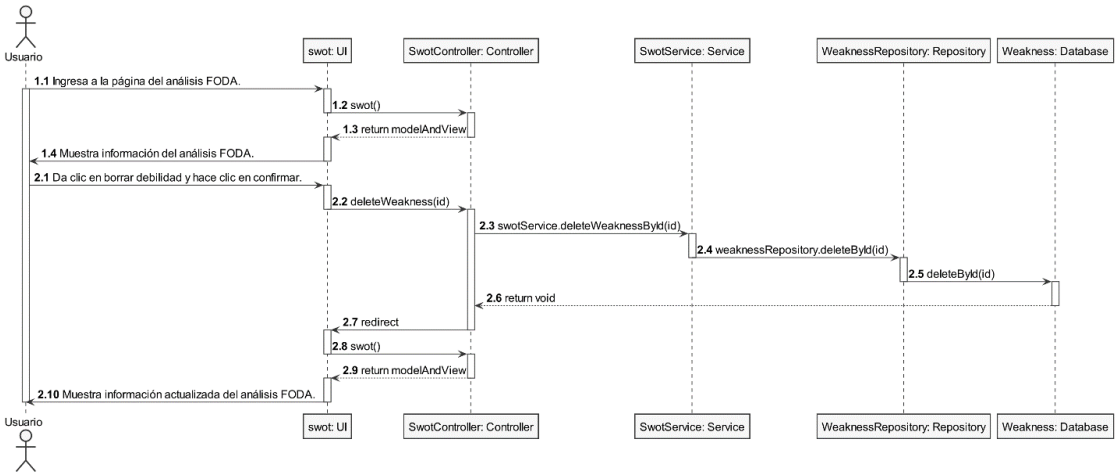

Figura 26. Diagrama de secuencia para borrar debilidades del departamento.

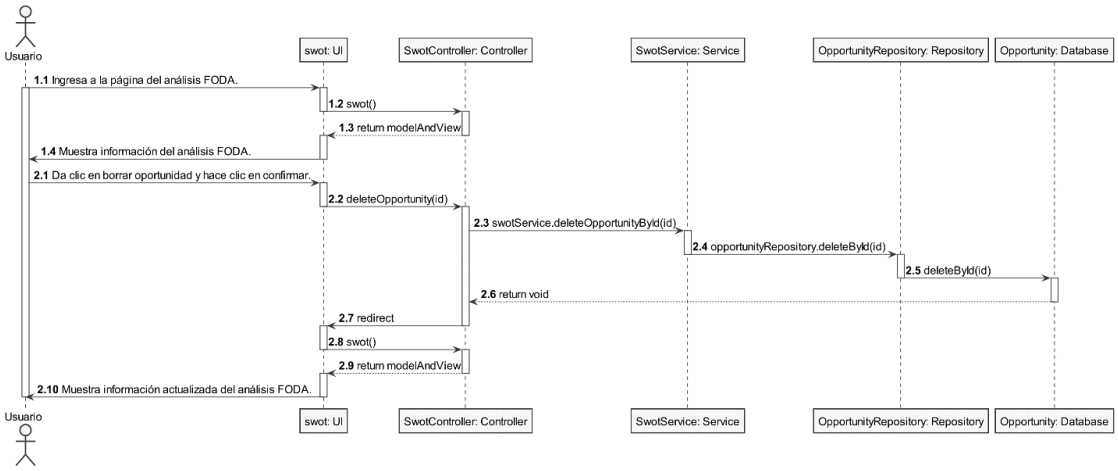

Figura 27. Diagrama de secuencia para borrar oportunidades del departamento.

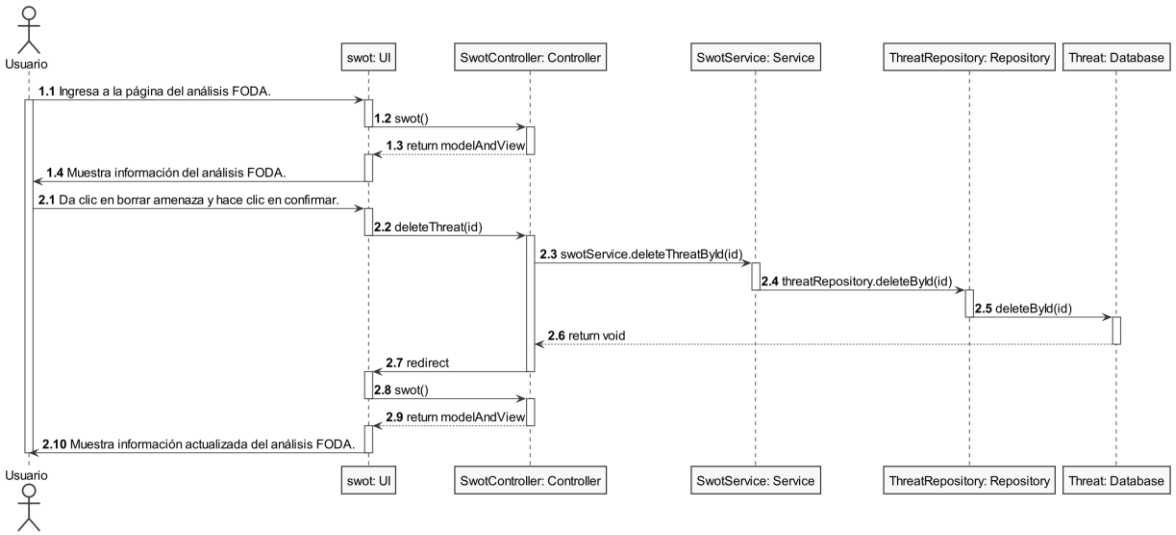

Figura 28. Diagrama de secuencia para borrar amenazas del departamento.

Después de ejecutar la operación de persistencia, StrengthRepository.java/ WeaknessRepository.java/OpportunityRepository.java/ThreatRepository.java regresa a SwotController.java que a continuación redirecciona hacia la vista swot.html que muestra al actor usuario la información actualizada del análisis FODA.

Crear objetivo

En la Figura 29 se muestra el diagrama de secuencia para la creación de objetivos dentro de la aplicación web. El actor usuario le da clic al botón de agregar nuevo objetivo en la vista de objetivos (goals.html). Enseguida el sistema muestra una ventana donde el actor usuario introduce la información necesaria. Los campos que el actor usuario debe introducir son la acción, la descripción, la cantidad, la métrica, el propósito, la fecha de inicio, la fecha de finalización, la meta más amplia, el proyecto de la iglesia, el presupuesto, el tipo de fondo y el impacto.

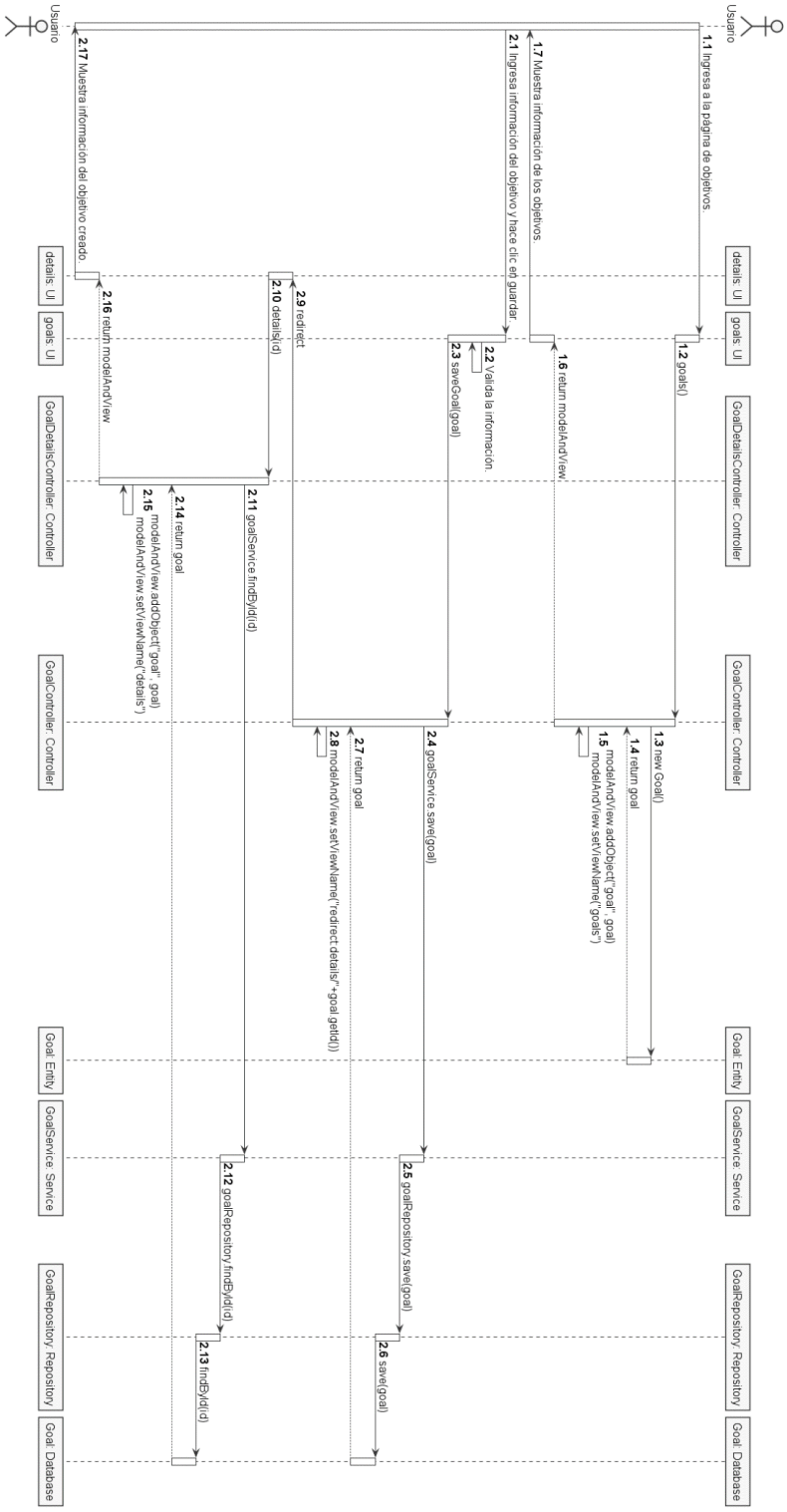

Figura 29. Diagrama de secuencia para agregar objetivos.

Una vez ingresada la información, al hacer clic en guardar, se envían los datos del objetivo a GoalController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. GoalController.java ejecuta el método save de GoalService.java que así mismo utiliza el GoalRepository.java para conectarse con la tabla Goal de la base de datos y guardar el objetivo.

GoalRepository.java regresa el objeto creado al GoalController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador del objetivo. GoalDetailsController.java toma el parámetro y se conecta con la base de datos por medio de GoalService.java para acceder a los datos del nuevo objetivo. Como resultado la vista details.html muestra al actor usuario la información del objetivo creado.

Actualizar información del objetivo

En la Figura 30 se muestra el diagrama de secuencia para la actualización de objetivos dentro de la aplicación web. El actor usuario le da clic al botón de editar objetivo en la vista de los objetivos (goals.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. Los campos que el actor usuario puede modificar son la acción, la descripción, la cantidad, la métrica, el propósito, la fecha de inicio, la fecha de finalización, la meta más amplia, el proyecto de la iglesia, el presupuesto, el tipo de fondo y el impacto.

Una vez hechos los cambios a la información, al hacer clic en guardar, se envían los datos del objetivo a GoalController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información.

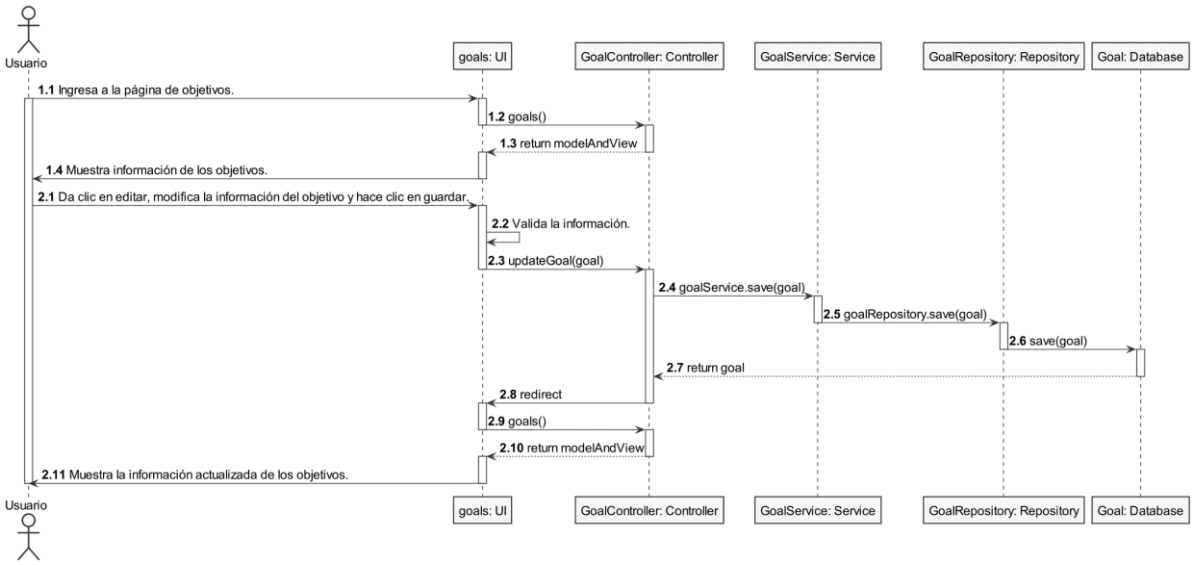

Figura 30. Diagrama de secuencia para actualizar objetivos.

GoalController.java ejecuta el método save de GoalService.java que así mismo utiliza el GoalRepository.java para conectarse con la tabla Goal de la base de datos y actualizar el objetivo. Después de ejecutar la operación de persistencia, GoalRepository.java regresa a GoalController.java que a continuación redirecciona hacia la vista goals.html que muestra al actor usuario la información actualizada del objetivo.

Borrar objetivo

En la Figura 31 se muestra el diagrama de secuencia para borrar objetivos dentro de la aplicación web. El actor usuario le da clic al botón de borrar objetivo en la vista de objetivos (goals.html). Enseguida el sistema muestra una ventana donde se le pide al actor usuario confirmar la acción. Al hacer clic en borrar, se envía como parámetro el identificador del objetivo a GoalController.java, el cual inicia el proceso para borrar el objetivo.

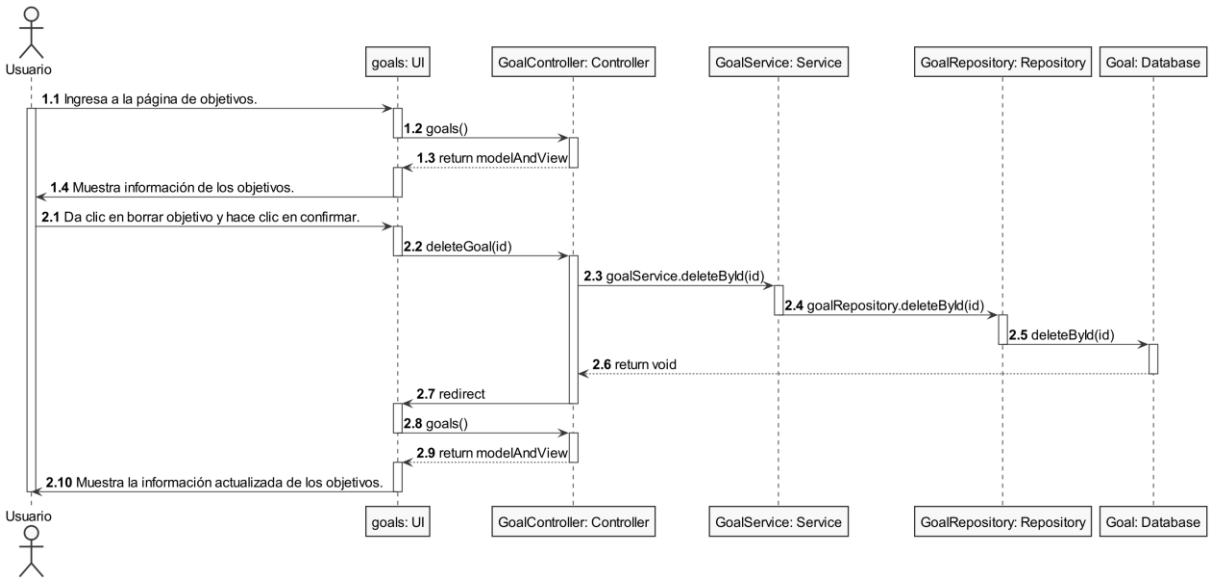

Figura 31. Diagrama de secuencia para borrar objetivos.

GoalController.java ejecuta el método deleteById de GoalService.java que así mismo utiliza el GoalRepository.java para conectarse con la tabla Goal de la base de datos y borrar el objetivo. Después de ejecutar la operación de persistencia, GoalRepository.java regresa a GoalController.java que a continuación redirecciona hacia la vista goals.html que muestra al actor usuario la información actualizada de los objetivos.

Actualizar KPI del objetivo

En la Figura 32 se muestra el diagrama de secuencia para la actualización del KPI del objetivo dentro de la aplicación web. El actor usuario le da clic al botón de editar KPI en la vista de detalles del objetivo (details.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. El campo que el actor usuario debe introducir es el nuevo valor del KPI.

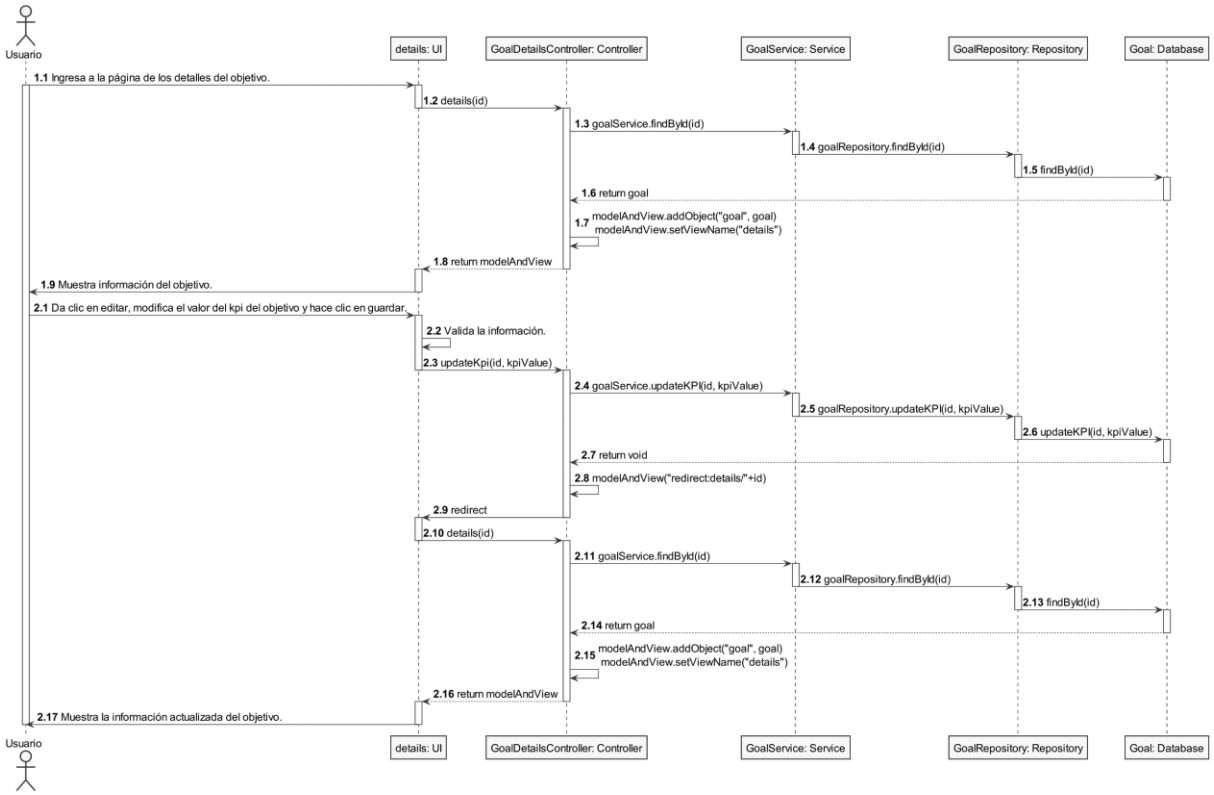

Figura 32. Diagrama de secuencia para actualizar KPI del objetivo.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos del KPI a GoalDetailsController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información. GoalDetailsController.java ejecuta el método updateKPI de GoalService.java que así mismo utiliza el GoalRepository.java para conectarse con la tabla Goal de la base de datos y actualizar el KPI del objetivo.

Después de ejecutar la operación de persistencia, GoalRepository.java regresa a GoalDetailsController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador del objetivo. GoalDetailsController.java toma el parámetro y se conecta con la base de datos por medio de GoalService.java para

acceder a los datos del objetivo. Como resultado la vista details.html muestra al actor usuario la información actualizada del KPI del objetivo.

Crear actividad

En la Figura 33 se muestra el diagrama de secuencia para la creación de actividades dentro de la aplicación web. El actor usuario le da clic al botón de agregar nueva actividad en la vista de detalles del objetivo (details.html). Enseguida el sistema muestra una ventana donde el actor usuario introduce la información necesaria. Los campos que el actor usuario debe introducir son la descripción, el presupuesto, la fecha de inicio, la fecha de finalización, la(s) persona(s) responsable(s) y el impacto.

Una vez ingresada la información, al hacer clic en guardar, se envían los datos de la actividad a ActivityController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. ActivityController.java ejecuta el método save de ActivityService.java que así mismo utiliza el ActivityRepository.java para conectarse con la tabla Activity de la base de datos y guardar la actividad.

ActivityRepository.java regresa el objeto creado al ActivityController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador del objetivo. GoalDetailsController.java toma el parámetro y se conecta con la base de datos por medio de GoalService.java para acceder a los datos del objetivo. Como resultado la vista details.html muestra al actor usuario la información de la actividad creada.

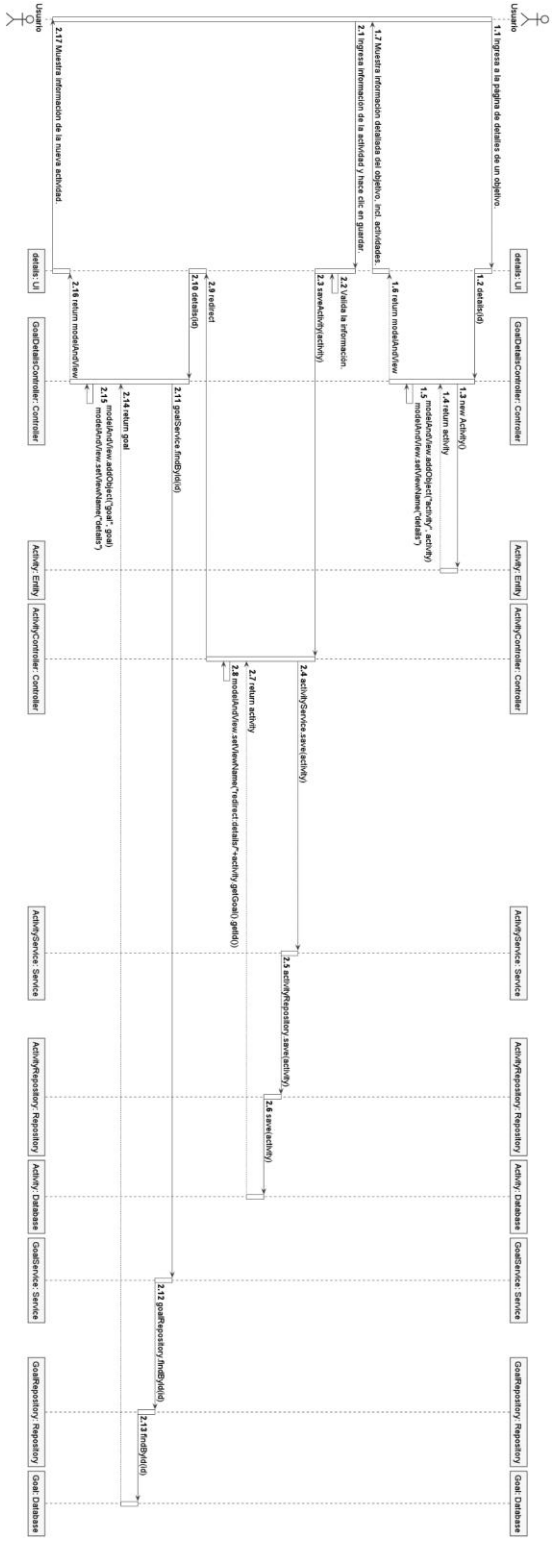

Figura 33. Diagrama de secuencia para agregar actividades.

Actualizar información de la actividad

En la Figura 34 se muestra el diagrama de secuencia para la actualización de actividades dentro de la aplicación web. El actor usuario le da clic al botón de editar actividad en la vista de detalles del objetivo (details.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. Los campos que el actor usuario puede modificar son la descripción, el presupuesto, la fecha de inicio, la fecha de finalización, la(s) persona(s) responsable(s) y el impacto.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos de la actividad a ActivityController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. ActivityController.java ejecuta el método save de ActivityService.java que así mismo utiliza el ActivityRepository.java para conectarse con la tabla Activity de la base de datos y actualizar la actividad.

Después de ejecutar la operación de persistencia, ActivityRepository.java regresa a ActivityController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador del objetivo. GoalDetailsController.java toma el parámetro y se conecta con la base de datos por medio de GoalService.java para acceder a los datos del objetivo. Como resultado la vista details.html muestra al actor usuario la información actualizada de la actividad.

Borrar actividad

En la Figura 35 se muestra el diagrama de secuencia para borrar actividades dentro de la aplicación web. El actor usuario le da clic al botón de borrar actividad en la vista de detalles del objetivo (details.html).

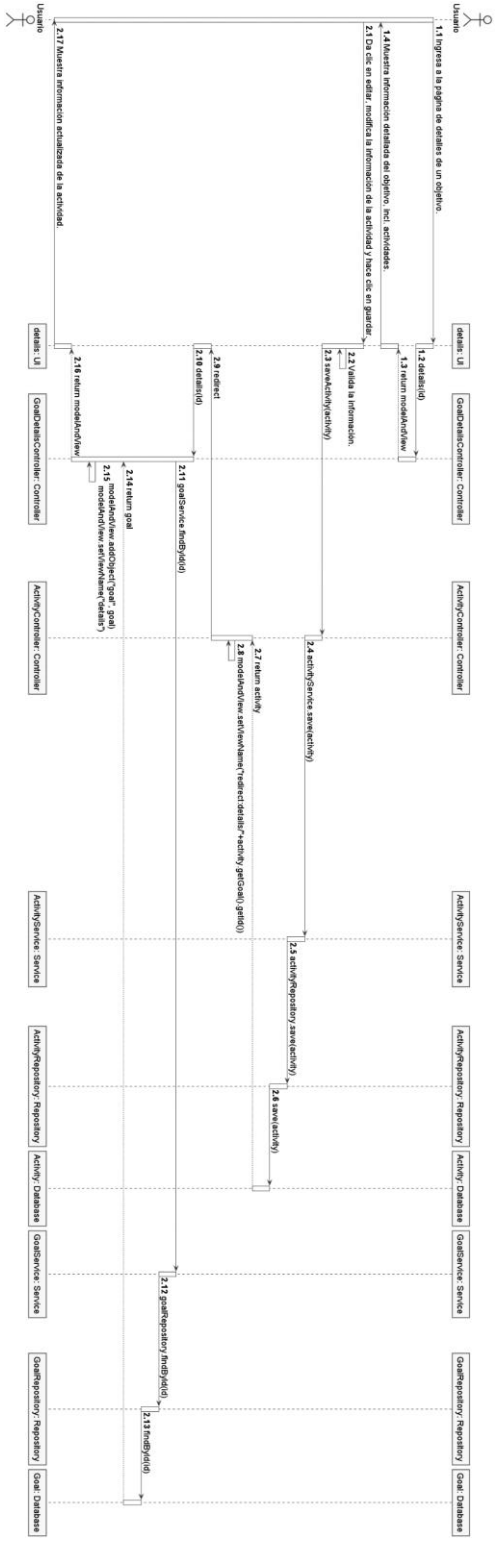

Figura 34. Diagrama de secuencia para actualizar actividades.

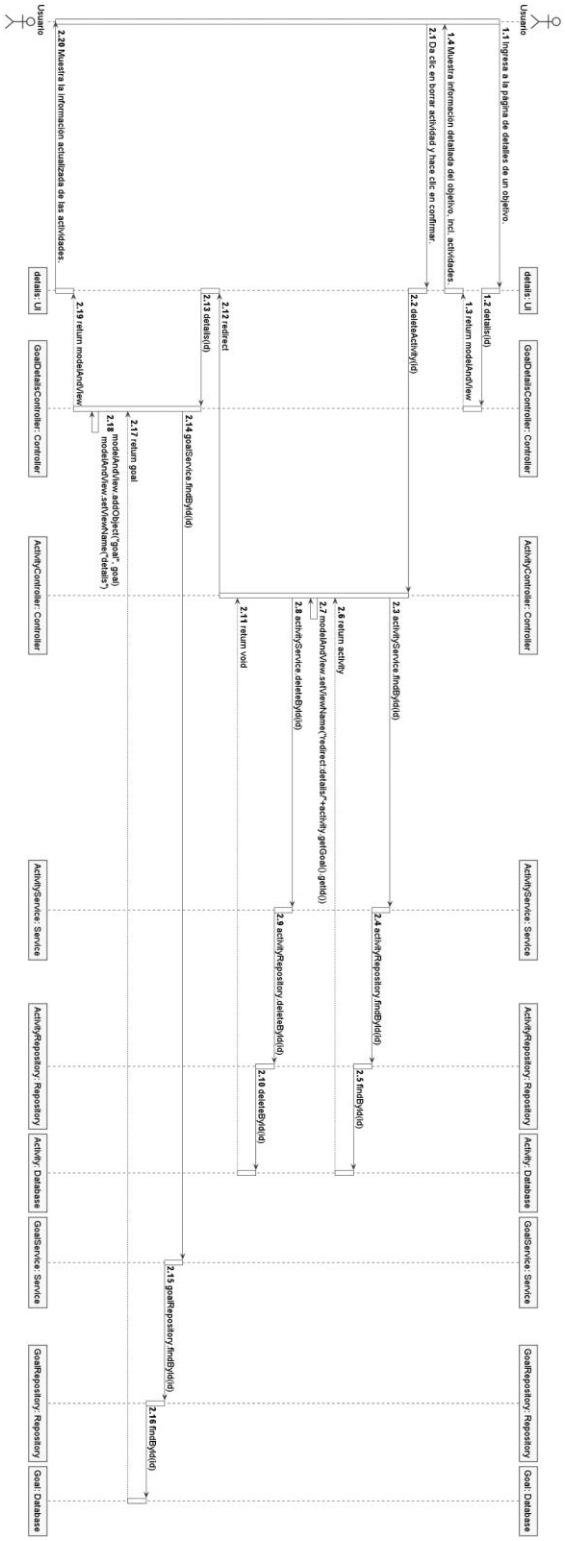

Figura 35. Diagrama de secuencia para borrar actividades.

Enseguida el sistema muestra una ventana donde se le pide al actor usuario confirmar la acción. Al hacer clic en borrar, se envía como parámetro el identificador de la actividad a ActivityController.java, el cual inicia el proceso para borrar la actividad.

ActivityController.java ejecuta el método findById de ActivityService.java que así mismo utiliza el ActivityRepository.java para conectarse con la tabla Activity de la base de datos y devolver la actividad a borrar. ActivityController.java asigna como parámetro el identificador del objetivo tomado de la actividad. Enseguida, ActivityController.java ejecuta el método deleteById para borrar la actividad.

Después de ejecutar la operación de persistencia, ActivityController.java redirecciona hacia la vista details.html, GoalDetailsController.java toma el parámetro y se conecta con la base de datos por medio de GoalService.java para acceder a los datos del objetivo. Como resultado la vista details.html muestra al actor usuario la información actualizada de las actividades.

Actualizar progreso de la actividad

En la Figura 36 se muestra el diagrama de secuencia para la actualización del progreso de la actividad dentro de la aplicación web. El actor usuario le da clic al botón de editar progreso en la vista de detalles de la actividad (details.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. El campo que el actor usuario debe introducir es el nuevo valor del progreso de la actividad.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos del progreso a ActivityDetailsController.java, el cual valida que los datos del campo se hayan ingresado e inicia el proceso para guardar la información.

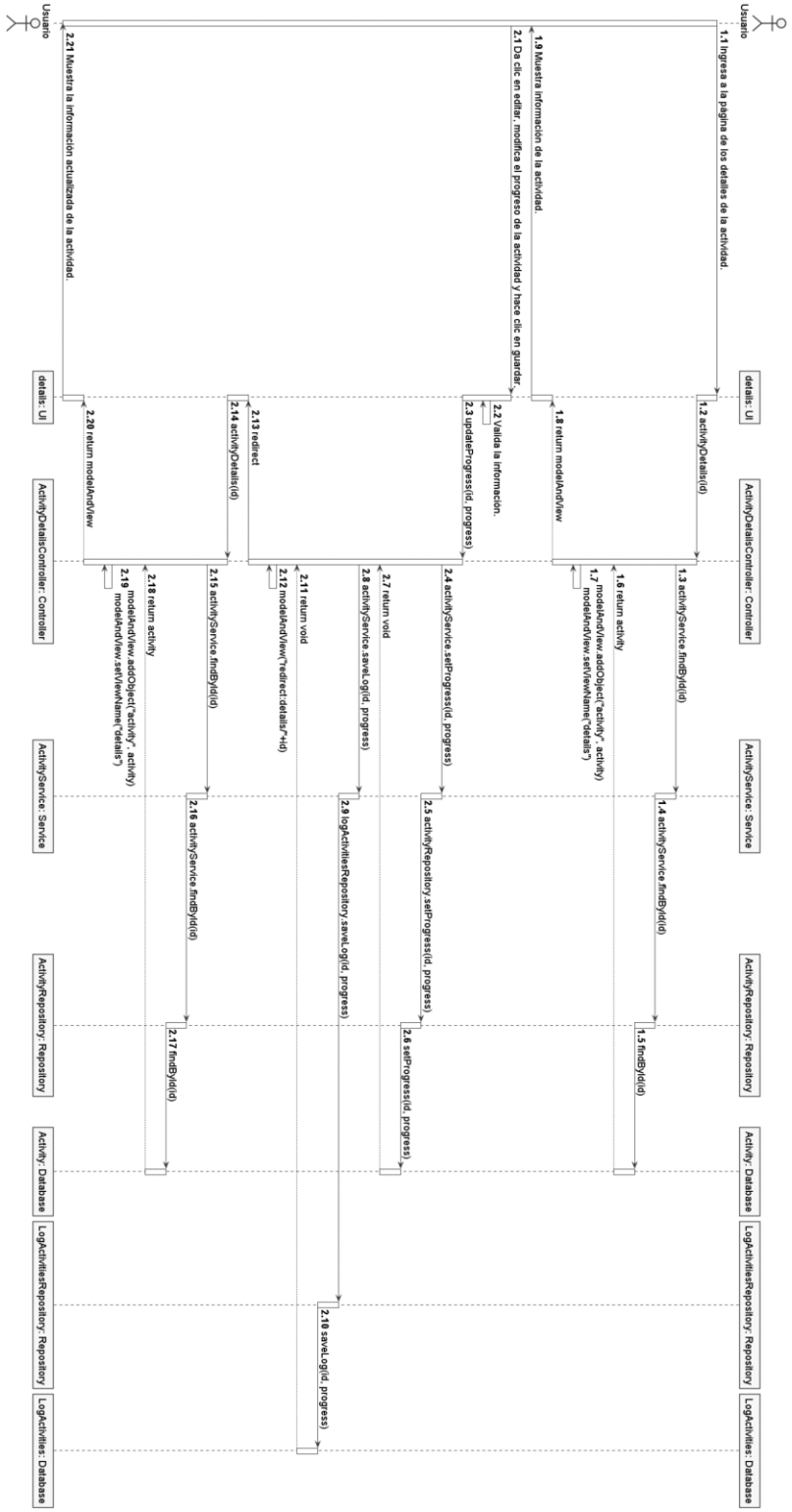

Figura 36. Diagrama de secuencia para actualizar el progreso de la actividad.

ActivityDetailsController.java ejecuta el método setProgress de ActivityService.java que así mismo utiliza el ActivityRepository.java para conectarse con la tabla Activity de la base de datos y actualizar el progreso de la actividad. Enseguida, ActivityDetailsController.java, ejecuta el método saveLog de ActivityService.java que así mismo utiliza el LogActivitiesRepository.java para conectarse con la tabla LogActivities de la base de datos y crear el log del progreso de la actividad.

Después de ejecutar las operaciones de persistencia, ActivityDetailsController.java redirecciona hacia la vista details.html pasando como parámetro el identificador de la actividad. ActivityDetailsController.java toma el parámetro y se conecta con la base de datos por medio de ActivityService.java para acceder a los datos de la actividad. Como resultado la vista details.html muestra al actor usuario la información actualizada del progreso de la actividad.

Crear elemento del presupuesto de la actividad

En la Figura 37 se muestra el diagrama de secuencia para la creación de elementos del presupuesto de la actividad dentro de la aplicación web. El actor usuario le da clic al botón de agregar nuevo elemento del presupuesto en la vista de detalles de la actividad (details.html). Enseguida el sistema muestra una ventana donde el actor usuario introduce la información necesaria. Los campos que el actor usuario debe introducir son la descripción y el presupuesto.

Una vez ingresada la información, al hacer clic en guardar, se envían los datos del elemento del presupuesto de la actividad a ActivityDetailsController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información.

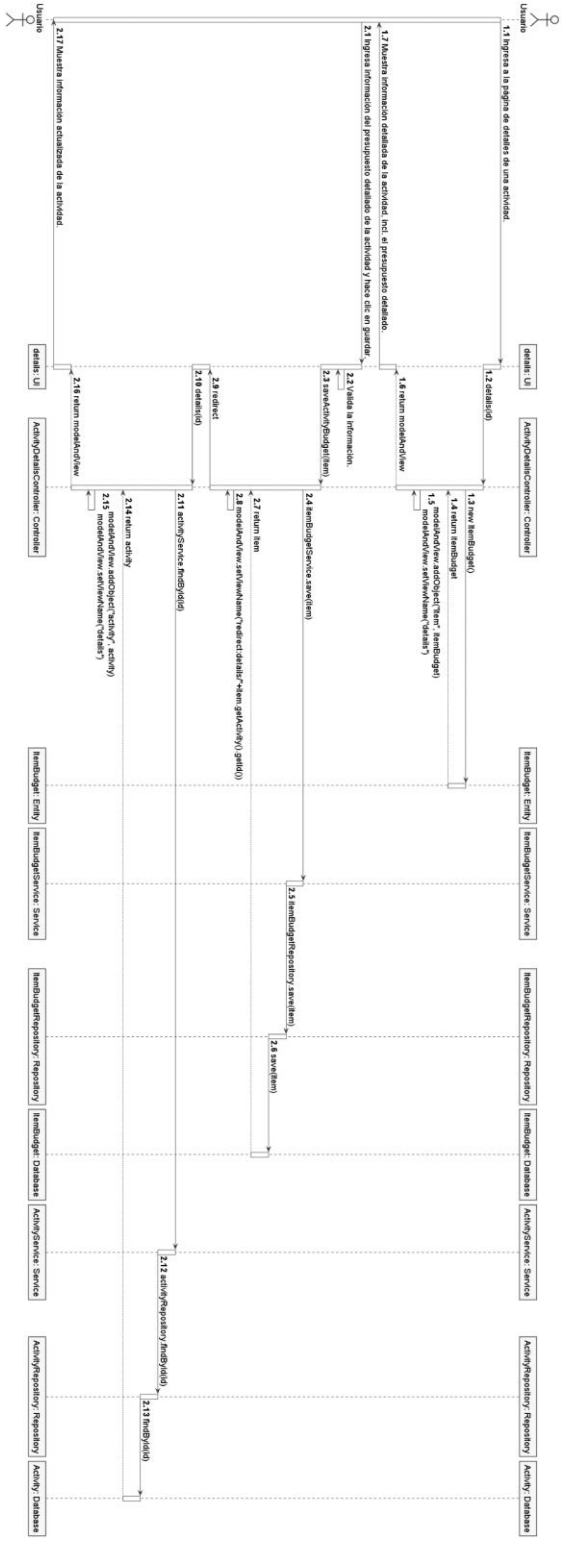

Figura 37. Diagrama de secuencia para agregar elementos del presupuesto.

ActivityDetailsController.java ejecuta el método save de ItemBudgetService.java que así mismo utiliza el ItemBudgetRepository.java para conectarse con la tabla ItemBudget de la base de datos y guardar el elemento del presupuesto de la actividad. ItemBudgetRepository.java regresa el objeto creado al ActivityDetailsController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador de la actividad. ActivityDetailsController.java toma el parámetro y se conecta con la base de datos por medio de ActivityService.java para acceder a los datos de la actividad. Como resultado la vista details.html muestra al actor usuario la información del elemento del presupuesto de la actividad creado.

Actualizar elemento del presupuesto de la actividad

En la Figura 38 se muestra el diagrama de secuencia para la actualización de elementos del presupuesto de la actividad dentro de la aplicación web. El actor usuario le da clic al botón de editar elemento del presupuesto en la vista de detalles de la actividad (details.html). Enseguida el sistema muestra una ventana donde el actor usuario modifica la información necesaria. Los campos que el actor usuario puede modificar son la descripción y el presupuesto.

Una vez hecho los cambios a la información, al hacer clic en guardar, se envían los datos del elemento del presupuesto de la actividad a ActivityDetailsController.java, el cual valida que los datos de todos los campos se hayan ingresado e inicia el proceso para guardar la información. ActivityDetailsController.java ejecuta el método save de ItemBudgetService.java que así mismo utiliza el ItemBudgetRepository.java para conectarse con la tabla ItemBudget de la base de datos y actualizar el elemento del presupuesto de la actividad.

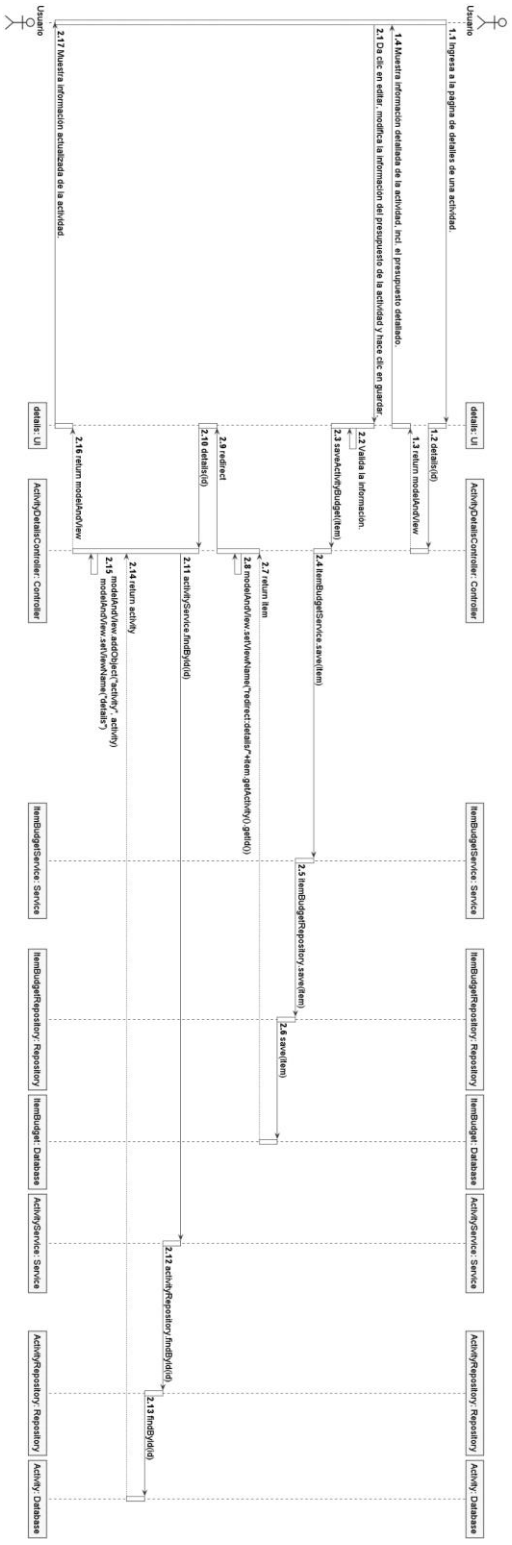

Figura 38. Diagrama de secuencia para actualizar elementos del presupuesto.

Después de ejecutar la operación de persistencia, ItemBudgetRepository.java regresa a ActivityDetailsController.java que a continuación redirecciona hacia la vista details.html pasando como parámetro el identificador de la actividad. ActivityDetails-Controller.java toma el parámetro y se conecta con la base de datos por medio de ActivityService.java para acceder a los datos de la actividad. Como resultado la vista details.html muestra al actor usuario la información actualizada del elemento del presupuesto de la actividad.

Borrar elemento del presupuesto de la actividad

En la Figura 39 se muestra el diagrama de secuencia para borrar elementos del presupuesto de la actividad dentro de la aplicación web. El actor usuario le da clic al botón de borrar elemento del presupuesto en la vista de detalles de la actividad (details.html).

Enseguida el sistema muestra una ventana donde se le pide al actor usuario confirmar la acción. Al hacer clic en borrar, se envía como parámetro el identificador del elemento del presupuesto a ActivityDetailsController.java, el cual inicia el proceso para borrar la actividad.

ActivityDetailsController.java ejecuta el método findById de ItemBudgetService.java que así mismo utiliza el ItemBudgetRepository.java para conectarse con la tabla ItemBudget de la base de datos y devolver el elemento a borrar. ActivityDetails-Controller.java asigna como parámetro el identificador de la actividad tomado del elemento. Enseguida, ActivityDetailsController.java ejecuta el método deleteById para borrar el elemento del presupuesto.

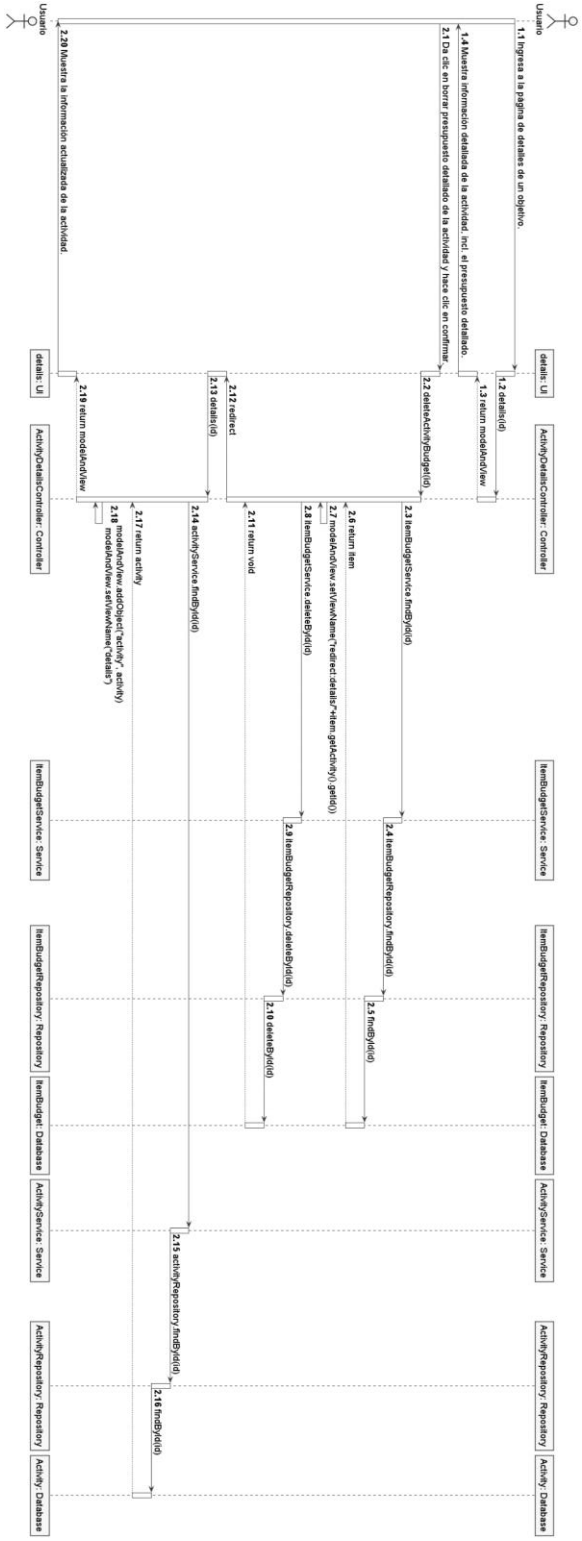

Figura 39. Diagrama de secuencia para borrar elementos del presupuesto.

Después de ejecutar las operaciones de persistencia, ActivityController.java redirecciona hacia la vista details.html, ActivityDetailsController.java toma el parámetro y se conecta con la base de datos por medio de ActivityService.java para acceder a los datos de la actividad. Como resultado la vista details.html muestra al actor usuario la información actualizada de los elementos del presupuesto de la actividad.

Desarrollo de la aplicación

En esta etapa se desarrolló la aplicación web para la gestión del plan estratégico de la Universidad de Montemorelos. A continuación, se describen las tecnologías escogidas para el desarrollo del backend y frontend del producto de software.

En primer lugar, se empleó Java como lenguaje de programación y el framework spring para el desarrollo de la aplicación debido a la rapidez, flexibilidad, productividad, seguridad y soporte que tiene para la creación de aplicaciones web modernas. Asimismo, se utilizó Thymeleaf como motor de plantillas Java para generar páginas HTML debido a su alta compatibilidad con el framework spring.

Además, se empleó JPA para manejar la conexión de la aplicación con la base de datos, principalmente porque JPA ayuda a reducir el tiempo de la definición del mapeo entre las tablas de la base de datos y el modelo, entre otras funciones, dando como resultado un desarrollo más ágil del software. Finalmente, se utilizó PostgreSQL por su velocidad, seguridad y robustez como base de datos, siendo ésta una de las opciones más atractivas de código abierto.

Siguiendo el patrón MVC de arquitectura de software, se crearon los componentes necesarios para la interacción entre el usuario y la aplicación. Para esto, se estableció cada una de las capas necesarias para cumplir con las acciones definidas.

Para empezar, se trabajó en el modelo, siendo este la representación de la información necesaria para el funcionamiento correcto de la aplicación. Para ello, se crearon entidades asociadas a las tablas en la base de datos utilizando JPA.

La Figura 40 muestra como ejemplo una porción de código de la clase Goal.java donde se puede apreciar la definición de los atributos y las anotaciones correspondientes para su uso con JPA. La anotación @Entity especifica que esta clase es una entidad, esto significa que JPA creará una tabla a partir de ella. La anotación @Table especifica el nombre que tendrá la entidad como tabla en la base de datos. La anotación @Id asigna el atributo ID como la llave primaria de la entidad.

```
1  @Entity
2  @Table(name = "goal")
3  public class Goal {
4      @Id
5      @Column(name = "goal_id")
6      private Long id;
7
8      @Column(name = "description")
9      private String description;
10
11     @Column(name = "kpi_value")
12     private int kpiValue;
13
14     @Column(name = "start_date")
15     private Date startDate;
16
17     @Column(name = "end_date")
18     private Date endDate;
19
20     @OneToMany(fetch = FetchType.LAZY, mappedBy = "goal", cascade = {CascadeType.DE-
       TACH, CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH})
21     private List<Activity> activities;
22 }
```

Figura 40. Fragmento de código de la clase Goal.java.

La anotación @Column especifica la columna asignada para los atributos de la entidad. La anotación @OneToMany especifica una relación de uno a muchos entre las entidades de objetivos y actividades. La propiedad fetch especifica de qué manera se cargarán los datos. La propiedad mappedBy especifica qué atributo se usa para representar la clase principal en la clase secundaria. La propiedad cascade especifica las operaciones de persistencia que se propagarán a las entidades relacionadas.

Enseguida, se crearon los componentes de repositorios y los servicios. Los repositorios se encargan de gestionar todas las operaciones de persistencia entre las entidades y las tablas de la base de datos. La Figura 41 muestra una porción de código de la interfaz GoalRepository.java que extiende de JPA los métodos para la persistencia de datos. La anotación @Repository indica que la interfaz es un repositorio JPA. La anotación @Modifying indica que un método de consulta debe considerarse como una consulta de modificación siendo esta un UPDATE. La anotación @Query se utiliza para declarar consultas directamente en los métodos del repositorio. La propiedad value especifica la consulta a realizar. La propiedad nativeQuery especifica que la consulta será declarada en el lenguaje nativo del motor de la base de datos.

```
1 @Repository("goalRepository")
2 public interface GoalRepository extends JpaRepository<Goal, Long> {
3     @Modifying
4     @Query(
5         value = "UPDATE goal SET kpi_value = ?2 WHERE goal_id = ?1",
6         nativeQuery = true)
7     void updateKPI(Long id, int kpiValue);
8 }
```

Figura 41. Fragmento de código de la interfaz GoalRepository.java.

Los servicios se crearon para facilitar la ejecución de las funciones definidas en los repositorios al hacer uso de ellas en los controladores. Para esto se crearon interfaces donde se declaran los métodos que contienen la lógica de las operaciones de persistencia. En la Figura 42 se muestra una porción de código de la interfaz GoalService.java donde están declarados los métodos que se utilizan para realizar las operaciones básicas de persistencia. Dentro de los métodos declarados está updateKPI que actualiza el valor del KPI de un objetivo específico. También está el método findById que busca y regresa la información de un objetivo específico. Además, está el método save que guarda y el método deleteById que borra un objetivo específico.

```
1 public interface GoalService {
2     public void updateKPI(Long id, int kpiValue);
3
4     public Goal findById(Long id);
5
6     public Goal save(Goal goal);
7
8     public void deleteById(Long id);
9 }
```

Figura 42. Fragmento de código de la interfaz GoalService.java.

Por otro lado, se crearon las clases que implementan la lógica de los servicios. La Figura 43 muestra una porción de código de la clase GoalServiceImpl.java donde están los métodos que aplican la lógica de las operaciones de persistencia. La anotación @Service indica que la clase anotada es un servicio, esto quiere decir que esta clase almacena la lógica de negocio de la aplicación. La anotación @Autowired marca al método constructor de la clase para hacer la inyección de dependencias con el propósito de utilizar sus funcionalidades en otras clases. La anotación @Override indica

que un método en una subclase está reemplazando el comportamiento heredado, en este caso se sobrescriben los métodos de la interfaz GoalService. La anotación @Transactional define el alcance de la transacción de la base de datos dentro del contexto de persistencia. Esta anotación se utiliza debido a que el método updateKPI en la línea 13 es independiente de los demás que ya cuentan con esta anotación porque pertenecen a los métodos por defecto del JpaRepository.

Después, se crearon los controladores, estos responden a las entradas de los usuarios invocando la lógica de actualización del modelo o vista. La Figura 44 muestra una porción de código de la clase GoalController.java donde están declarados los métodos que responden a las solicitudes de los usuarios y realizan acciones de gestión de la información de los objetivos como mostrar, crear, actualizar o borrar un registro en la base de datos. La anotación @Controller indica que la clase es un controlador. La anotación @RequestMapping sirve para mapear las solicitudes web en métodos de clases. La anotación @GetMapping sirve para mapear solicitudes HTTP GET en los métodos del controlador.

Asimismo, la anotación @PostMapping sirve para mapear solicitudes HTTP POST en los métodos del controlador. En la línea 12 de la Figura 44 se declara el método que atenderá las solicitudes de los usuarios al momento de entrar en la vista de objetivos. En la línea 17 se agrega el objeto goals que contiene la lista de los objetivos del usuario. Además, en la línea 19 se asigna la vista que será mostrada al actor usuario como respuesta a la solicitud. En las líneas 24, 35 y 46 se encuentran declarados los métodos que atenderán a las solicitudes de guardar, actualizar y borrar objetivos, respectivamente.

```
1 @Service("goalService")
2 public class GoalServiceImpl implements GoalService {
3
4     private GoalRepository goalRepository;
5
6     @Autowired
7     public GoalServiceImpl(GoalRepository goalRepository) {
8         this.goalRepository = goalRepository;
9     }
10
11    @Override
12    @Transactional
13    public void updateKPI(Long id, int kpiValue) {
14        goalRepository.updateKPI(id, kpiValue);
15    }
16
17    @Override
18    public Goal findById(Long id) {
19        return goalRepository.findById(id);
20    }
21
22    @Override
23    public Goal save(Goal goal) {
24        return goalRepository.save(goal);
25    }
26
27    @Override
28    public void deleteById(Long id) {
29        goalRepository.deleteById(id);
30    }
31
32 }
```

Figura 43. Fragmento de código de la clase GoalServiceImpl.java.

```
1  @Controller
2  @RequestMapping("/strategy/goal")
3  public class GoalController {
4
5      @Autowired
6      private UserService userService;
7
8      @Autowired
9      private GoalService goalService;
```

```
10
11      @GetMapping(value="/goals")
12      public ModelAndView goals(){
13          ModelAndView modelAndView = new ModelAndView();
14          Authentication auth = SecurityContextHolder.getContext().getAuthentication();
15          User user = userService.findByEmail(auth.getName());
16          modelAndView.addObject("goal", new Goal());
17          modelAndView.addObject("goals", user.getGoals());
18          modelAndView.addObject("department", user.getDepartment().getName());
19          modelAndView.setViewName("strategy/goal/goals");
20          return modelAndView;
21      }
22
23      @PostMapping(value = "/save")
24      public ModelAndView saveGoal(@Valid Goal goal) {
25          ModelAndView modelAndView = new ModelAndView();
26          Authentication auth = SecurityContextHolder.getContext().getAuthentication();
27          User user = userService.findByEmail(auth.getName());
28          goal.setUser(user);
29          Goal newGoal = goalService.save(goal);
30          modelAndView.setViewName("redirect:/strategy/goal/details/"+newGoal.getId());
31          return modelAndView;
32      }
33
34      @PostMapping(value= "/update")
35      public ModelAndView updateGoal(@Valid Goal goal){
36          ModelAndView modelAndView = new ModelAndView();
37          Authentication auth = SecurityContextHolder.getContext().getAuthentication();
38          User user = userService.findByEmail(auth.getName());
39          goal.setUser(user);
40          goalService.save(goal);
41          modelAndView.setViewName("redirect:/strategy/goal/goals");
42          return modelAndView;
43      }
44
45      @GetMapping(value= "/delete/{id}")
46      public ModelAndView deleteGoal(@PathVariable Long id){
47          ModelAndView modelAndView = new ModelAndView();
48          goalService.deleteById(id);
49          modelAndView.setViewName("redirect:/strategy/goal/goals");
50          return modelAndView;
51      }
52
53  }
```

Figura 44. Fragmento de código de la clase GoalController.java.

Finalmente, se crearon las vistas en HTML con Thymeleaf para definir cómo se deben mostrar los datos del modelo. La Figura 45 muestra una porción de la vista goals.html donde se expone la información de los objetivos del actor usuario. En la línea 15 está el botón para agregar un nuevo objetivo. En la línea 24 se crea la tabla que muestra la información de los objetivos al actor usuario. En la línea 44 se toma el objeto goals que fue mandado por el controlador GoalController.java y se recorre para obtener y mostrar la información de cada objetivo específico. En la línea 46 se muestra el nombre del objetivo como enlace que lleva a la página para modificar la información detallada del objetivo, que incluye las actividades. En la línea 59 está el botón de editar el objetivo. Por último, en la línea 64 se encuentra el botón para borrar el objetivo, este botón solo será visible si el objetivo no cuenta con actividades.

```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <body>
4    <div class="d-sm-flex align-items-center justify-content-between mb-4">
5      <h1 class="h3 mb-0 font-weight-bold text-gray-800">
6        <i class="fas fa-fw fa-bullseye"></i>
7        Objetivos - <span class="h4" th:text="${department}"></span>
8      </h1>
9    </div>
10
11   <!-- Table Goals -->
12   <div class="card shadow mb-4">
13     <div class="card-header py-3 d-flex flex-row align-items-center justify-content-be-
tween">
14       <h6 class="m-0 font-weight-bold text-primary">Objetivos</h6>
15       <a href="#" class="btn btn-sm btn-primary btn-icon-split shadow-sm" data-tog-
gle="modal" data-target="#addModal">
16         <span class="icon text-white-50">
17           <i class="fas fa-plus"></i>
18         </span>
19         <span class="text">Agregar</span>
```

```
20        </a>
21      </div>
22      <div class="card-body">
23        <div class="table-responsive">
24          <table class="table table-bordered" width="100%" cellspacing="0">
25            <thead>
26              <tr>
27                <th><i class="fas fa-bullseye"></i> Objetivo</th>
28                <th><i class="fas fa-calendar-alt"></i> Inicio</th>
29                <th><i class="fas fa-calendar-alt"></i> Fin</th>
30                <th><i class="fas fa-hourglass-start"></i> Estatus</th>
31                <th><i class="fas fa-wrench"></i> Opción</th>
32              </tr>
33            </thead>
34            <tfoot>
35              <tr>
36                <th><i class="fas fa-bullseye"></i> Objetivo</th>
37                <th><i class="fas fa-calendar-alt"></i> Inicio</th>
38                <th><i class="fas fa-calendar-alt"></i> Fin</th>
39                <th><i class="fas fa-hourglass-start"></i> Estatus</th>
40                <th><i class="fas fa-wrench"></i> Opción</th>
41              </tr>
42            </tfoot>
43            <tbody>
44              <tr th:each="goal : ${goals}">
45                <td>
46                  <a th:href="@{/strategy/goal/details/{id}(id=${goal.id})}" class="text-
decoration-none" th:text="${goal.name}"></a>
47                </td>
48                <td th:text="${#dates.format(goal.startDate, 'dd-MMM-yyyy')}"></td>
49                <td th:text="${#dates.format(goal.endDate, 'dd-MMM-yyyy')}"></td>
50                <td style="text-align: center;">
51                  <span th:if="${goal.active}">
52                    <span class="badge badge-primary"> Activo</span>
53                  </span>
54                  <span th:unless="${goal.active}">
55                    <span class="badge badge-secondary">Inactivo</span>
56                  </span>
57                </td>
58                <td>
59                  <a href="#" class="btn btn-sm btn-info shadow-sm" data-tog-
gle="modal" data-target="#editModal">
60                    <span class="icon text-white-50">
61                      <i class="fas fa-edit"></i>
62                    </span>
```

```
63                 </a>
64                 <a th:if="${#lists.isEmpty(goal.activities)}" href="#" class="btn btn-
sm btn-danger shadow-sm" data-toggle="modal" data-target="#deleteModal">
65                   <span class="icon text-white-50">
66                     <i class="fas fa-trash"></i>
67                   </span>
68                 </a>
69               </td>
70             </tr>
71           </tbody>
72         </table>
73       </div>
74     </div>
75   </div>
76 </body>
77 </html>
```

Figura 45. Fragmento de código de la vista goals.html.

Pruebas previas

Se realizaron pruebas piloto de la aplicación por parte de dos analistas de sistemas del Instituto de Ciencia de Datos para verificar el funcionamiento correcto de la herramienta. Estas pruebas de software tuvieron el objetivo de encontrar cualquier detalle en el proceso de calidad de la herramienta.

Específicamente, se realizaron pruebas de caja negra que se centran en los requisitos funcionales del software, para analizar el flujo de trabajo de la aplicación web y cerciorarse de que conforme a una serie de entradas se devuelvan los valores esperados. En el Apéndice A se describe la documentación de los casos de uso como requisitos funcionales del software que se siguieron para el desarrollo de las pruebas. En la documentación se describe la información de los actores de la siguiente manera:

1. Nombre del actor: cada actor recibe un nombre.
2. Descripción: esta sección describe al actor.

Además, se describen los requisitos funcionales de la siguiente manera:

1. Nombre del caso de uso: a cada caso de uso se le asigna un nombre.

2. Resumen: esta sección describe brevemente el caso de uso.

3. Actores: esta sección nombra a los actores en el caso de uso.

4. Precondición: esta sección especifica una o más condiciones que deben cumplirse al comienzo del caso de uso.

5. Descripción: es una descripción de la secuencia principal del caso de uso. Es decir, es la secuencia más habitual de interacciones entre los actores y el sistema.

6. Alternativas: esta sección proporciona una descripción de las posibles ramas alternativas de la secuencia principal.

7. Postcondición: esta sección identifica la condición que debe cumplirse siempre al final del caso de uso si se ha seguido la secuencia principal.

Despliegue de la herramienta

Gracias al enfoque que tiene spring boot con respecto a la facilidad de desarrollo y despliegue de software en producción, se lanzó la aplicación web con la ayuda de la compilación de apache tomcat incrustada en el proyecto. Se realizó el despliegue de la herramienta en un servidor de la Universidad de Montemorelos para su utilización como aplicación web. Este servidor cuenta con un sistema operativo Ubuntu 20.04.1 LTS, 50GB de disco duro y 4GB de memoria RAM. Se puede acceder a la aplicación mediante el siguiente enlace: planea.um.edu.mx.

Mantenimiento

En la etapa de mantenimiento se llegaron a producir varias versiones del producto final por medio del proceso de iteraciones, dando como resultado versiones que cumplían con los requisitos establecidos, pero sujetos a cambios. Con esto en mente se llevaron a cabo cinco reuniones dadas en los meses de noviembre y diciembre de 2020 y febrero, mayo y julio de 2021, una cada mes, con el propósito de recolectar comentarios para llevar a cabo el proceso de mantenimiento.

Las personas involucradas en las reuniones fueron el Lic. Jaime Alcántara, director del departamento de Efectividad Institucional, el Ing. Alejandro Urías, director del Instituto de Mejora Continua de la Facultad de Ingeniería y Tecnología, el Dr. Harvey Alférez, director del Instituto de Ciencia de Datos de la Facultad de Ingeniería y Tecnología, el Ing. Alejandro García, director de la Facultad de Ingeniería y Tecnología y el Dr. Ismael Castillo, rector de la Universidad de Montemorelos.

Los comentarios dados por este grupo de personas dieron como resultado la integración de las secciones de misión, visión, y análisis FODA en una nueva sección llamada marco filosófico. También se modificó el ingreso de la descripción de las actividades, el KPI y el presupuesto del objetivo por el actor usuario. Además, se añadió la validación de los datos ingresados por los usuarios en cada campo para evitar el ingreso de datos incompatibles con el modelo.

CAPÍTULO IV

PRESENTACIÓN Y ANÁLISIS DE RESULTADOS

Introducción

Este capítulo presenta los resultados del desarrollo de la aplicación web para la gestión del plan estratégico de la Universidad de Montemorelos. Específicamente, se tomó a la Facultad de Ingeniería y Tecnología como caso de estudio.

La información obtenida es el resultado del trabajo en conjunto que se realizó con el Ing. Alejandro García en el establecimiento del plan estratégico de la Facultad de Ingeniería y Tecnología para el siguiente quinquenio de operaciones. Para cumplir con este propósito, se llevaron a cabo 10 reuniones entre mayo y octubre del año 2021. Durante este tiempo se trabajó en la definición de la misión, la visión, el análisis FODA, los objetivos y las actividades del plan estratégico de la facultad.

Resultados

Como resultado se logró ingresar en la aplicación web la información de la misión y la visión de la facultad. También se ingresaron los elementos del análisis FODA, conformado por 19 fortalezas, 31 debilidades, 32 oportunidades y nueve amenazas. Se identificaron ocho objetivos para el quinquenio de los cuales se ingresó la información de los dos objetivos a cumplir para el año 2022. Asimismo, se ingresó la información de 16 actividades, ocho asignadas a cada objetivo.

En la Figura 46 se muestra una captura de pantalla de la página de inicio de sesión de la aplicación. Aquí ambos actores, el administrador y el usuario, introducen su correo y contraseña para iniciar sesión. Una vez autenticados, los actores acceden a su cuenta y son recibidos con la página de inicio (ver Figura 47).

Figura 46. Captura de pantalla de la página de inicio de sesión.

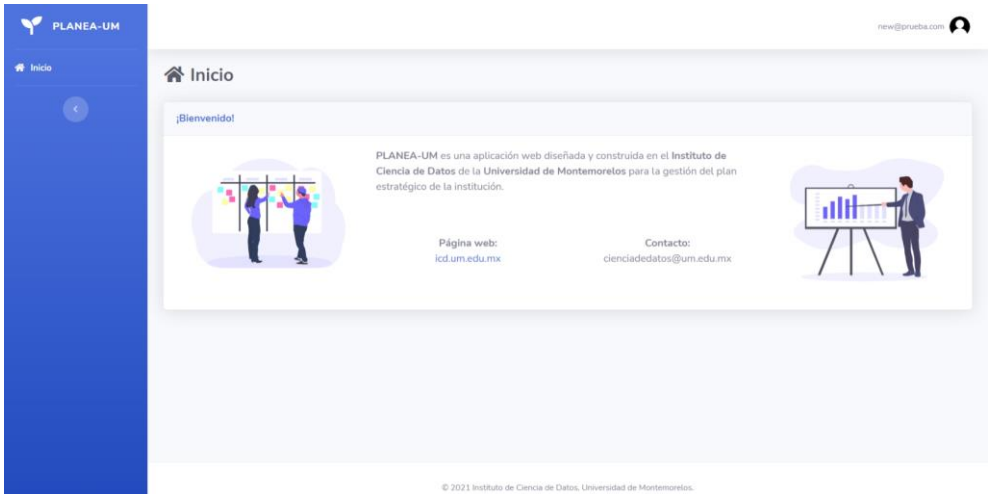

Figura 47. Captura de pantalla de la página de inicio.

Vistas del actor administrador

El ejemplo a continuación es con respecto al actor administrador. La primera sección del menú es la página de departamentos (ver Figura 48). Aquí el actor administrador visualiza los departamentos que conforman su organización. En la Figura 48 se muestra la información de los departamentos que conforman la organización de la Universidad de Montemorelos.

En la Figura 49 se muestra el formulario para agregar un nuevo departamento, el cual está compuesto por tres campos. Los campos requeridos son el nombre del departamento, el departamento arriba y la organización. El nombre del departamento debe ser capturado manualmente por el actor administrador. Los campos departamento arriba y organización son de opción múltiple, dando al actor administrador la capacidad de seleccionar la información requerida.

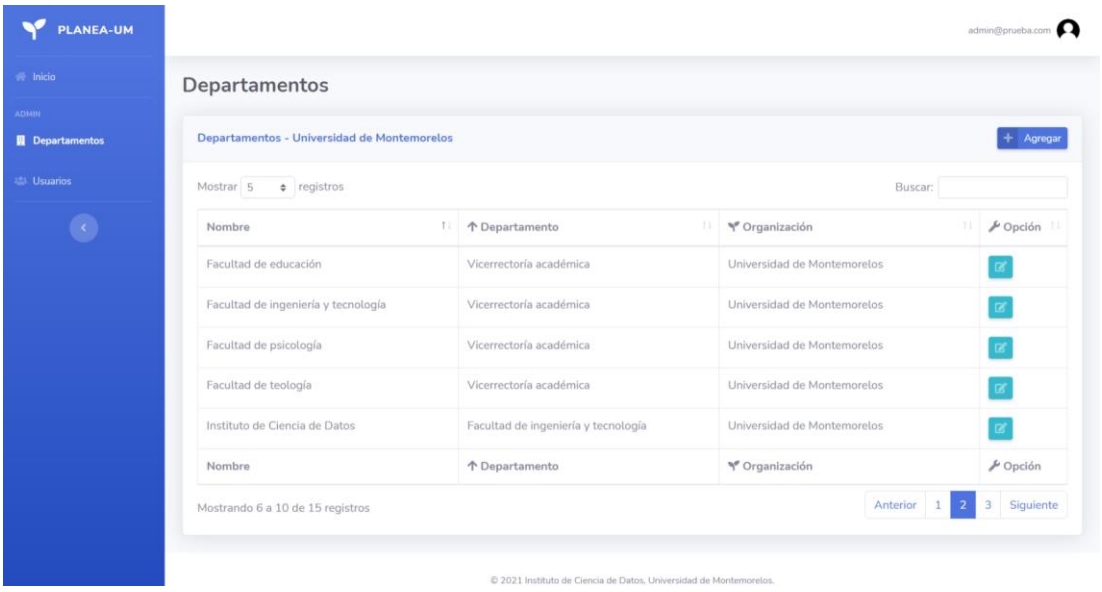

Figura 48. Captura de pantalla de la página de departamentos.

Adicionalmente, el actor administrador puede editar y borrar los departamentos de su organización. En la Figura 50 se muestra la manera en la que se edita un departamento de la Universidad de Montemorelos. Además, en la Figura 51 se muestra la manera de borrar un departamento específico, el botón para borrar solo será visible si el departamento no cuenta con usuarios.

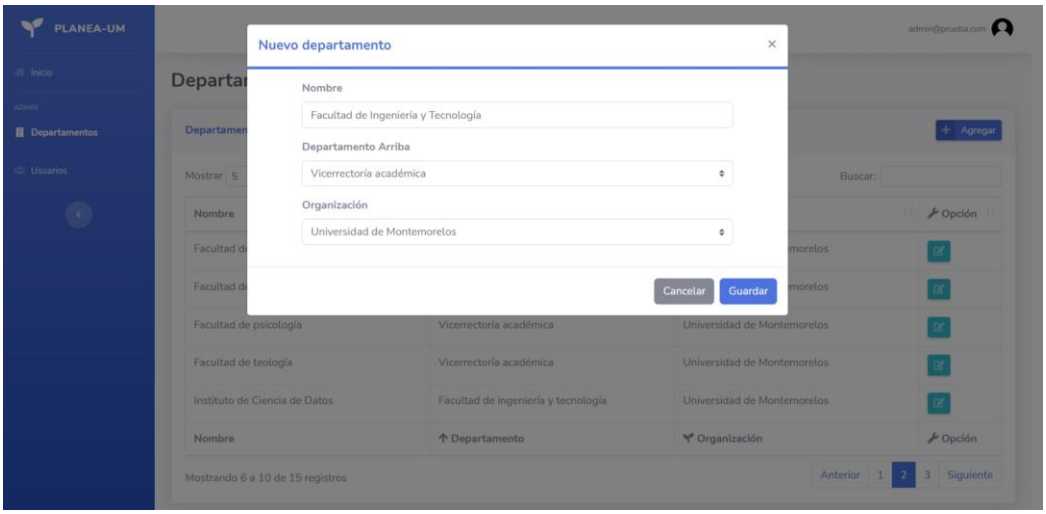

Figura 49. Captura de pantalla de la página de agregar departamentos.

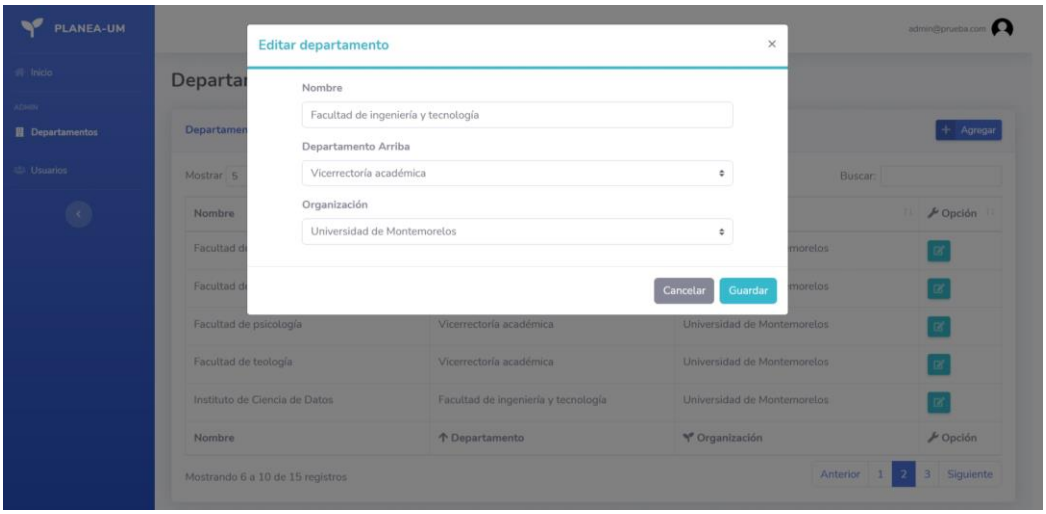

Figura 50. Captura de pantalla de la página de editar departamento.

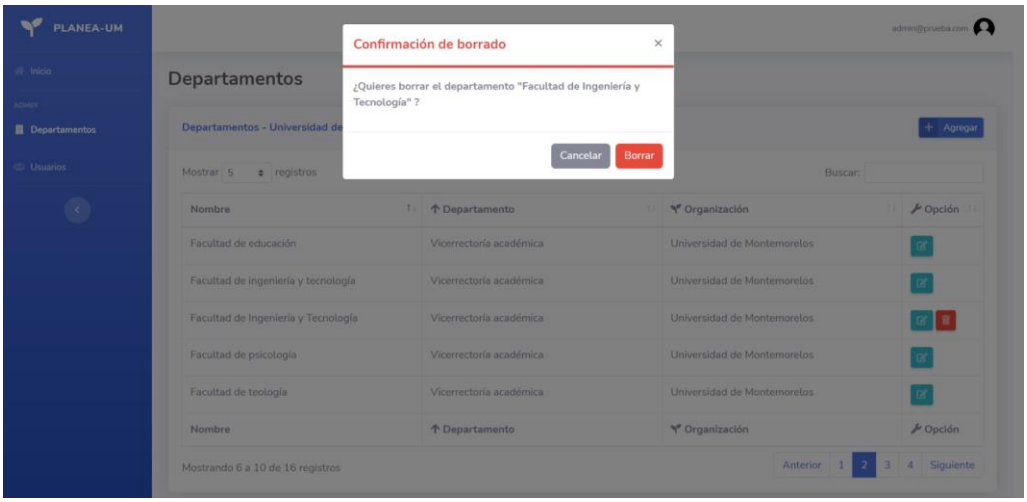

Figura 51. Captura de pantalla de la página de borrar departamento.

La segunda sección del menú es la página de usuarios (ver Figura 52). Aquí el actor administrador visualiza los usuarios que conforman su organización. En la Figura 52 se muestra la información de los usuarios que conforman la organización de la Universidad de Montemorelos.

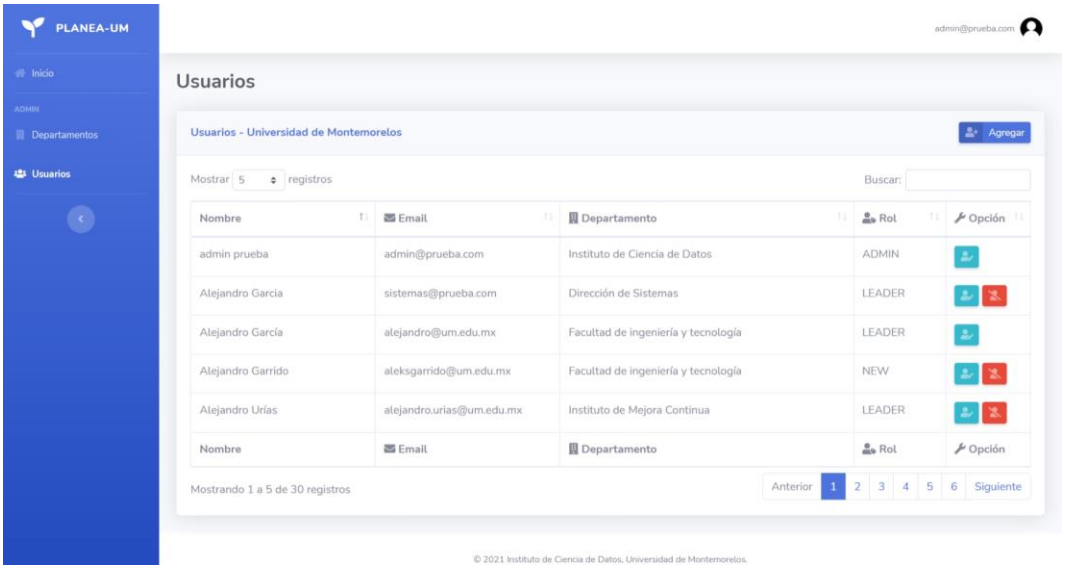

Figura 52. Captura de pantalla de la página de usuarios.

En la Figura 53 se muestra el formulario para agregar un nuevo usuario, el cual está compuesto por seis campos. Los campos requeridos son el nombre y apellido del usuario, el correo electrónico, la contraseña, el departamento al que pertenece y el rol que tendrá. El nombre y apellido del usuario, el correo electrónico y la contraseña deben ser capturados manualmente por el actor administrador. Los campos departamento y rol son de opción múltiple, dando al actor administrador la capacidad de seleccionar la información requerida.

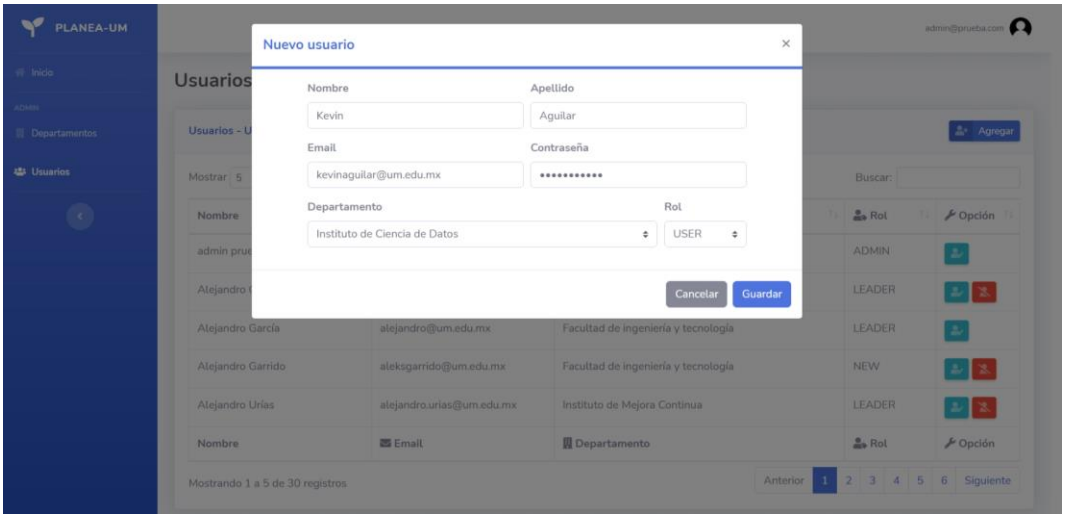

Figura 53. Captura de pantalla de la página de agregar usuarios.

Adicionalmente, el actor administrador puede editar y borrar los usuarios de su organización. En la Figura 54 se muestra la manera en la que se edita un usuario de la Universidad de Montemorelos. Además, en la Figura 55 se muestra la manera de borrar un usuario específico, el botón para borrar solo será visible si el usuario no cuenta con objetivos.

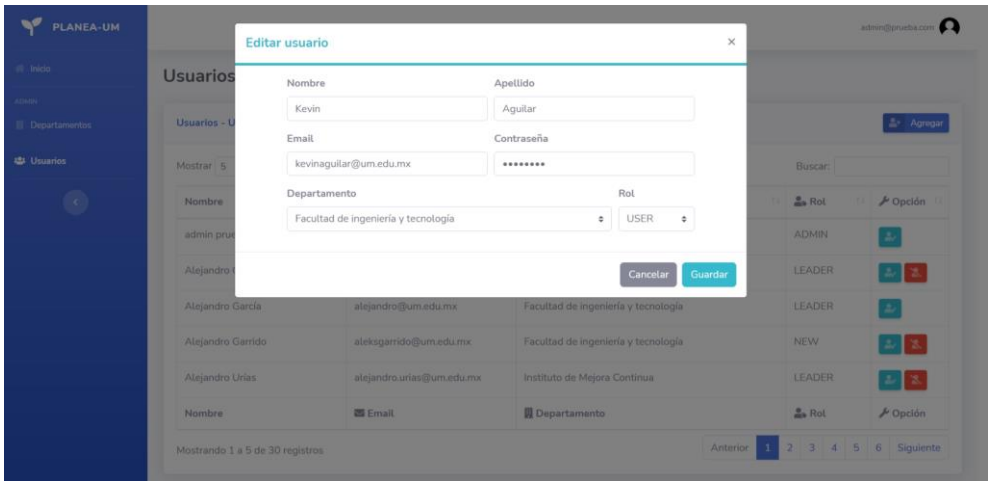

Figura 54. Captura de pantalla de la página de editar usuario.

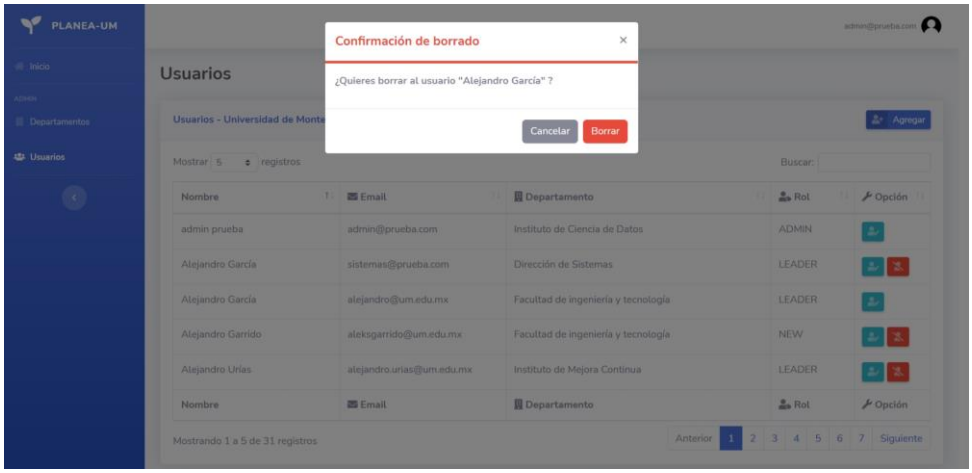

Figura 55. Captura de pantalla de la página de borrar usuario.

Vistas del actor usuario

El ejemplo a continuación es con respecto al actor usuario. La primera sección del menú es la página de equipo (ver Figura 56). Aquí el actor usuario visualiza a los miembros del equipo que pertenecen al mismo departamento. En la Figura 56 se muestra la información de las personas que conforman el equipo de la Facultad de Ingeniería y Tecnología.

La siguiente sección corresponde al marco filosófico del departamento. Aquí se encuentra la información de misión, visión y análisis FODA. El actor usuario puede modificar la misión (ver Figura 57) y visión (ver Figura 58) del departamento.

Además de añadir, editar y borrar los elementos que conforman el análisis FODA, siendo estos las fortalezas, debilidades, amenazas y oportunidades del departamento (ver Figura 59). En la Figura 57 se muestra la declaración de misión de la Facultad de Ingeniería y Tecnología. Igualmente, en la Figura 58 se muestra la declaración de visión capturada en la aplicación web.

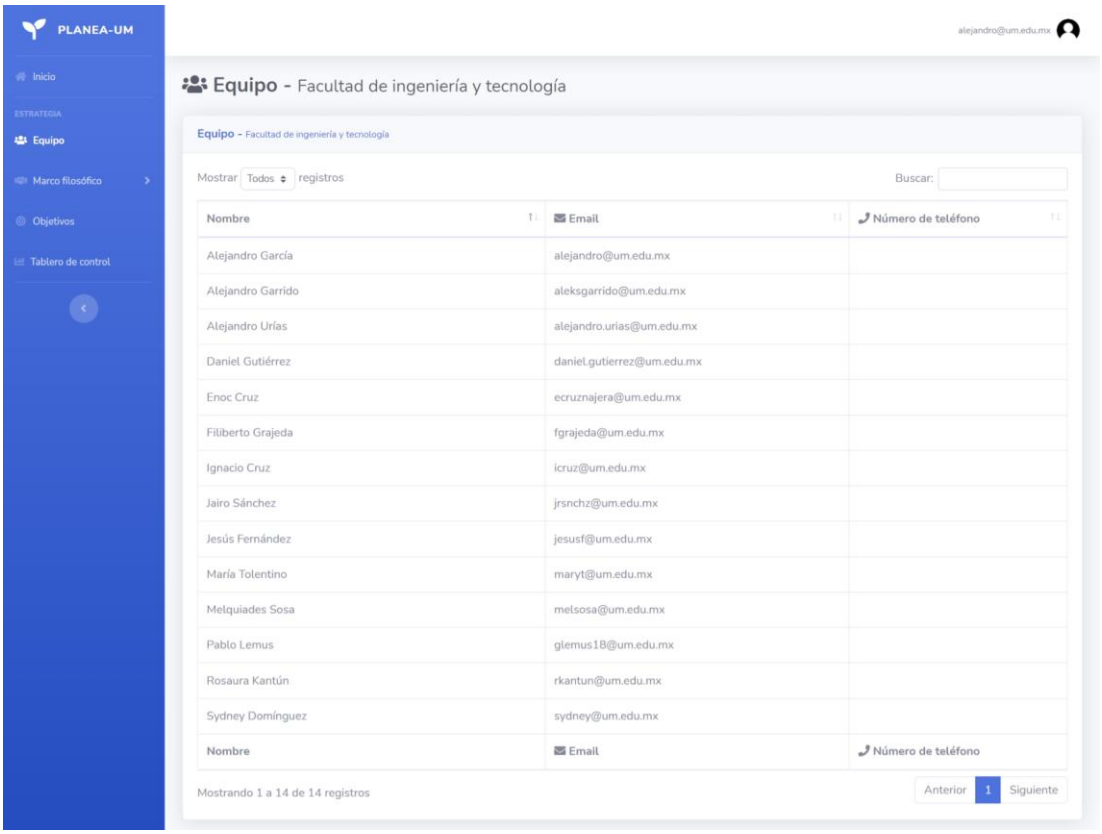

Figura 56. Captura de pantalla de la página de equipo.

Figura 57. Captura de pantalla de la página de misión.

Figura 58. Captura de pantalla de la página de visión.

En la Figura 59 se muestran los elementos que conforman el análisis FODA de la Facultad de Ingeniería y Tecnología capturados en la aplicación web. Estos se dividen en factores internos (fortalezas y debilidades) y factores externos (oportunidades y amenazas).

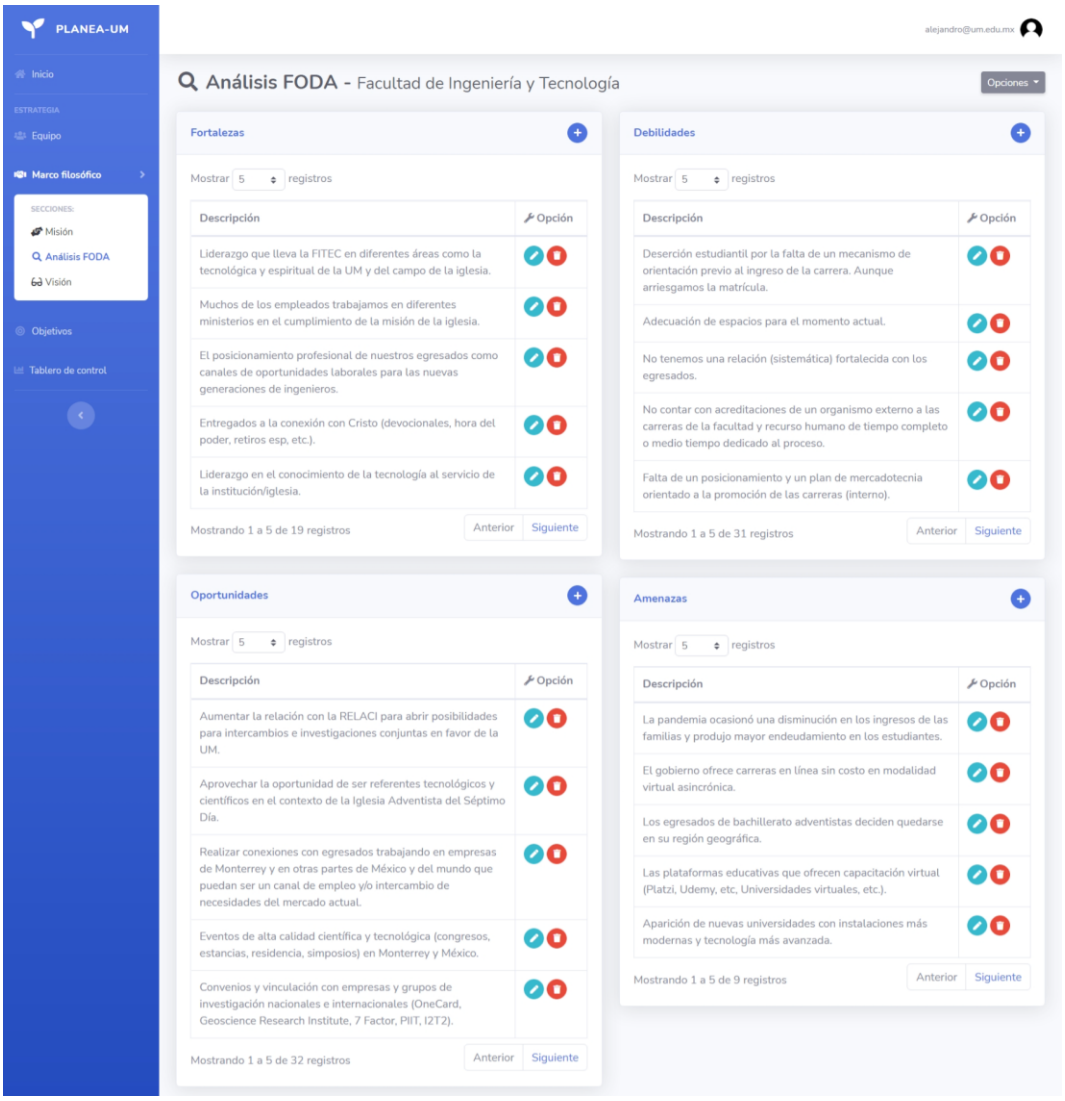

Figura 59. Captura de pantalla de la página de análisis FODA.

La siguiente sección es la de objetivos donde se muestran los objetivos del departamento. En la Figura 60 se muestra el listado de objetivos de la Facultad de Ingeniería y Tecnología. La Figura 61 muestra el formulario para agregar un nuevo objetivo, el cual está compuesto por 12 campos. La mayoría de los campos son de opción múltiple, estos son la acción, la métrica, la fecha de inicio y de finalización, la meta más amplia, el proyecto de iglesia, el tipo de fondo y el impacto.

Estos campos dan al actor usuario la capacidad de seleccionar la información requerida. Por otro lado, se encuentran los campos donde la información debe ser capturada manualmente por el actor usuario, estos son la descripción, la cantidad, el propósito y el presupuesto.

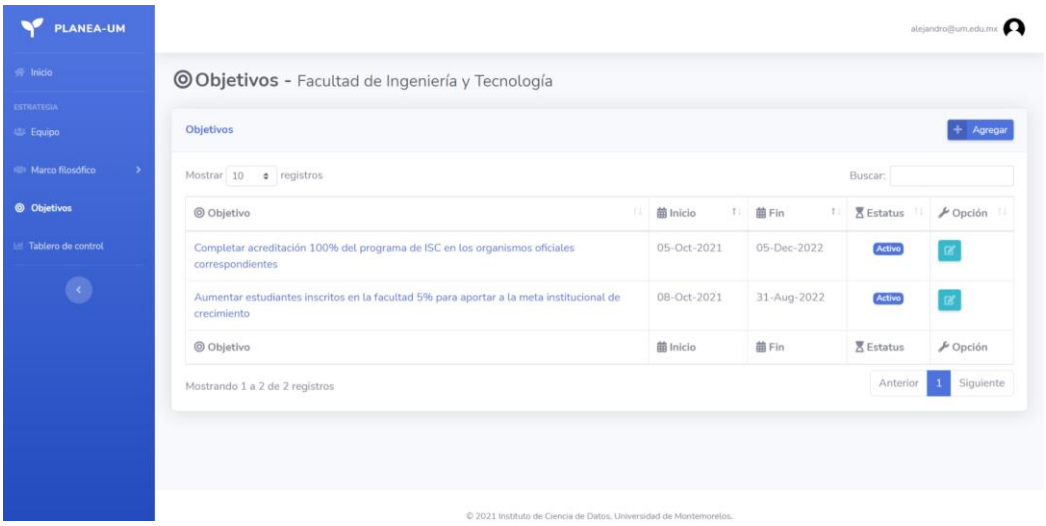

Figura 60. Captura de pantalla de la página de objetivos.

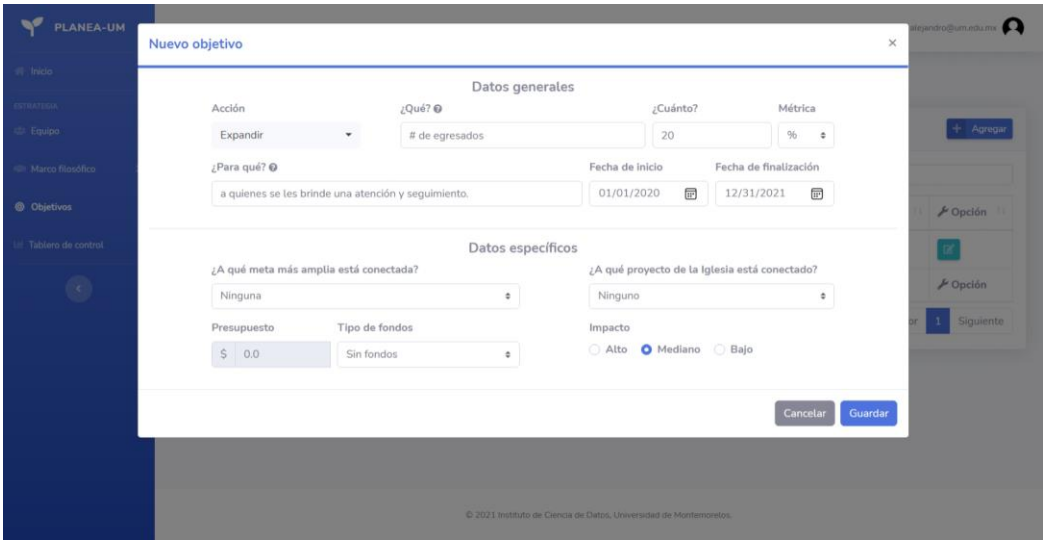

Figura 61. Captura de pantalla de la página de agregar objetivos.

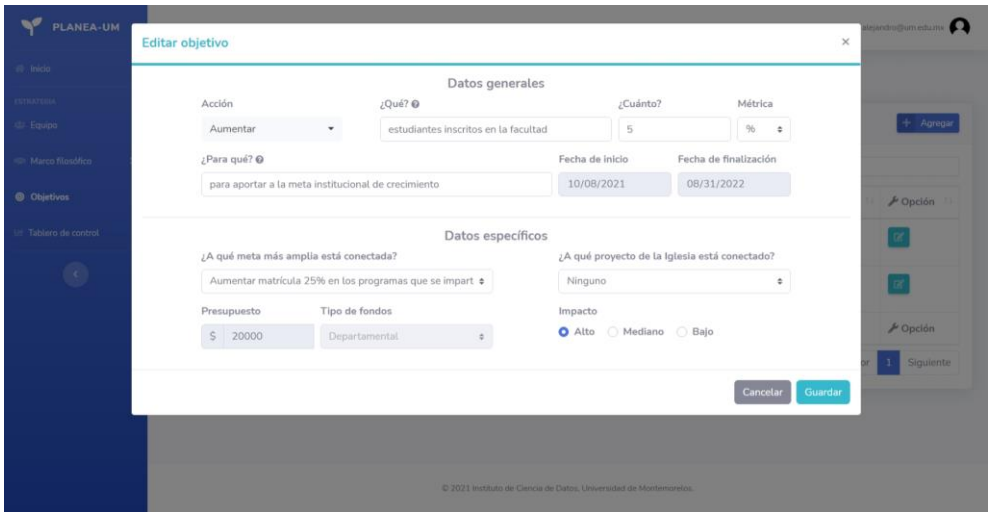

Figura 62. Captura de pantalla de la página de editar objetivo.

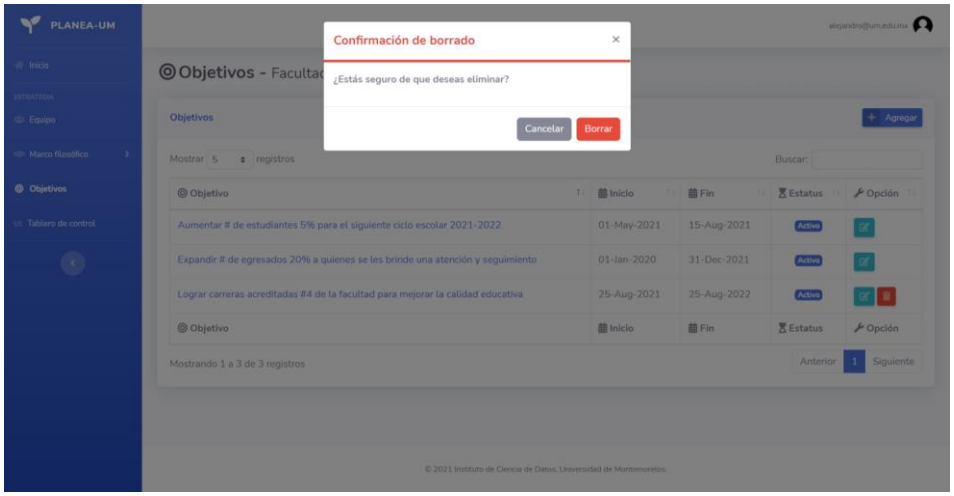

Figura 63. Captura de pantalla de la página de borrar objetivo.

Adicionalmente, el actor usuario puede editar y borrar los objetivos de su departamento. En la Figura 62 se muestra la manera en la que se edita un objetivo de la Facultad de Ingeniería y Tecnología. Además, en la Figura 63 se muestra la manera de borrar un objetivo específico, el botón para borrar solo será visible si el objetivo no cuenta con actividades.

Al momento de crear un nuevo objetivo, se muestra la página de información detallada que al mismo tiempo refiere al actor usuario a crear las actividades específicas a realizar para el cumplimiento del objetivo en cuestión. En la Figura 64 se muestran los detalles de un objetivo de la Facultad de Ingeniería y Tecnología.

Los detalles de un objetivo se dividen en tres áreas: el nombre del objetivo, la información del objetivo y las actividades. En la información del objetivo se visualiza la meta más amplia, el proyecto de iglesia, la fecha de inicio y de finalización, el presupuesto, el impacto, el KPI y el progreso del KPI.

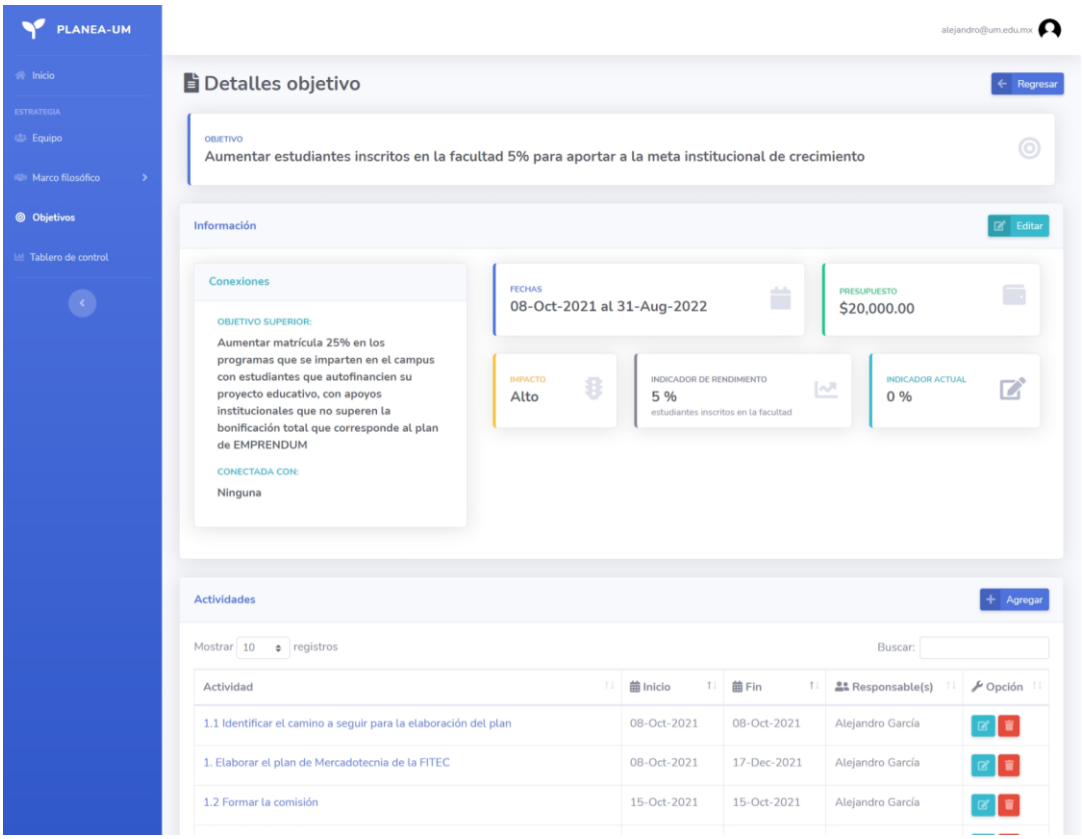

Figura 64. Captura de pantalla de la página de detalles de un objetivo.

La Figura 65 muestra el formulario para agregar una nueva actividad, el cual está compuesto por seis campos. La mayoría de los campos son de opción múltiple, estos son la fecha de inicio y de finalización, la(s) persona(s) responsable(s) y el impacto, dando al actor usuario la capacidad de seleccionar la información requerida. Por otro lado, se encuentran los campos donde la información debe ser capturada manualmente por el actor usuario, estos son la descripción de la actividad y el presupuesto.

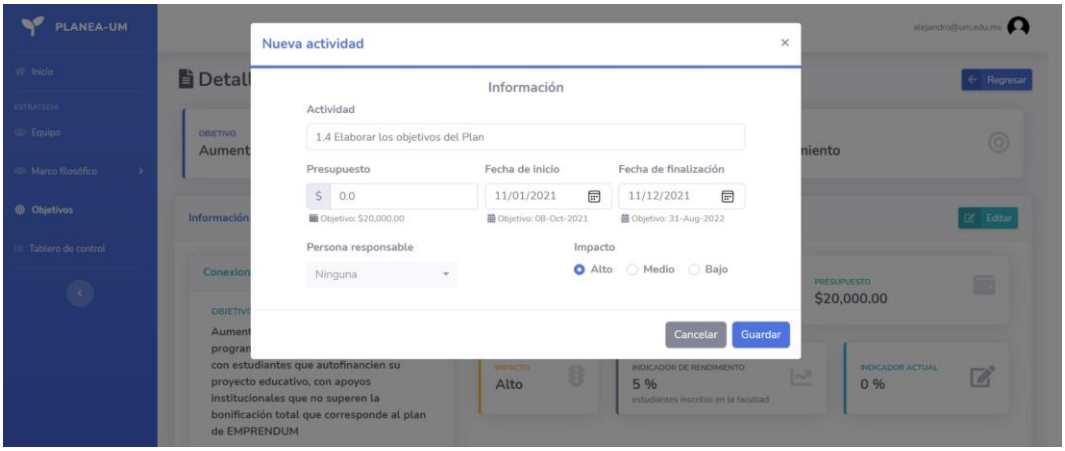

Figura 65. Captura de pantalla de la página de agregar actividades.

Adicionalmente, el actor usuario puede editar y borrar las actividades de un objetivo específico. En la Figura 66 se muestra la manera en la que se edita una actividad de la Facultad de Ingeniería y Tecnología. Además, en la Figura 67 se muestra la manera de borrar una actividad específica, el botón para borrar solo será visible si la actividad no cuenta con elementos del presupuesto. Una vez que el actor usuario ha creado las actividades, puede acceder a la información detallada de una actividad para actualizar el progreso de esta o añadir elementos al presupuesto detallado.

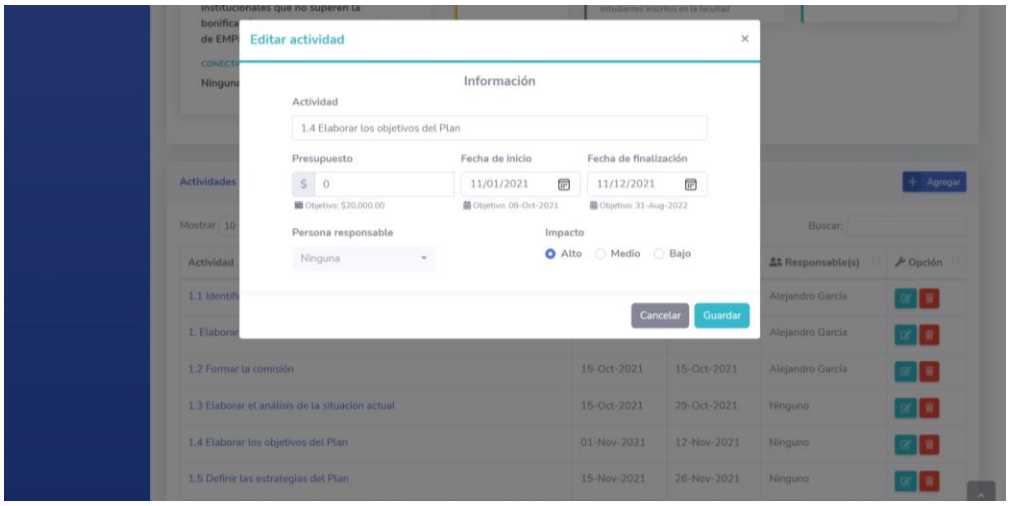

Figura 66. Captura de pantalla de la página de editar actividad.

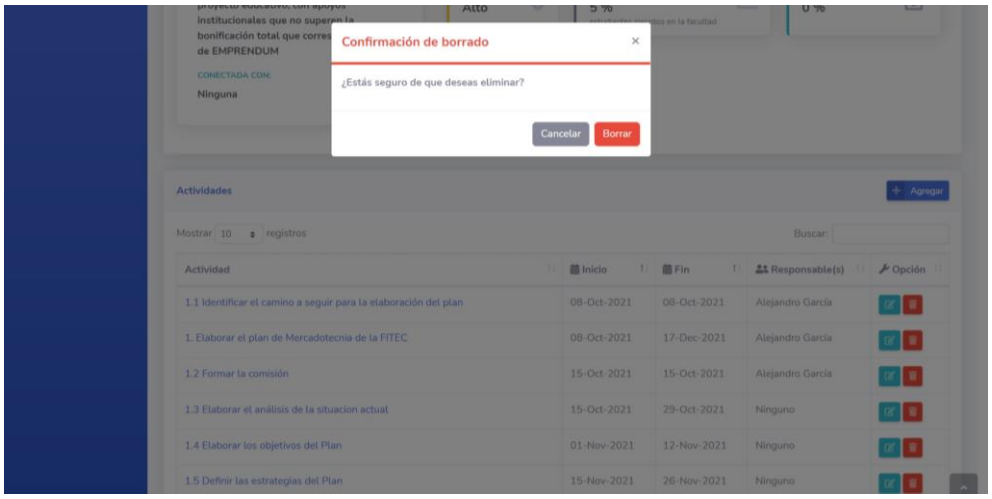

Figura 67. Captura de pantalla de la página de borrar actividad.

En la Figura 68 se muestran los detalles de una actividad de la Facultad de Ingeniería y Tecnología. Los detalles de una actividad se dividen en tres áreas: el nombre de la actividad, la información de la actividad y los elementos del presupuesto detallado. En la información de la actividad se visualiza la fecha de inicio y finalización, el presupuesto, la(s) persona(s) responsable(s), el impacto y el progreso. La Figura 69

muestra el formulario para agregar un nuevo elemento al presupuesto detallado de una actividad, el cual está compuesto por dos campos, la descripción del elemento y el presupuesto a asignar.

Adicionalmente, el actor usuario puede editar y borrar los elementos del presupuesto detallado de una actividad especifica. En la Figura 70 se muestra la manera en la que se edita un elemento del presupuesto de una actividad de la Facultad de Ingeniería y Tecnología. Además, en la Figura 71 se muestra la manera de borrar un elemento del presupuesto específico. Asimismo, el actor usuario puede actualizar el progreso de una actividad especifica. El color de la barra de progreso le indica al actor usuario el estado de cumplimiento de la actividad. La Figura 72 muestra la actualización del progreso de una actividad de la Facultad de Ingeniería y Tecnología.

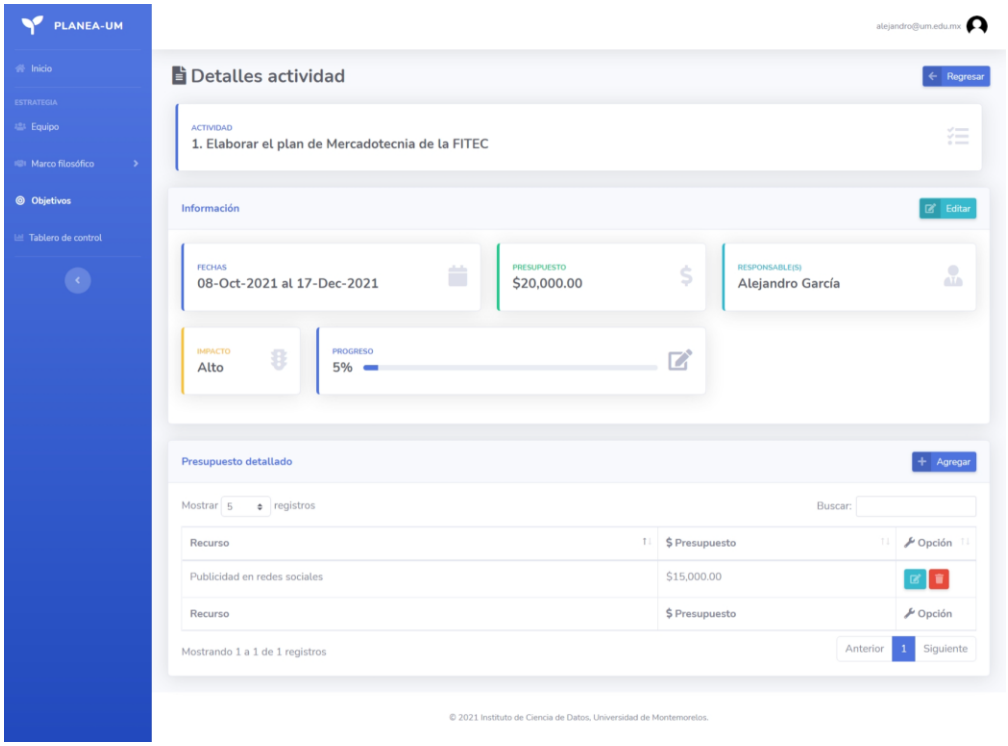

Figura 68. Captura de pantalla de la página de detalles de una actividad.

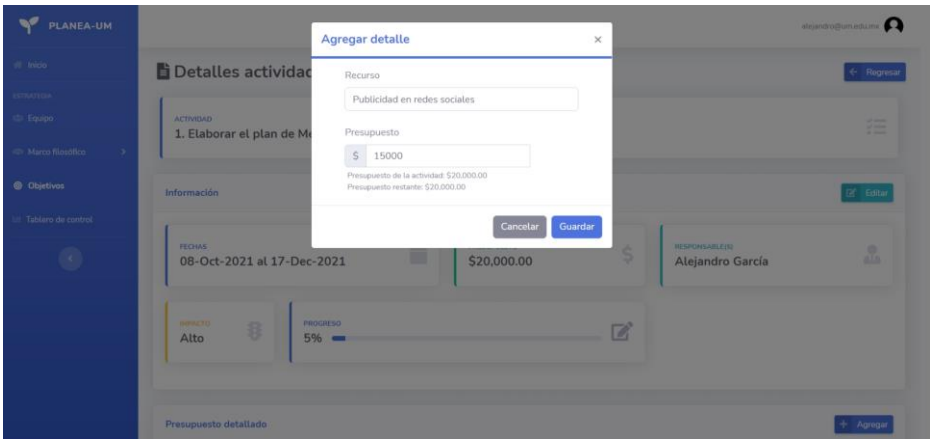

Figura 69. Captura de pantalla de la página de agregar elemento del presupuesto.

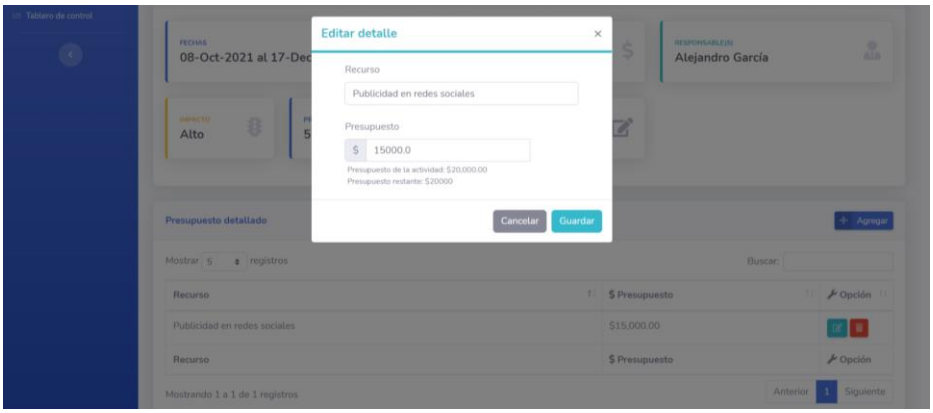

Figura 70. Captura de pantalla de la página de editar elemento del presupuesto.

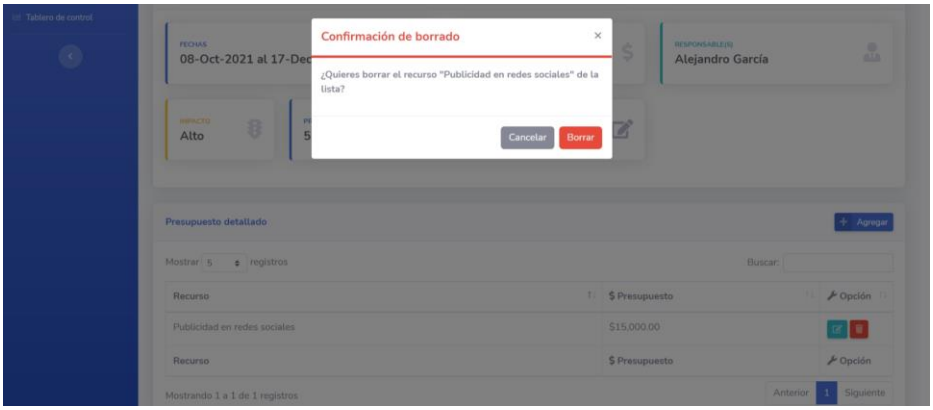

Figura 71. Captura de pantalla de la página de borrar elemento del presupuesto.

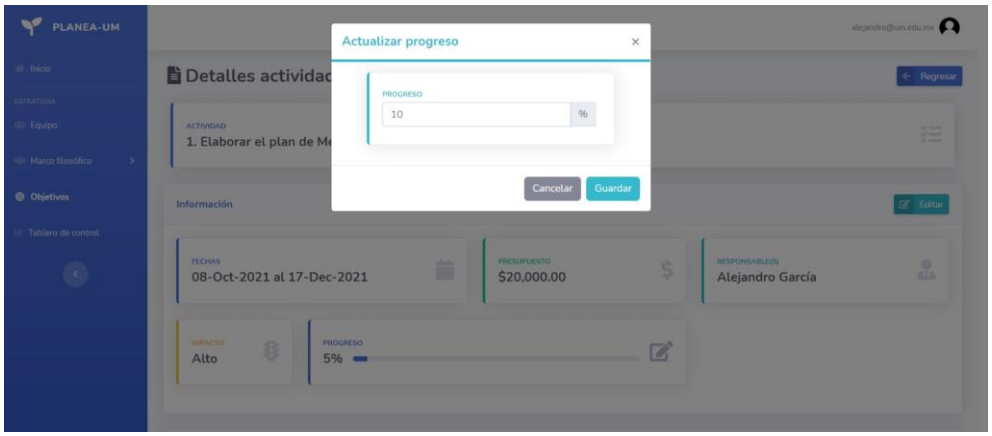

Figura 72. Captura de pantalla de la página de actualizar progreso de una actividad.

La última sección corresponde al tablero de control del departamento. La Figura 73 muestra la visualización del progreso de los objetivos del departamento en un tablero de control. En esta última sección de la aplicación se presenta la información del progreso de los objetivos y de las actividades. El tablero de control muestra al equipo el avance en los objetivos y las actividades. El progreso debe ser revisado una vez a la semana. Este proceso se pretende incorporar a la agenda de actividades de las sesiones de consejo técnico de la Facultad de Ingeniería y Tecnología.

La información visualizada en el tablero de control muestra al actor usuario el porcentaje del progreso del objetivo basado en el avance del KPI. También, muestra el porcentaje del progreso de las actividades basado en el promedio del avance del grupo de actividades del objetivo. Los colores en las barras de progreso muestran al actor usuario si el porcentaje de avance está al corriente con respecto a la fecha establecida de finalización para que los objetivos y/o las actividades se cumplan.

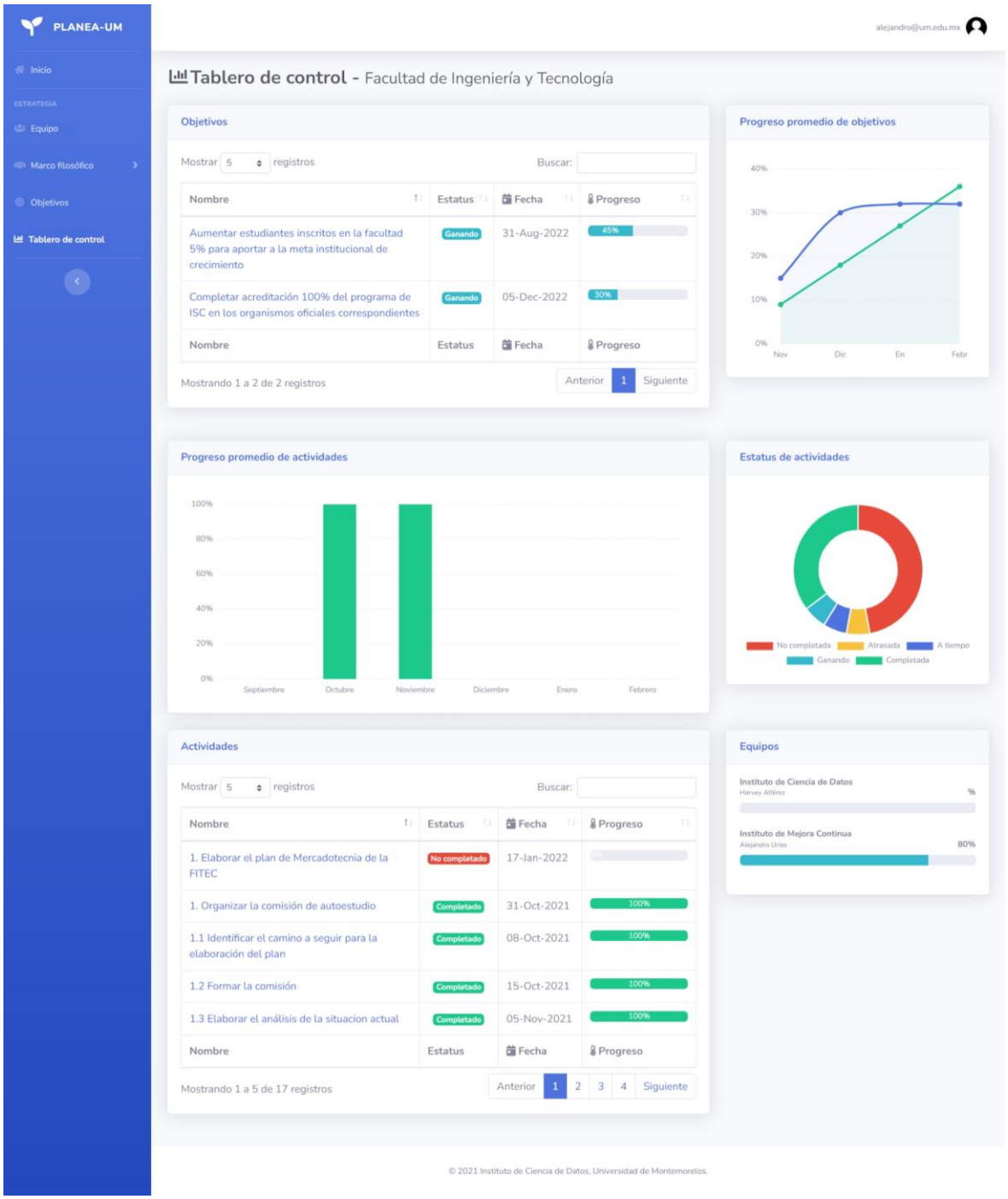

Figura 73. Captura de pantalla de la página del tablero de control.

CAPÍTULO V

DISCUSIÓN, CONCLUSIONES Y RECOMENDACIONES

Discusión

En relación con los resultados de la investigación, se puede señalar que la hipótesis se cumple. La gestión del plan estratégico mediante una aplicación web tiene un efecto positivo en su cumplimiento en la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos. Esto es así, siempre y cuando se tomen en cuenta las siguientes indicaciones: (a) que las personas que utilicen la aplicación web se comprometan a realizar el seguimiento de sus objetivos y actividades en reuniones de rendición de cuentas y (b) que la información del progreso de los objetivos y las actividades en el tablero de control esté disponible.

Durante el tiempo dedicado al desarrollo del plan estratégico de la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos, el Ing. Alejandro García, director de esta facultad, dijo lo siguiente con respecto a PlaneaUM:

> La herramienta sin duda alguna es un gran aporte para el futuro de la Universidad de Montemorelos. No ha existido anteriormente un lugar en donde se pueda gestionar el plan estratégico de la facultad de tal manera que nuestros objetivos se alineen con los de la institución, además de que podamos visualizar el progreso de los objetivos conforme pasa el tiempo. Ciertamente PlaneaUM será de gran ayuda para la facultad a medida en que aprendamos a sacarle el mayor provecho y nos comprometamos a alimentar el sistema constantemente. Por otro lado, el diseño de la aplicación es bastante agradable, lo que la hace fácil de utilizar. Además, la herramienta hizo posible la formalización de ciertos objetivos, que anteriormente estábamos trabajando, y que al no formalizarlos no sabíamos acerca de su avance.

Asimismo, el Dr. Ismael castillo, rector de la Universidad de Montemorelos, utilizó la funcionalidad del marco filosófico de la aplicación web y comentó lo siguiente:

> Felicidades por el desarrollo de PlaneaUM, mi gratitud y reconocimiento al equipo del Instituto de Ciencia de Datos por este gran aporte a la administración de la Universidad de Montemorelos. Vamos a utilizar esta herramienta para gestionar el plan estratégico de la institución.

Finalmente, PlaneaUM ofrece la flexibilidad de modificar y actualizar la información del plan estratégico, por ejemplo al establecer la fecha de finalización para los objetivos y las actividades, al actualizar los KPIs y al facilitar el seguimiento del plan por medio de un tablero de control. Ciertamente, la definición del plan estratégico mediante la utilización de PlaneaUM sirve para formalizar y analizar los objetivos y las actividades a cumplir.

Conclusiones

La principal contribución de esta investigación es la creación de una aplicación web para la gestión del plan estratégico de la Universidad de Montemorelos. Por medio de la aplicación web es posible realizar la gestión del plan estratégico de la Facultad de Ingeniería y Tecnología de la Universidad de Montemorelos con base en la declaración de misión y visión, el análisis FODA, la declaración de objetivos y actividades y el seguimiento por medio de un tablero de control.

Con el fin de construir este software, se trabajó en la adaptación de los conceptos de planeación estratégica de los libros de *Becoming a mission-driven church* (Brantley et al., 2015) y *The 4 disciplines of execution* (McChesney et al., 2012).

Durante la construcción de la aplicación web se siguió una metodología utilizando un proceso iterativo incremental con el fin de cumplir con los requisitos

funcionales para el cumplimiento de los casos de uso de los actores involucrados en el software. Además, se utilizó spring framework para el desarrollo de la herramienta.

Recomendaciones

Como trabajo futuro se espera realizar un desarrollo extendido de la aplicación web considerando lo siguiente:

1. Se espera que el resultado de esta investigación pueda ser utilizado por un mayor número de personas pertenecientes a la Universidad de Montemorelos. El Instituto de Ciencia de Datos, por medio de su boletín semanal, compartió información de la aplicación web a los directivos de la institución los días 19 y 26 de octubre. Como resultado de esta primera divulgación de la herramienta, 10 directivos solicitaron acceso.

2. Se espera recibir la retroalimentación de los usuarios que tengan acceso a la aplicación web para la construcción de futuras versiones de la herramienta. Esto con la finalidad de asegurar el funcionamiento correcto del software y de expandir las funcionalidades de la aplicación web.

3. Se espera que la herramienta sea utilizada por otras instituciones de la Iglesia Adventista del Séptimo Día para gestionar su plan estratégico. Esto con el fin de que el resultado de esta investigación sea de beneficio no sólo para la Universidad de Montemorelos sino también para otras instituciones.

APÉNDICE A

DOCUMENTACIÓN DE CASOS DE USO

Actores

Nombre del actor: Administrador.
Descripción: Este actor representa al administrador de la organización. Es la persona que se encarga de gestionar los departamentos y los usuarios de la institución dentro de la aplicación web.

Nombre del actor: Usuario.
Descripción: Este actor representa al usuario líder de un departamento. Es la persona que se encarga de gestionar el plan estratégico de su departamento dentro de la institución.

Requisitos funcionales

Nombre del caso de uso: Crear departamento.
Resumen: El Administrador debe ingresar la información del departamento: nombre del departamento, departamento arriba y organización a la que pertenezca.
Actores: Administrador.
Precondición: Ninguna.
Descripción: - El Administrador elige la opción de agregar nuevo departamento. - La aplicación muestra el formulario de creación de departamento. - El Administrador ingresa los datos del departamento: nombre del departamento, departamento arriba y organización a la que pertenezca.

- La aplicación genera el nuevo departamento, lo registra en la base de datos y muestra al Administrador el listado de departamentos actualizado.
Alternativas: Ninguna.
Postcondición: El departamento se ha guardado.

Nombre del caso de uso: Actualizar información del departamento.
Resumen: El Administrador actualiza la información de un departamento específico.
Actores: Administrador.
Precondición: Ninguna.
Descripción: - El Administrador elige la opción de editar departamento. - El Administrador actualiza el nombre del departamento y / o el departamento arriba. - La aplicación registra los cambios hechos al departamento en la base de datos.
Alternativas: Ninguna.
Postcondición: El departamento se ha actualizado.

Nombre del caso de uso: Borrar departamento.
Resumen: El Administrador borra un departamento específico.
Actores: Administrador.
Precondición: El departamento no debe tener usuarios.
Descripción: - El Administrador elige la opción de borrar departamento.

- El Administrador confirma la acción de borrar el departamento. - La aplicación registra la eliminación del departamento en la base de datos.
Alternativas: Ninguna.
Postcondición: El departamento se ha borrado.

Nombre del caso de uso: Crear usuario.
Resumen: El Administrador debe ingresar la información del usuario: nombre y apellido, correo electrónico, contraseña, departamento al que pertenezca y rol.
Actores: Administrador.
Precondición: No debe existir un usuario con el mismo correo electrónico.
Descripción: - El Administrador elige la opción de agregar nuevo usuario. - La aplicación muestra el formulario de creación de usuario. - El Administrador ingresa los datos del usuario: nombre y apellido, correo electrónico, contraseña, departamento al que pertenezca y rol. - La aplicación genera el nuevo usuario, lo registra en la base de datos y muestra al Administrador el listado de usuarios actualizado.
Alternativas: Ninguna.
Postcondición: El usuario se ha guardado.

Nombre del caso de uso: Actualizar información del usuario.
Resumen: El Administrador actualiza la información de un usuario específico.
Actores: Administrador.

Precondición: No debe existir un usuario con el mismo correo electrónico.
Descripción: - El Administrador elige la opción de editar usuario. - El Administrador actualiza la información del usuario: nombre y apellido, correo electrónico, contraseña, departamento al que pertenezca y rol. - La aplicación registra los cambios hechos al usuario en la base de datos.
Alternativas: Ninguna.
Postcondición: El usuario se ha actualizado.

Nombre del caso de uso: Borrar usuario.
Resumen: El Administrador borra un usuario específico.
Actores: Administrador.
Precondición: El usuario no debe tener objetivos.
Descripción: - El Administrador elige la opción de borrar usuario. - El Administrador confirma la acción de borrar el usuario. - La aplicación registra la eliminación del usuario en la base de datos.
Alternativas: Ninguna.
Postcondición: El usuario se ha borrado.

Nombre del caso de uso: Actualizar misión del departamento.
Resumen: El Usuario actualiza la misión de su departamento.
Actores: Usuario.

Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar misión. - El Usuario actualiza la misión de su departamento. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: La misión se ha actualizado.

Nombre del caso de uso: Actualizar visión del departamento.
Resumen: El Usuario actualiza la visión de su departamento.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar visión. - El Usuario actualiza la visión de su departamento. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: La visión se ha actualizado.

Nombre del caso de uso: Agregar FODA del departamento.
Resumen: El Usuario debe ingresar la descripción de la fortaleza, debilidad, oportunidad o amenaza.
Actores: Usuario.

Precondición: Ninguna.
Descripción: - El Usuario elige la opción de agregar nueva fortaleza, debilidad, oportunidad o amenaza. - La aplicación muestra el formulario de creación de la fortaleza, debilidad, oportunidad o amenaza. - El Usuario ingresa la descripción de la fortaleza, debilidad, oportunidad o amenaza. - La aplicación genera la nueva fortaleza, debilidad, oportunidad o amenaza, la registra en la base de datos y muestra al Usuario el listado del análisis FODA actualizado.
Alternativas: Ninguna.
Postcondición: La fortaleza, debilidad, oportunidad o amenaza se ha guardado.

Nombre del caso de uso: Actualizar FODA del departamento.
Resumen: El usuario actualiza la descripción de una fortaleza, debilidad, oportunidad o amenaza específica.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar fortaleza, debilidad, oportunidad o amenaza. - El Usuario actualiza la descripción de la fortaleza, debilidad, oportunidad o amenaza. - La aplicación registra los cambios hechos en la base de datos.

Alternativas: Ninguna.
Postcondición: La fortaleza, debilidad, oportunidad o amenaza se ha actualizado.

Nombre del caso de uso: Borrar FODA del departamento
Resumen: El Usuario borra una fortaleza, debilidad, oportunidad o amenaza específica.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de borrar fortaleza, debilidad, oportunidad o amenaza. - El Usuario confirma la acción de borrar la fortaleza, debilidad, oportunidad o amenaza. - La aplicación registra la eliminación de la fortaleza, debilidad, oportunidad o amenaza en la base de datos.
Alternativas: Ninguna.
Postcondición: La fortaleza, debilidad, oportunidad o amenaza se ha borrado.

Nombre del caso de uso: Crear objetivo.
Resumen: El Usuario debe ingresar la información del objetivo: acción, descripción, cantidad, métrica, propósito, fecha de inicio, fecha de finalización, meta más amplia, proyecto de la iglesia, presupuesto, tipo de fondo e impacto.
Actores: Usuario.
Precondición: Ninguna.

Descripción: - El Usuario elige la opción de agregar nuevo objetivo. - La aplicación muestra el formulario de creación de objetivo. - El Usuario ingresa los datos del objetivo: acción, descripción, cantidad, métrica, propósito, fecha de inicio, fecha de finalización, meta más amplia, proyecto de la iglesia, presupuesto, tipo de fondo e impacto. - La aplicación genera el nuevo objetivo, lo registra en la base de datos y muestra al Usuario la pantalla de detalles del objetivo.
Alternativas: Ninguna.
Postcondición: El objetivo se ha guardado.

Nombre del caso de uso: Actualizar información del objetivo.
Resumen: El Usuario actualiza la información de un objetivo específico.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar objetivo. - El Usuario actualiza la acción, descripción, cantidad, métrica, propósito, fecha de inicio, fecha de finalización, meta más amplia, proyecto de la iglesia, presupuesto, tipo de fondo o impacto. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: El objetivo se ha actualizado.

Nombre del caso de uso: Borrar objetivo.
Resumen: El Usuario borra un objetivo específico.
Actores: Usuario.
Precondición: El objetivo no debe tener actividades.
Descripción: - El Usuario elige la opción de borrar objetivo. - El Usuario confirma la acción de borrar objetivo. - La aplicación registra la eliminación del objetivo en la base de datos.
Alternativas: Ninguna.
Postcondición: El objetivo se ha borrado.

Nombre del caso de uso: Actualizar KPI del objetivo.
Resumen: El Usuario actualiza la información del KPI de un objetivo específico.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar KPI. - El Usuario actualiza el KPI del objetivo. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: El KPI del objetivo se ha actualizado.

Nombre del caso de uso: Crear actividad.
Resumen: El Usuario debe ingresar la información de la actividad: descripción, presupuesto, fecha de inicio, fecha de finalización, persona(s) responsable(s) e impacto.
Actores: Usuario.
Precondición: Las fechas de inicio y finalización deben estar dentro de las del objetivo.
Descripción: - El Usuario elige la opción de agregar nueva actividad. - La aplicación muestra el formulario de creación de actividad. - El Usuario ingresa los datos de la actividad: descripción, presupuesto, fecha de inicio, fecha de finalización, persona(s) responsable(s) e impacto. - La aplicación genera la nueva actividad, la registra en la base de datos y muestra al Usuario la pantalla de detalles del objetivo con el listado de actividades actualizado.
Alternativas: Ninguna.
Postcondición: La actividad se ha guardado.

Nombre del caso de uso: Actualizar información de la actividad.
Resumen: El Usuario actualiza la información de una actividad específica.
Actores: Usuario.
Precondición: Las fechas de inicio y finalización deben estar dentro de las del objetivo. Si la actividad tiene elementos del presupuesto no es posible actualizar el presupuesto.

Descripción: - El Usuario elige la opción de editar objetivo. - El Usuario actualiza la acción, descripción, cantidad, métrica, propósito, fecha de inicio, fecha de finalización, meta más amplia, proyecto de la iglesia, presupuesto, tipo de fondo o impacto. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: La actividad se ha actualizado.

Nombre del caso de uso: Borrar actividad.
Resumen: El Usuario borra una actividad específica.
Actores: Usuario.
Precondición: La actividad no debe tener elementos del presupuesto.
Descripción: - El Usuario elige la opción de borrar actividad. - El Usuario confirma la acción de borrar actividad. - La aplicación registra la eliminación de la actividad en la base de datos.
Alternativas: Ninguna.
Postcondición: La actividad se ha borrado.

Nombre del caso de uso: Actualizar progreso de la actividad.
Resumen: El Usuario actualiza el progreso de una actividad específica.
Actores: Usuario.

Precondición: Ninguna.
Descripción: - El Usuario elige la opción de editar progreso de la actividad. - El Usuario actualiza el progreso de la actividad. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: El progreso de la actividad se ha actualizado.

Nombre del caso de uso: Crear elemento del presupuesto de la actividad.
Resumen: El Usuario debe ingresar la información del elemento del presupuesto de la actividad: descripción y presupuesto.
Actores: Usuario.
Precondición: El presupuesto del elemento no debe exceder el de la actividad.
Descripción: - El Usuario elige la opción de agregar nuevo elemento. - La aplicación muestra el formulario de creación del elemento del presupuesto. - El Usuario ingresa los datos del elemento del presupuesto: descripción y presupuesto. - La aplicación genera el nuevo elemento, lo registra en la base de datos y muestra al Usuario la pantalla de detalles de la actividad con el listado de elementos del presupuesto actualizado.
Alternativas: Ninguna.
Postcondición: El elemento del presupuesto de la actividad se ha guardado.

Nombre del caso de uso: Actualizar elemento del presupuesto de la actividad.
Resumen: El Usuario actualiza un elemento del presupuesto específico de una actividad.
Actores: Usuario.
Precondición: El presupuesto del elemento no debe exceder el de la actividad.
Descripción: - El Usuario elige la opción de editar elemento. - El Usuario actualiza la descripción o el presupuesto. - La aplicación registra los cambios hechos en la base de datos.
Alternativas: Ninguna.
Postcondición: El elemento del presupuesto de la actividad se ha actualizado.

Nombre del caso de uso: Borrar elemento del presupuesto de la actividad.
Resumen: El Usuario borra un elemento del presupuesto específico de una actividad.
Actores: Usuario.
Precondición: Ninguna.
Descripción: - El Usuario elige la opción de borrar elemento. - El Usuario confirma la acción de borrar elemento. - La aplicación registra la eliminación del elemento en la base de datos.
Alternativas: Ninguna.
Postcondición: El elemento del presupuesto de la actividad se ha borrado.

APÉNDICE B

DICCIONARIO DE DATOS

PK significa llave primaria.

FK significa llave foránea.

Action			
Esta tabla contiene información sobre las acciones.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK action_id	bigint	ID de la acción.	No
name	varchar(255)	Nombre de la acción.	No

Activity			
Esta tabla contiene información sobre las actividades.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK activity_id	bigint	ID de la actividad.	No
name	varchar(255)	Nombre de la actividad.	No
start_date	timestamp	Fecha de inicio de actividad.	No
end_date	timestamp	Fecha de final de actividad.	No
budget	double	Presupuesto de la actividad.	No
impact	varchar(255)	Impacto de la actividad.	No
progress	integer	Progreso de la actividad	No
FK goal_id	bigint	ID del objetivo. FK relacionado con goal.goal_id.	No
FK user_id	bigint	ID del usuario. FK relacionado con users.user_id.	No

Activity_user			
Esta tabla contiene información sobre las actividades y los usuarios.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
FK activity_id	bigint	ID de la actividad. FK relacionado con activity.activity _id.	No
FK user_id	bigint	ID del usuario. FK relacionado con users.user_id.	No

Church _project			
Esta tabla contiene información sobre los proyectos de la iglesia.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK church_project_id	bigint	ID del proyecto.	No
name	varchar(255)	Nombre del proyecto.	No

Department			
Esta tabla contiene información sobre los departamentos.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK department_id	bigint	ID del departamento.	No
name	varchar(255)	Nombre del departamento.	No
mission	varchar(255)	Misión del departamento.	No
vision	varchar(255)	Visión del departamento.	No

FK department_up_department_id	bigint	ID del departamento arriba. FK relacionado con department.department_id.	Si
FK organization_id	bigint	ID de la organización. FK relacionado con organization.organization_id.	No

Fund_type			
Esta tabla contiene información sobre los tipos de fondos.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK fund_type_id	bigint	ID del tipo de fondo.	No
name	varchar(255)	Nombre de la acción.	No

Goal			
Esta tabla contiene información sobre los objetivos.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK goal_id	bigint	ID del objetivo.	No
description	varchar(255)	Descripción de lo que se quiere lograr.	No
quantity	integer	Cantidad o medida en que se quiere lograr.	No
purpose	varchar(255)	Propósito o importancia del cumplimiento del objetivo.	No
start_date	timestamp	Fecha de inicio del objetivo.	No
end_date	timestamp	Fecha de final del objetivo.	No
kpi_value	integer	Valor por medir del indicador clave de rendimiento.	No

impact	varchar(255)	Impacto del objetivo.	No
active	boolean	Campo para saber si el objetivo está activo o no.	No
FK action_id	bigint	ID de la acción. FK relacionado con action.action _id.	No
FK measurement_id	bigint	ID de la medida. FK relacionado con measurement.measurement_id.	No
FK fund_type_id	bigint	ID del tipo de fondo. FK relacionado con fund_type.fund_type_id.	No
FK church_project_id	bigint	ID del proyecto. FK relacionado con church_project.church_project_id.	Si
FK goal_up_goal_id	bigint	ID del objetivo. FK relacionado con goal.goal_id.	Si
FK user_id	bigint	ID del usuario. FK relacionado con users.user_id.	No

Item_budget			
Esta tabla contiene información detallada sobre el presupuesto de la actividad.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK item_id	bigint	ID del presupuesto detallado.	No
budget	double	Cantidad del presupuesto.	No
resource	varchar(255)	Recurso específico.	No
FK activity_id	bigint	ID de la actividad. FK relacionado con activity.activity_id.	No

Log_activities			
Esta tabla contiene información del log de la actividad.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK log_id	bigint	ID del log.	No
date	timestamp	Fecha del log.	No
progress	integer	Progreso de la actividad.	No
FK activity_id	bigint	ID de la actividad. FK relacionado con activity.activity_id.	No

Measurement			
Esta tabla contiene información sobre las medidas posibles.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK measurement_id	bigint	ID de la medida.	No
name	varchar(255)	Nombre de la medida.	No

Opportunities			
Esta tabla contiene información de las oportunidades del departamento.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK opportunity_id	bigint	ID de la oportunidad.	No
description	varchar(255)	Descripción de la oportunidad.	No
FK department_id	bigint	ID del departamento. FK relacionado con department.department_id.	No

Organization			
Esta tabla contiene información de las organizaciones.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK organization_id	bigint	ID de la organización.	No
name	varchar(255)	Nombre de la organización.	No

Role			
Esta tabla contiene información de los roles.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK role_id	bigint	ID del rol.	No
name	varchar(255)	Nombre del rol.	No

Strengths			
Esta tabla contiene información de las fortalezas del departamento.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK strength_id	bigint	ID de la fortaleza.	No
description	varchar(255)	Descripción de la fortaleza.	No
FK department_id	bigint	ID del departamento. FK relacionado con department.department_id.	No

Threats			
Esta tabla contiene información de las amenazas del departamento.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK opportunity_id	bigint	ID de la amenaza.	No
description	varchar(255)	Descripción de la amenaza.	No

FK department_id	bigint	ID del departamento. FK relacionado con department.department_id.	No

Users			
Esta tabla contiene información sobre los usuarios.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK user_id	bigint	ID del usuario.	No
email	varchar(255)	Correo electrónico del usuario.	No
password	varchar(255)	Contraseña del usuario.	No
name	varchar(255)	Nombre(s) del usuario.	No
last_name	varchar(255)	Apellido(s) del usuario.	No
phone_number	varchar(255)	Número de teléfono del usuario.	No
active	integer	Campo para saber si el usuario está activo o no.	No
FK department_id	bigint	ID del departamento. FK relacionado con department.department_id.	No

User_role			
Esta tabla contiene información sobre los roles de los usuarios.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
FK user_id	bigint	ID del usuario. FK relacionado con users.user_id.	No

FK role_id	bigint	ID del rol. FK relacionado con role.role _id.	No

Weakness			
Esta tabla contiene información de las debilidades del departamento.			
Atributo	**Tipo de dato**	**Descripción**	**Nulo**
PK weakness_id	bigint	ID de la debilidad.	No
description	varchar(255)	Descripción de la debilidad.	No
FK department_id	bigint	ID del departamento. FK relacionado con department.department_id.	No

REFERENCIAS

Aldehayyat, J. S. y Anchor, J. R. (2008). Strategic planning tools and techniques in Jordan: Awareness and use. *Strategic Change, 17*(7-8), 281-293.

Alex, B., Taylor, L., Winch, R., Hillert, G., Grandja, J., Bryant, J., . . . Stein, E. (2021). *Spring security reference.* Recuperado de https://docs.spring.io/spring-security/site/docs/current/reference/html5/

Allio, M. K. (2012). Strategic dashboards: Designing and deploying them to improve implementation. *Strategy & Leadership, 40*(5), 24-31.

Asana. (2021). *Gestiona en línea el trabajo, los proyectos y las tareas de tu equipo*. Recuperado de https://asana.com/

Bass, L., Clements, P. y Kazman, R. (2003). *Software architecture in practice* (2a. ed.). Boston: Addison-Wesley Professional.

Bauer, C., King, G. y Gregory, G. (2015). *Java persistence with Hibernate* (2a. ed.). Shelter Island, NY: Manning Publications.

Bechtell, M. L. (1996). Navigating organizational waters with hoshin planning. *National Productivity Review, 15*(2), 23-42.

Binstock, A. y Maple, S. (2019). *The largest survey ever of Java developers*. Recuperado de https://blogs.oracle.com/javamagazine/post/the-largest-survey-ever-of-java-developers

Biswas, R. y Ort, E. (2006). *The Java persistence API - a simpler programming model for entity persistence*. Recuperado de https://www.oracle.com/technical-resources/articles/java/jpa.html

Brantley, P., Jackson, D. y Cauley, M. (2015). *Becoming a mission-driven church*. Miami, FL: Pacific Press.

Bryson, J. M. (2015). Strategic planning for public and nonprofit organizations. *International Encyclopedia of the Social & Behavioral Sciences* (2a. ed.) (pp. 515-521). Amsterdam: Elsevier.

Byars, L. L. (1991). *Strategic management: Formulation and implementation. Concepts and cases* (3a. ed.). New York, NY: HarperCollins Publishers.

Cascade. (2021). *Strategy software for planning & execution*. Recuperado de https://www.cascade.app

Cooper, L. G. (2000). Strategic marketing planning for radically new products. *Journal of Marketing, 64*(1), 1-16.

Deblauwe, W. (2021). *Taming thymeleaf: practical guide to building a web application with spring boot and thymeleaf*. Morrisville, NC: Lulu Press.

Díaz González, Y. y Fernández Romero, Y. (2012). Patrón Modelo-Vista-Controlador. *Telemática, 11*(1), 47-57.

Doerr, J. (2018). *Measure what matters: How Google, Bono, and the Gates Foundation rock the world with OKRs.* New York, NY: Penguin.

Envisio. (2021). *Strategic planning and performance management software*. Recuperado de https://envisio.com/

Fernández, A. (2001). El Balanced Scorecard. *Revista de antiguos alumnos del IESE*, 81, 83.

Gellersen, H. W. y Gaedke, M. (1999). Object-oriented Web application development. *IEEE Internet Computing, 3*(1), 60-68.

Gierke, O., Darimont, T., Strobl, C., Paluch, M. y Bryant, J. (2021). *Spring data JPA - Reference documentation*. Recuperado de https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Good, M. (2018). *Thymeleaf with spring boot*. Recuperado de https://michaelcgood.com/wp-content/uploads/2018/01/Thymeleaf-spring-boot.pdf

Gracia del Busto, H. y Yanes Enríquez, O. (2012). Mapeo Objeto / Relacional (ORM). *Telemática, 10*(3), 1-7.

Harold, K. (2017). *Project management: a systems approach to planning, scheduling, and controlling* (12a. ed.). Hoboken, NJ: Wiley.

Heckler, M. (2021). *Spring boot, up and running: building cloud native Java and Kotlin applications*. Sebastopol, CA: O'Reilly.

Hemrajani, A. (2006). *Agile Java development with spring, hibernate and eclipse*. Indianapolis, IN: Sams Publishing.

Hibernate. (2021). *Hibernate ORM*. Recuperado de https://hibernate.org/orm/

Jazayeri, M. (2007). Some trends in web application development. *Future of Software Engineering (FOSE '07)*. Mineápolis, MN: IEEE.

JetBrains. (2020). *Java programming - the state of developer ecosystem in 2020 infographic*. Recuperado de https://www.jetbrains.com/lp/devecosystem-2020/java/

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., . . . Mark, P. (2021). Spring framework documentation. Recuperado de https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html

Kalkan, A. y Bozkurt, Ö. Ç. (2013). The choice and use of strategic planning tools and techniques in Turkish SMEs according to attitudes of executives. *Procedia - Social and Behavioral Sciences, 99*, 1016-1025.

Kaplan, R. S. y Norton, D. P. (1992). The balanced scorecard: measures that drive performance. *Harvard Business Review, 70*(1), 71-79.

Kaplan, R. S. y Norton, D. P. (1996). Using the balanced scorecard as a strategic management system. *Harvard Business Review, 74*(1), 75-85.

Kaplan, R. S. y Norton, D. P. (2004). The strategy map: Guide to aligning intangible assets. *Strategy & Leadership, 32*(5), 10-17.

Kaplan, R. S. y Norton, D. P. (2005). The office of strategy management. *Harvard Business Review, 83*(10), 72-80.

Klochkov, A. K. (2010). *Кпи и мотивация персонала: полный сборник практических инструментов*. Москва: ЭКСМО.

Kotler, P. (1997). *Marketing management: Analysis, planning, implementation, and control* (9a. ed.). Upper Saddle River, NJ: Prentice Hall.

Locke, E. A. y Latham, G. P. (1990). *A theory of goal setting & task performance*. Englewood Cliffs, NJ: Prentice Hall.

McChesney, C., Covey, S. y Huling, J. (2012). The 4 disciplines of execution. London: Simon & Schuster.

Meredith, J. R., Shafer, S. M. y Mantel, S. J. (2018). Project management: a strategic managerial approach (10a. ed.). Hoboken, NJ: Wiley.

Monday. (2021). *Una forma nueva de trabajar*. Recuperado de https://monday.com

Niven, P. R. y Lamorte, B. (2016). *Objectives and key results: Driving focus, alignment, and engagement with OKRs*. Hoboken, NJ: Wiley.

O'Neil, E. J. (2008). Object/relational mapping 2008: hibernate and the entity data model (Edm). *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, (pp. 1351-1356). Vancouver, Canada: ACM Press.

Paakki, J., Karhinen, A., Gustafsson, J., Nenonen, L. y Verkamo, A. I. (2000). Software metrics by architectural pattern mining. *Proceedings of the International Conference on Software: Theory and Practice (16th IFIP World Computer Congress)*, (pp. 325-332). Berlín, Alemania: Springer.

Pickton, D. W. y Wright, S. (1998). What's swot in strategic analysis? *Strategic change, 7*(2), 101-109.

Porter, M. (1985). *The competitive advantage: creating and sustaining superior performance*. New York, NY: Free Press.

Ramírez Rojas, J. L. (2017). *Procedimiento para la elaboración de un análisis FODA como una herramienta de planeación estratégica en las empresas*. Recuperado de http://www.uv.mx/iiesca/files/2012/12/herramienta2009-2.pdf

Reenskaug, T. M. (1979). *The original MVC reports*. Recuperado de https://www.duo.uio.no/bitstream/handle/10852/9621/Reenskaug-MVC.pdf

Ricca, F. y Tonella, P. (2001). Analysis and testing of web applications. *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001*. Toronto: IEEE.

Riquelme Leiva, M. (2016). *FODA: matriz o análisis FODA. Una herramienta esencial para el estudio de la empresa*. Recuperado de https://www.analisisfoda.com/

Rowley, D. J., Lujan, H. D. y Dolence, M. G. (1997). *Strategic change in colleges and universities: planning to survive and prosper* (1a. ed.). San Francisco, CA: Jossey-Bass.

Sammut-Bonnici, T. y Galea, D. (2015). PEST analysis. En C. L. Cooper (Ed.), *Wiley encyclopedia of management* (p. 1). Chichester, UK: John Wiley & Sons, Ltd.

Sarikaya, A., Correll, M., Bartram, L., Tory, M. y Fisher, D. (2018). What do we talk about when we talk about dashboards? *IEEE Transactions on Visualization and Computer Graphics, 25*(1), 682-692.

Shahin, A. y Mahbod, M. A. (2007). Prioritization of key performance indicators: An integration of analytical hierarchy process and goal setting*. International Journal of Productivity and Performance Management, 56*(3), 226-240.

Slack. (2021). *Slack es donde está el futuro del trabajo*. Recuperado de https://slack.com/

Spilca, L. (2020). *Spring security in action*. Shelter Island, NY: Manning Publications.

Stacey, R. D. (1993). *Strategic management and organisational dynamics*. London: Pitman.

Stack Overflow. (2021). *Stack overflow developer survey 2021*. Recuperado de https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

Steiner, G. A. (1979). *Strategic planning: what every manager must know*. New York, NY: Free Press.

Stonehouse, G. y Pemberton, J. (2002). Strategic planning in SMEs – some empirical findings. *Management Decision, 40*(9), 853-861.

Taylor, R. N., Medvidović, N. y Dashofy, E. M. (2010). *Software architecture: foundations, theory, and practice*. Hoboken, NJ: John Wiley.

Tennant, C. y Roberts, P. A. (2000). Hoshin Kanri: a technique for strategic quality management. *Quality Assurance, 8*(2), 77-90.

Tennant, C. y Roberts, P. (2001). Hoshin Kanri: a tool for strategic policy deployment. *Knowledge and Process Management, 8*(4), 262-269. https://doi.org/10.1002/kpm.121

Thymeleaf. (2021). *Thymeleaf*. Recuperado de https://www.thymeleaf.org/

Trainer, J. F. (2004). Models and tools for strategic planning. *New Directions for Institutional Research, 2004*(123), 129-138.

Trello. (2021). *Trello siempre hace avanzar*. Recuperado de https://trello.com

Walls, C. (2016). *Spring boot in action*. Shelter Island, NY: Manning Publications.

Walls, C. (2019). *Spring in action* (5a. ed.). Shelter Island, NY: Manning Publications.

Webb, P., Syer, D., Long, J., Nicoll, S., Winch, R., Wilkinson, A., . . . Frederick, S. (2021). *Spring boot reference documentation*. Recuperado de https://docs.spring.io/spring-boot/docs/current/reference/html/

Webster, J. L., Reif, W. E. y Bracker, J. S. (1989). The manager's guide to strategic planning tools and techniques. *Strategy & Leadership, 17*(6), 4-48.

WorkBoard. (2021). *OKR software for leading organizations*. Recuperado de https://www.workboard.com/

Wrike. (2021). *Versatile and robust project management software*. Recuperado de https://www.wrike.com/