Universidad de Sancti Spíritus "José Martí Pérez"
Facultad de Ciencias Técnicas y Económicas
Carrera de Ingeniería Informática

Trabajo de Diploma para optar por el título de Ingeniería Informática

Desarrollo de una API RESTful para el subsistema informático de gestión del plan de trabajo individual del profesor universitario

Development of a RESTful API for the informatic subsystem or the individual work plan of university professors

Autor(a): Rossana del Carmen Torres González
Tutor(a): Dr.Sc. Lydia Rosa Ríos Rodríguez

Sancti Spíritus, Cuba 2022

DEDICATORIA

A mi familia y amigos, en especial a mis padres.

RESUMEN

La Universidad de Sancti Spíritus “José Martí Pérez” está inmersa en un proceso de virtualización de todos sus procesos. En este contexto se desarrolla la presente investigación con el objetivo de desarrollar una API REST que permita a cuadros y funcionarios de las áreas y dependencias planificar, organizar, controlar y evaluar el plan de trabajo individual del profesor universitario. Para lo cual inicialmente se buscan los fundamentos teóricos, metodológicos y tecnológicos que sustentan la propuesta de solución, luego se diseña empleando la metodología RUP y UML, como lenguaje de modelado, e implementa utilizando el entorno de desarrollo Node.js, NestJS como framework y PostgreSQL como gestor de base de datos. El producto resultante parte de las necesidades del cliente, realiza el tratamiento de errores y controla el acceso a sus opciones. Las pruebas y técnicas aplicadas al software final permitieron evaluar su correcto funcionamiento.

ABSTRACT

The objective of this research is to develop a REST API for the management of the Individual Work Plan of the university professor of the University of Sancti Spíritus "José Martí Pérez". For its achievement, the study of computer solutions and their corresponding tools is documented, as a basis for the development of a computer service that responds to the needs found. For the development of the system, the RUP methodology was used and UML as modeling language. For the implementation of the API, we worked on the development environment Node.js, and NestJS as framework, with PostgreSQL database manager.

CONTENIDO

INTRODUCCIÓN

La introducción y el uso de las Tecnologías de la Información y las Comunicaciones (TIC) y de Internet ha significado a escala mundial un salto vertiginoso en el desarrollo científico técnico. (González Reyes & Febles Estrada, 2021). De acuerdo con (Hernández Pozo et al., 2022) las TIC abren nuevos caminos e imponen la premura de realizar cambios, primeramente, en la mentalidad de los individuos a través del desarrollo de una cultura informacional; y en segundo término en la proyección estratégica y diseño de todos los procesos inherentes a las organizaciones.

Es por ello que en los últimos años la mayoría de los países han establecido e implementado proyectos, políticas y estrategias para promover el uso de las TIC y aprovechar los beneficios y los aportes que estas ofrecen (Alcibar et al., 2018). En relación al tema, Cuba ha identificado desde muy temprano la conveniencia y necesidad de dominar e introducir en la práctica social las TIC y lograr una cultura digital como una de las características imprescindibles del hombre nuevo, lo que facilitaría a nuestra sociedad acercarse más hacia el objetivo de un desarrollo sostenible (Carbo, 2016).

Conceptualmente, la informatización de la sociedad en Cuba, se define como el proceso de utilización ordenada y masiva de las tecnologías de la información y las comunicaciones para satisfacer las necesidades de información y conocimiento de todas las personas y esferas de la sociedad. Este proceso busca lograr eficacia y eficiencia, que permitan una mayor generación de riquezas y hagan sustentables el aumento sistemático de la calidad de vida de los ciudadanos, sobre una política preferentemente orientada al uso social e intensivo de los recursos de las TIC,

buscando extender sus beneficios a la mayor parte posible de la población e instituciones (González Reyes & Febles Estrada, 2021).

El país lleva a cabo un programa importante para la informatización de la sociedad desde finales de la década de los 80, con el propósito de promover el uso masivo de las tecnologías a escala nacional, a partir de los objetivos estratégicos generales que el país se propone (Abreu et al., 2018). La creación en 1987 de los Joven Club de Computación y Electrónica; del Portal de la Red de Salud de Cuba en 1992; de la Universidad de las Ciencias Informáticas (UCI) en el año 2002 con el objetivo fundamental de la formación de recursos humanos en este campo; del Ministerio para la Informática y las Comunicaciones (MIC), que ha constituido una estrategia vital para el desarrollo de las TIC en Cuba; la creación de la Unión de Informáticos de Cuba (UIC) y la enseñanza de la computación, masiva y gratuita, en las escuelas, son ejemplos de los esfuerzos del país en función de un propósito noble: la informatización de la sociedad (González Reyes & Febles Estrada, 2021).

La informatización de la educación es un factor principal en la existencia y desarrollo de una educación superior moderna, pues su objetivo primordial es el desarrollo y crecimiento del potencial de cada individuo. En este escenario, las Instituciones de Educación Superior (ES) constituyen uno de los sectores sobres los que más se ha enfocado el país en función de la informatización, tanto del sistema de enseñanza y aprendizaje como de la gestión de sus procesos. A la par, se fue gestando la superación profesional y capacitación de toda la población para hacer uso eficiente de estos recursos (Abreu et al., 2018). Como resultado, actualmente el Ministerio de Educación Superior (MES) en Cuba tiene estructurada una red universitaria de datos

(RedUniv) que se concibió con el objetivo de construir y operar de forma estable, una infraestructura nacional de redes capaz de propiciar y promover el proceso de integración de las nuevas tecnologías de la información y las comunicaciones en los procesos fundamentales de la Educación Superior (ES) cubana; en tanto permita divulgar el quehacer científico, cultural e ideológico de universidades y centros de investigación, pero no es posible considerar que se haya alcanzado cierto grado de sistematización al respecto (Ruíz Jhones & Vidal Larramendi, 2019).

Actualmente, las universidades trabajan en la incorporación de las tecnologías de la información y las comunicaciones a sus procesos docentes, investigativos y de gestión, con el objetivo de garantizar la efectividad de la gestión de sus procesos y que, por supuesto, les conduzca a la consecución de sus propósitos y a la concreta realización de su misión y visión institucionales (Castanedo Abay, 2019). Tal es el caso de la universidad de la Universidad de Sancti Spíritus “José Martí Pérez”, donde se han desarrollado numerosos proyectos encaminados a una mejor gestión universitaria.

De acuerdo con lo anteriormente planteado, el departamento de Ingeniería Informática de la Facultad Ciencias Técnicas y Económicas de la Universidad de Sancti Spíritus “José Martí Pérez” se ha propuesto la tarea de informatizar la gestión del Plan de Trabajo Individual (PTI) del profesor universitario, con el objetivo de un mejor control y organización del mismo.

Sobre el estudio realizado acerca de este proceso se pudo constatar que actualmente se realiza de forma manual, y la institución no cuenta con una herramienta informática que preserve los planes de los profesores por un período de hasta cinco años, lo que dificulta el seguimiento sistemático de su desempeño.

Esta **situación problemática** genera como **problema de investigación**: ¿Cómo contribuir a la informatización de la gestión del plan de trabajo individual del profesor universitario?

Para dar solución al problema planteado, se concreta como **objetivo general** de la investigación, crear una API REST que les permita a cuadros y funcionarios de las áreas y dependencias de la universidad “José Martí Pérez”, planificar, organizar, controlar y evaluar el plan de trabajo individual del profesor universitario.

Para orientar y dar cumplimiento al objetivo general, se formulan las siguientes **preguntas y tareas de investigación**:

Preguntas de investigación

1. ¿Cuáles son los fundamentos teóricos, metodológicos y tecnológicos que sustentan la creación de una API REST para la gestión del plan de trabajo individual del profesor universitario?
2. ¿Cómo diseñar una API REST que facilite la gestión del plan de trabajo individual del profesor universitario?
3. ¿Cómo implementar una API REST que facilite la gestión del plan de trabajo individual del profesor universitario?

Tareas de investigación

1. Definir los fundamentos teóricos, metodológicos y tecnológicos que sustentan la creación de una API REST para la gestión del plan de trabajo individual del profesor universitario.

1. Diseñar una API REST para la gestión del plan de trabajo individual del profesor universitario.
2. Implementar una API REST para facilitar la gestión del plan de trabajo individual del profesor universitario.

Atendiendo a lo planteado anteriormente, la estructura de la investigación cuenta con:

Capítulo 1: Fundamentos teóricos, metodológicos y tecnológicos

En este capítulo se abordan los principales conceptos asociados al objeto y campo de estudio. También se realiza un análisis sobre las principales tecnologías, herramientas y metodologías de desarrollo de software, seleccionadas para el análisis, diseño e implementación de la aplicación.

Capítulo 2: Diseño de la API REST

En este capítulo se describe el modelo del negocio e identifican los procesos involucrados en el mismo, junto con las reglas que lo rigen. Se analizan los principales requisitos funcionales y no funcionales a tener en cuenta en el desarrollo de la API, y se muestra el modelo entidad-relación y modelo físico de la base de datos utilizada.

Capítulo 3: Implementación de la API REST

En este capítulo se describe de forma general la implementación de la API REST, teniendo en cuenta la seguridad, el tratamiento de errores, y su documentación. Se definen las pruebas de integración para comprobar el correcto funcionamiento del sistema.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS, METODOLÓGICOS Y TECNOLÓGICOS QUE SUSTENTAN LA CREACIÓN DE UNA API REST PARA LA GESTIÓN DEL PLAN DE TRABAJO INDIVIDUAL DEL PROFESOR UNIVERSITARIO

Este capítulo está orientado a la fundamentación teórica de la investigación, con el objetivo de revisar los principales conceptos del objeto de estudio; analizar las posibles herramientas informáticas planteadas como solución a la situación problemática identificada y las tecnologías de software que propicien una óptima implementación de la API.

1.1 Las Tecnologías de la Información y las Comunicaciones en la Educación Superior cubana

Las TIC se han constituido en elementos sustantivos inherentes al desarrollo de todas las esferas de la vida. Su uso en las universidades condicionan de manera favorable que estas funcionen estructuradamente; tanto en el desarrollo de investigaciones científicas, acciones de innovación tecnológica, gestión del conocimiento, y la actividad pedagógica-metodológica como eje fundamental del proceso docente educativo (Navarro et al., 2018). Es por ello que actualmente las Instituciones de Educación Superior trabajan en la incorporación de estas tecnologías a sus procesos docentes, investigativos y de gestión; con el objetivo de garantizar la efectividad de la gestión de sus procesos y que, por supuesto, les conduzca a la consecución de sus propósitos y a la concreta realización de su misión y visión institucionales (Castanedo Abay, 2019).

Esta revolución tecnológica ha generado nuevas y mayores oportunidades de interacción que requieren, a su vez, una capacidad de adaptación, tanto de los individuos como de las instituciones. Frente a este panorama, las instituciones de educación superior han venido experimentando cambios y realizando iniciativas para convertirse en contextos altamente tecnológicos. Este tipo de instituciones han entendido que el uso pertinente de las TIC y los procesos de innovación que las incorporan les permiten alcanzar sus principales objetivos, tanto formativos como de gestión (Nakano et al., 2016).

En Cuba, el uso de las TIC para la gestión universitaria, comenzó en la década de los 90, con la introducción, por decisión del MES, del sistema ASSETS para las estructuras económicas contables y de recursos humanos. Posteriormente en la década del 2000,

se implementó el Sistema de Gestión de la Nueva Universidad (SIGENU) para las tareas de las secretarías docentes, protagonizado por la Universidad Tecnológica de La Habana (CUJAE). A partir de entonces, comienza un largo proceso de implementación y uso de las TIC en todas las universidades del país, por ejemplo, la plataforma de aprendizaje MOODLE para la educación a distancia y como complemento a la presencialidad, mediante la habilitación de aulas virtuales con múltiples opciones de contenido multimedia donde se garantiza el intercambio entre los estudiantes y profesores (Jhones & Larramendi, 2019). La red universitaria cubana (REDUNIV), que se gestiona desde el MES, es otro ejemplo de la aplicación de las TIC para la difusión de servicios y contenidos (Navarro et al., 2018).

Esfuerzos adicionales interesantes se han realizado por un grupo de universidades. La Universidad de las Ciencias Informáticas (UCI), por ejemplo, cuenta con una plataforma integrada de sistemas de gestión denominada AKADEMOS, desarrollada a imagen y semejanza de esta universidad.

En sentido general, se disponen de una gran variedad de herramientas informáticas implementadas en las propias universidades que facilitan la gestión de la información en algunas actividades que se desarrollan. La CUJAE, la Universidad Central “Marta Abreu” de Las Villas (UCLV), y muchos otros centros poseen diferentes sistemas que soportan la gestión y los flujos de información de algunos procesos; como es también el caso de la Universidad de Sancti Spíritus “José Martí Pérez”.

1.2 La informatización en la Universidad de Sancti Spíritus “José Martí Pérez”

La Universidad de Sancti Spíritus “José Martí Pérez” ha sido impulsora de numerosas investigaciones y proyectos orientados a la informatización de los procesos que se

gestionan, no solo en la institución sino también en la sociedad. Ejemplo de ello lo constituye el Sistema de Gestión de Información diseñado e implementado en el Departamento de Ingeniería Informática (SIGIDI), con el objetivo de contribuir al control de los trabajadores, medios básicos y actividades de la carrera de Ingeniería Informática y del propio departamento (Sánchez Martínez et al., 2017).

Otro ejemplo lo constituye, el desarrollo de una aplicación web basada en mapas conceptuales con el fin de apoyar el aprendizaje del proceso de obtención de biogás y las tecnologías asociadas a ello; encabezado por trabajadores docentes del departamento de Ingeniería Informática de la Facultad Ciencias Técnicas y Empresariales con el apoyo del Centro de Estudios en Energía y Procesos Industriales (CEEPI) pertenecientes al proyecto “Estudio Prospectivo para la producción de biogás con fines energéticos en la provincia de Sancti Spíritus” de la propia universidad. Esta herramienta informática está dirigida a decisores y especialistas en el campo de la producción de este combustible, así como a todo aquel interesado en su comprensión.

Con el propósito de continuar contribuyendo a la informatización de los procesos universitarios, el departamento de Ingeniería Informática se propone, mediante esta investigación, comenzar con la informatización de la gestión del Plan de Trabajo Individual de los profesores.

1.3 Plan de Trabajo Individual del profesor universitario

El Plan de Trabajo Individual del profesor universitario, constituye el instrumento de planificación que permite definir las actividades a realizar por el mismo, su programación durante el año y la declaración prevista de los resultados (Torres Vivanco & Castellanos Gutiérrez, 2021). La gestión de estos planes de trabajo contribuyen a la

evaluación de desempeño del docente, que, basada en la recopilación de evidencias, y de acuerdo con las actividades establecidas, permite valorar los resultados y la calidad del trabajo realizado durante el año, así como la efectividad de la labor desarrollada en la formación integral de los estudiantes, el ejemplo personal y el prestigio (Bautista et al., 2018).

En este contexto, el sistema para la planificación, control y evaluación de los profesores universitarios toma como referencia las políticas y estrategias trazadas por el Ministerio de Educación Superior (MES) para definir las actividades del plan de trabajo de los docentes, teniendo en cuenta los siguientes aspectos (Bautista et al., 2018):

- Trabajo docente-educativo en pregrado y posgrado
- Trabajo político-ideológico
- Trabajo metodológico
- Trabajo de investigación e innovación
- Superación
- Extensión universitaria

La elaboración del plan de trabajo de los profesores universitarios, se realiza cada año natural por el jefe del departamento docente, de acuerdo con las tareas que contribuyen a las prioridades y objetivos del curso, la disciplina laboral, la labor educativa y el incremento de la calidad de la educación superior, confeccionan los planes individuales de sus profesores, alineados con las necesidades de la universidad, facultad y departamento (del Pino Sarduy & Fernández Álvarez, 2021).

Posteriormente, en el transcurso del año, se realiza un control sistemático del desarrollo de las actividades planificadas, que permite adoptar las decisiones adecuadas para su cumplimiento y así obtener los resultados programados (Torres Vivanco & Castellanos Gutiérrez, 2021).

Al concluir el año docente se realiza una evaluación con el objetivo de valorar los resultados y la calidad del trabajo realizado; se recoge, además, los incumplimientos y se especifican las recomendaciones para la próxima etapa (del Pino Sarduy & Fernández Álvarez, 2021).

Para esta evaluación el jefe de departamento tendrá en consideración los aspectos siguientes: la autoevaluación del profesor universitario, la categoría docente, el control sistemático realizado al desarrollo de las actividades durante el año, las opiniones recogidas de los profesores, trabajadores y de la organización sindical en los análisis periódicos realizados en el departamento, la opinión de los estudiantes emitida a través de la Federación Estudiantil Universitaria, así como el criterio de otros dirigentes que han intervenido en el desarrollo del trabajo realizado por el profesor (Torres Vivanco & Castellanos Gutiérrez, 2021). La calificación final se expresa como:

- Excelente: cuando se obtienen resultados relevantes y se contribuye, de forma significativa, a los logros del centro.
- Bien: cuando se cumple con calidad el trabajo planificado.
- Regular: cuando existen incumplimientos en el plan de trabajo individual.
- Mal: cuando existen incumplimientos significativos y/o reiterativos.

1.4 Tecnologías informáticas utilizadas para el diseño e implementación de la API REST

1.4.1 Servicio web

Un servicio web es un conjunto de protocolos y estándares que proporcionan un medio de interoperabilidad entre diferentes aplicaciones de software a través de la red, independientemente del lenguaje de programación, el hardware o el sistema operativo que ambas manejen (Coral Quinto, 2018). Son una consecuencia natural de la evolución de la web, ya que, desde sus inicios, como forma de compartir y distribuir información a escala global, convirtiéndose en una gigantesca biblioteca de contenidos distribuidos, la web ha ido ampliando su alcance para permitir formas de interacción más sofisticadas entre el cliente y el servidor (Haro et al., 2019).

Existen numerosas tecnologías para la implementación de los servicios web, las más reconocidas son SOAP y REST.

SOAP (*Simple Object Access Protocol*) es un protocolo sobre el que se establece el intercambio de información con el servicio web (Pacheco Laje, 2018). La arquitectura de un servicio web que implemente este protocolo consta de tres entidades:

- Proveedor de servicios: es la entidad direccionable de la red que acepta y ejecuta la solicitud del consumidor.
- Registro de servicios: es un directorio basado en la red que contiene los servicios disponibles.
- Consumidor de servicios: es una aplicación, un servicio o algún otro tipo de módulo de software que requiere un servicio.

Para la comunicación entre estas entidades los mensajes y las invocaciones de métodos se definen como documentos XML y se envían a través de un protocolo de transporte SMTP, FTP o HTTP (Tihomirovs & Grabis, 2016).

REST (*REpresentational State Transfer*): es un estilo de arquitectura de software cliente-servidor que utiliza el protocolo HTTP para transmitir datos en formatos XML, HTML, JSON o texto sin formato (Tihomirovs & Grabis, 2016). Así como REST es el estilo de arquitectura, RESTful es el término con que se definen a los servicios que la implementan. Las solicitudes a un servicio RESTful generalmente constan de los siguientes componentes:

- Ruta: https://api.example.com/user
- Método HTTP: GET (leer los datos de una entidad en concreto), PUT (actualizar una entidad existente o crearla si no existe), DELETE (borrar una entidad en concreto), POST (añadir información a una entidad ya existente).
- Encabezado: información adicional (opcional) que el cliente debe transmitir en la solicitud, como credenciales de autorización, tipo de contenido del cuerpo, agente de usuario para definir qué tipo de aplicación está realizando la solicitud, y más.
- Parámetros: campos variables (opcionales) que alteran cómo se devolverá el recurso.
- Cuerpo: (opcional) contiene datos que deben enviarse al servidor.

Aunque SOAP ha sido la opción preferida y elegida por muchas empresas, para otras resulta demasiado compleja y poco flexible. Por esta razón, se ha comenzado a utilizar servicios basados en REST para mostrar cantidades de datos masivas (Tihomirovs & Grabis, 2016).

1.4.2 API

Las interfaces de programación de aplicaciones (*Application Programming Interface*, *API*) son componentes de software que exponen servicios o datos proporcionados por una aplicación a través de un conjunto de recursos predefinidos, como métodos, objetos o URI. Mediante el uso de estos recursos, otras aplicaciones pueden acceder a los datos o servicios sin tener que implementar los objetos y procedimientos subyacentes (Márquez Martín, 2022).

Al igual que una interfaz gráfica facilita a un usuario el uso de un programa, una API facilita a un desarrollador la creación de una aplicación que sea capaz de interactuar con software de terceros, proporcionando una serie de métodos que le permiten hacer uso de funciones de ese software externo sin entender exactamente cómo funciona (Fernández Fontao, 2018). Suelen ser utilizadas para desarrollar aplicaciones web, pero también para interactuar con sistemas operativos, con base de datos, con librerías de software o incluso con componentes del hardware (Márquez Martín, 2022).

Una API REST es un tipo de servicio web que utiliza la arquitectura REST para atender una solicitud realizada desde una aplicación cliente. Se consideran más fácil de usar que un protocolo prescrito como SOAP, que tiene requisitos específicos como mensajería XML y seguridad integrada y cumplimiento de transacciones que lo hacen más lento y pesado. Por el contrario, REST es un conjunto de pautas que se pueden implementar según sea necesario, lo que hace que las API sean más rápidas y livianas, y con mayor escalabilidad (Prayogi et al., 2020).

Por otra parte, es de importancia destacar que, aunque las API y los servicios web pueden facilitar la transferencia de datos entre aplicaciones a través de Internet, no son

lo mismo y los términos no deben usarse indistintamente en todos los casos. Todos los servicios web son API, pero no todas las API son servicios web (Monago Ruiz, 2019).

1.5 Metodologías y herramientas

La creación de una API REST se puede realizar utilizando una variedad de lenguajes y plataformas para compilarlos y ejecutarlos. Para el desarrollo de este proyecto se optó por trabajar sobre el entorno de ejecución de JavaScript, Node.js, ya que es fácilmente escalable, tiene baja latencia y posee una gran cantidad de paquetes gratuitos que se actualizan periódicamente. A partir de esta base, se estudian y analizan otras tecnologías referentes al ámbito de desarrollo.

1.5.1 Metodología de desarrollo de software

Una metodología de desarrollo de software es un marco de trabajo que se utiliza para estructurar, planificar y controlar el proceso de desarrollo de un sistema informático. Cada una de las metodologías disponibles es más adecuada para tipos específicos de proyectos, basados en consideraciones técnicas, organizacionales, de proyecto y de equipo (Ríos et al., 2017).

Metodología RUP

El *Rational Unified Process* (RUP) es un proceso de ingeniería de software bien definido y estructurado y junto con el lenguaje unificado UML, constituye la metodología más utilizada para el análisis, implementación y documentación de sistemas (Kroll & Kruchten, 2003). Proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades en una organización de desarrollo y con el propósito de asegurar

una producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales con unos costos y calendario predecibles (Bernstein & Booch, 2020).

Esta metodología proporciona una estructura bien definida para el ciclo de vida de un proyecto, que consta de cuatro fases de desarrollo: Inicio, Elaboración, Construcción y Transición.

Inicio

- El objetivo principal es establecer el alcance del sistema de manera adecuada.
- Se establece el caso de negocio que incluye el contexto empresarial, se identifican los requerimientos del sistema, los actores del mismo, y se propone una arquitectura aproximada (Fernández & Cadelli, 2014).

Elaboración

- Es la encargada de determinar la solución técnica del proyecto.
- Incluye las actividades de comunicación y modelado del modelo general del proceso, lo que aumenta la representación de la arquitectura al incluir cinco puntos de vista distintos del software: los modelos de caso de uso, de requerimientos, de diseño, de implementación y de despliegue (Martínez & Martinez, 2000).

Construcción:

- El enfoque principal está en el desarrollo de componentes y otras características del sistema que harán que cada caso de uso sea operativo para los usuarios finales (Martínez & Martinez, 2000).
- Esta es la fase en la que tiene lugar la mayor parte de la codificación.

Transición:

- El objetivo principal es transitar el sistema desde el desarrollo hasta la producción, poniéndolo a disposición del usuario final.

- El sistema pasa por una fase de evaluación y se compara con el nivel de calidad establecido en la fase de inicio (Fernández & Cadelli, 2014).

Según (Kroll & Kruchten, 2003) RUP es una guía en la aplicación efectiva de las mejores prácticas para el desarrollo de software; con el desarrollo iterativo, la adopción de un enfoque centrado en la arquitectura, mitigando el riesgo en cada etapa del proceso y verificando la calidad del software resultado.

Se opta por RUP como metodología a seguir en aras de, a partir de una exhaustiva documentación y planeación, facilitar el desarrollo del software y obtener buenos resultados en su implementación.

1.5.2 Entorno de desarrollo Node.js

Node.js es un entorno de desarrollo multiplataforma, de código abierto, para el lenguaje de programación JavaScript, con E/S (entrada y salida) de datos en una arquitectura orientada a eventos, y basado en el motor V8 de Google. Remplaza la representación del lado del cliente con la implementación del código del lado del servidor (Pham, 2020).

Node.js maneja múltiples operaciones con conexiones simultáneas en tiempo real, sin que las operaciones de E/S interfieran entre sí. Esto ha permitido que Node.js se convierta en un competidor para el desarrollo web de servidor (Tacilla Ludeña, 2016).

Sin embargo, muchas veces la sintaxis de Node.js puede resultar de bajo nivel para realizar ciertas tareas típicas de una aplicación orientada a web, por lo que, los desarrolladores al trabajar sobre esta plataforma se auxilian de frameworks JavaScript. Estos frameworks agregan una capa de abstracción sobre Node.js con lo cual facilitan y añaden servicios adicionales a los ya existentes (Puciarelli, 2020).

1.5.3 Framework

Los frameworks son entornos de trabajo basados en librerías y módulos que simplifican de manera rápida el desarrollo de aplicaciones específicas (Cabrera Sarmiento, 2021). Ofrecen funcionalidades integradas, así como una estructura y una manera uniforme de trabajar, por lo que, es mucho más sencillo crear y desarrollar distintos sistemas de software, reduce los tiempos y evita que se cometan errores (Monago Ruiz, 2019).

Actualmente el ecosistema de la plataforma Node.js tiene una gran cantidad de bibliotecas y frameworks para crear softwares que se superponen en capacidades o solo tienen diferencias menores (Demashov & Gosudarev, 2019). Para el desarrollo de este proyecto se consideró el uso de Express.js.

Express.js es un framework flexible de Node.js que proporciona un conjunto sólido de funciones como manejo de rutas, archivos estáticos, uso de motor de plantillas, integración con bases de datos, manejo de errores, middlewares, entre otras, para aplicaciones web y móviles (Demashov & Gosudarev, 2019). Sus características minimalistas unidas a su velocidad de respuesta lo convierten en un fuerte candidato para el desarrollo de API REST (Sánchez Salas, 2022). Sin embargo, no requiere de una arquitectura específica, lo que puede traer como consecuencia una aplicación ineficiente y menos optimizada. Además, no cuenta con un método avanzado para las pruebas unitarias de los componentes del software, lo que no solo consume tiempo, sino que ralentiza la tasa de productividad de la aplicación.

En aras de dar solución a estos inconvenientes, se estudiaron otras alternativas de frameworks Node.js que cumplieran principalmente con una estructura mejor organizada. Como resultado de esta búsqueda, se encontró NestJS.

NestJS fue desarrollado con la misión de crear aplicaciones de servidor de Node.js efectivas y escalables. Es compatible con el lenguaje JavaScript y TypeScript y hace uso de frameworks de servidor HTTP robustos, como Express.js y Fastify. Tiene un foco en la arquitectura, ofreciendo un marco de trabajo y una serie de herramientas preconfiguradas, que permiten desarrollar proyectos del lado del servidor con mayor agilidad y homogeneidad (Romeu, 2020).

Como todo framework, Nest facilita mucho el desarrollo y aporta productividad, además de promover prácticas deseables que aumentan la calidad de los proyectos. Una característica de gran ventaja es que ofrece funcionalidades base para desarrollar las pruebas unitarias para componentes y pruebas e2e para aplicaciones automáticamente (Pham, 2020).

Como resultado del estudio de estos dos marcos de trabajo de Node.js, se pudo concluir que tanto Express.js como NestJS son muy buenas opciones a tener en cuenta para el desarrollo de software. Ambos presentan características muy similares, sin embargo, en algunos aspectos, como la arquitectura y la implementación de las pruebas, NestJS le presta al desarrollador mejores funcionalidades y herramientas que le facilitan el trabajo y proporcionan un producto de mejor calidad. NestJS está muy bien documentado y es sencillo de aprender. Hay que destacar también, como se menciona anteriormente, que este framework usa Express.js de forma predeterminada. Por lo tanto, para el desarrollo de este proyecto se optó por el uso de NestJS como framework para el diseño e implementación de la API.

1.5.4 Sistema Gestor de Base de Datos

Una base de datos es una recopilación de grandes cantidades de datos relacionados entre sí, almacenados para su manejo y consulta posterior (Aguilera Martínez, 2019). Existen numerosas clasificaciones de base de datos, pero las más populares y que más historia tienen, son las bases de datos relacionales, las cuales estructuran la información en tablas, filas y columnas; donde cada fila es un registro, con un identificador único, y las columnas constituyen los atributos de los datos (Caqui Vargas & Pareja Limaco, 2018).

Un sistema gestor de bases de datos es un software constituido por una serie de programas dirigidos a crear, gestionar y administrar la información que se encuentra en la base de datos. Su principal objetivo es servir de interfaz entre los usuarios y las aplicaciones para facilitar la organización de los datos, garantizar su accesibilidad, calidad e integridad, brindando a su vez una manera eficaz de administrar esa información (Saltos Pérez, 2022).

PostgreSQL es un gestor de bases de datos relacional y orientado a objetos. Su licencia y desarrollo es de código abierto, siendo mantenida por una gran comunidad de desarrolladores y colaboradores de forma libre y desinteresadamente. Presenta fácil accesibilidad, es multiplataforma y está disponible para su utilización en casi todos los sistemas operativos utilizados en la actualidad sin disminuir su rendimiento. Es reconocido actualmente como uno de los sistemas gestores de bases de datos relacionales más potentes del mercado (Marrero et al., 2020).

1.5.5 Enterprise Architect

Enterprise Architect (EA) es una herramienta integral de diseño y análisis UML para UML, SysML, BPMN y muchas otras tecnologías. Respalda el desarrollo de software desde la recopilación de requisitos hasta las etapas de análisis, modelos de diseño, pruebas y mantenimiento. Es una herramienta gráfica multiusuario, basada en Windows, diseñada para ayudar a crear un software sólido y fácil de mantener. También presenta una salida de documentación flexible y de alta calidad (Lenawati & Sari, 2022).

1.5.6 JSON Web Token

Un *JSON Web Token* (JWT) es un token de acceso estandarizado que permite el intercambio seguro de datos entre dos partes. Contiene toda la información importante sobre una entidad, lo que implica que no hace falta consultar una base de datos ni que la sesión tenga que guardarse en el servidor. Son especialmente populares en los procesos de autentificación, ya que es posible cifrar mensajes cortos, dotarlos de información sobre el remitente y demostrar si este cuenta con los derechos de acceso requeridos (Ahmed & Mahmood, 2019).

1.5.7 Swagger

Para el desarrollo de API REST es esencial contar con una documentación ordenada y comprensible, para que los desarrolladores puedan comenzar a usarla fácilmente. Swagger es una serie de reglas, especificaciones y herramientas que ayudan a documentar las API.

Swagger especifica la lista de recursos que están disponibles en la API REST y las operaciones que se pueden llamar en esos recursos. También especifica la lista de

parámetros de una operación, incluido el nombre y el tipo de los parámetros, si los son necesarios u opcionales, e información sobre los valores aceptables para los mismos. Puede incluir el esquema JSON que describe la estructura del cuerpo de solicitud que se envía a una operación en una API REST, y a su vez describe la estructura de los cuerpos de respuesta que se devuelven de una operación (De, 2017).

1.5.8 Postman

Postman es una herramienta dirigida a desarrolladores web que permite realizar peticiones HTTP a cualquier API. Es muy útil a la hora de programar y hacer pruebas, puesto que ofrece la posibilidad de comprobar el correcto funcionamiento de los proyectos en desarrollo. Ofrece un conjunto de funcionalidades que ayudan a organizar las peticiones en colecciones, hacer y automatizar pruebas, mantener equipos sincronizados y crear objetos simulados de API (De, 2017).

1.5.9 GitHub

GitHub es una plataforma de alojamiento que ofrece a los desarrolladores la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, usando un sistema de control de versiones llamado Git. Facilita la organización de proyectos y permite la colaboración de varios desarrolladores en tiempo real (Monago Ruiz, 2019).

1.6 Conclusiones parciales

Posterior al estudio realizado en el presente capítulo, se arriban a las siguientes conclusiones parciales:

- Se determinó para la investigación el desarrollo de una API REST como base para un futuro sistema de gestión del plan de trabajo individual del profesor universitario.

- Se seleccionó la metodología *Rational Unified Process (RUP)* para desarrollo del software gracias a su documentación detallada y el uso de *Unified Modeling Language (UML)* para el modelado del sistema.
- Se eligió el entorno de desarrollo Node.js, con el apoyo del framework NestJS ya que cuenta con una estructura bien definida.
- Se optó por el uso de PostgreSQL como gestor de base de datos gracias a su alta disponibilidad y fácil manejo.

CAPÍTULO 2: DISEÑO DE UNA API PARA FACILITAR LA GESTIÓN DEL PLAN DE TRABAJO INDIVIDUAL DEL PROFESOR UNIVERSITARIO

En este capítulo, basados en la metodología de desarrollo de software RUP, se realiza un estudio del modelo del negocio, identificando los principales actores que intervienen en el mismo, y las reglas del negocio inherentes a él. Se describe con más detalle el flujo de acción del negocio a partir de los diagramas de actividades y el modelo de objetos. Se identifican los principales requisitos funcionales y no funcionales con los que contará el sistema. Por último, se muestran los diagramas correspondientes al diseño y colaboración del sistema, así como también, los modelos de la base de datos.

2.1 Modelado del negocio

El modelado del negocio es una disciplina de la metodología RUP que permite comprender los procesos del negocio de una organización, con el propósito de entender su estructura, su dinámica y sus procesos, e identificar posibles mejoras (Pressman, 2010). Un buen modelado del negocio le facilita a los clientes, usuarios finales y desarrolladores, tener una idea común de la organización y derivar los requerimientos del sistema a partir del modelo de negocio que se obtenga (Vallejo et al., 2020).

2.1.1 Procesos del negocio

El modelo del negocio describe el negocio en términos de casos de usos del negocio, que corresponde a lo que generalmente se le llama procesos. Un proceso de negocio es un grupo de tareas relacionadas que se llevan a cabo en una determinada secuencia, y que emplean los recursos de la organización para dar resultados en apoyo a sus objetivos (Pressman, 2010).

Un modelo de casos de uso del negocio describe los procesos del mismo y su interacción con elementos externos, tales como socios y clientes. Es decir, describe las funciones que el negocio pretende realizar y cómo es utilizado por sus clientes.

Apoyados en estos conocimientos se identifican los siguientes procesos del negocio:

- Confeccionar Plan de Trabajo Individual

  El proceso de confección del Plan de Trabajo Individual comienza cuando, a inicios del año docente, el jefe de departamento elabora las propuestas de los planes de cada uno de los profesores del departamento, en dependencia de sus funciones

según su categoría docente. El profesor analiza las actividades planificadas para su Plan de Trabajo Individual; en caso de no estar de acuerdo con ello, ambos deben llegar a un convenio en el que se tengan en cuenta las nuevas proposiciones, siempre y cuando, se cumpla con los requisitos imprescindibles con los que debe cumplir según su categoría docente.

- Reglas del negocio

Para la confección del Plan de Trabajo Individual, el jefe de departamento debe tener en cuenta las funciones con las que debe cumplir el profesor según su categoría docente, y en dependencia de ello definir las actividades a realizar.

Tabla 1. *Descripción de CU del negocio: Confeccionar Plan de Trabajo Individual*

<table>
<tr><td colspan="2">Caso de Uso del Negocio</td><td colspan="2">CU1: Confeccionar Plan de Trabajo Individual</td></tr>
<tr><td colspan="2">Actores</td><td colspan="2">Jefe de departamento, Profesor</td></tr>
<tr><td colspan="2">Propósito</td><td colspan="2">Definir el Plan de Trabajo Individual del profesor universitario para el año docente</td></tr>
<tr><td colspan="2">Casos de usos asociados</td><td colspan="2"></td></tr>
<tr><td colspan="4">Resumen
El caso de uso inicia cuando el jefe de departamento elabora una propuesta de Plan de Trabajo Individual para cada uno de los profesores. El profesor, analiza las actividades planificadas para su plan de trabajo, y en caso de no estar de acuerdo con lo planteado, debe llegar a un convenio con su superior. El caso de uso finaliza cuando no existen diferencias por ambas partes acerca del Plan de Trabajo Individual del profesor, y este queda aprobado para su posterior cumplimiento.</td></tr>
<tr><td colspan="2">Acción del actor</td><td colspan="2">Respuesta del negocio</td></tr>
<tr><td>1.</td><td>El jefe de departamento elabora una propuesta para el Plan de Trabajo Individual de cada uno de los profesores.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>2.</td><td>El profesor analiza la propuesta de su Plan de Trabajo Individual. Está de acuerdo.</td></tr>
<tr><td>3.</td><td>Aprueba el Plan de Trabajo Individual del profesor.</td><td></td><td></td></tr>
<tr><td colspan="4">Cursos alternos
Línea 2: si el profesor no está de acuerdo con las actividades propuestas en</td></tr>
</table>

su Plan de Trabajo Individual, le solicita un cambio del mismo al jefe de departamento y ambos deben llegar a un convenio acerca de los posibles cambios a realizar.

- Evaluar Plan de Trabajo Individual

La evaluación del Plan de Trabajo Individual tiene lugar una vez finalizado el año, cuando el jefe de departamento le solicita al profesor su autoevaluación del Plan de Trabajo Individual. Teniendo en cuenta la autoevaluación del profesor, el control sistemático realizado al desarrollo de las actividades durante el año y los resultados obtenidos, el jefe de departamento le otorga una calificación al trabajo realizado por el docente. Esta evaluación es discutida con el profesor, que puede estar de acuerdo o no, se deja evidencia escrita y es archivada en su expediente docente.

- Reglas del negocio para el proceso

En el momento de efectuares la evaluación de los resultados del trabajo, el profesor debe tener laborado al menos el 70% del período que se evalúa. En caso de no cumplir este requisito tiene que esperar al próximo período evaluativo.

Tabla 2. *Descripción de CU del negocio: Evaluar Plan de Trabajo Individual*

Caso de Uso del Negocio	CU2: Evaluar Plan de Trabajo Individual
Actores	Jefe de departamento, Profesor
Propósito	Evaluar el cumplimiento del Plan de Trabajo Individual del profesor
Casos de usos asociados	Autoevaluar Plan de Trabajo Individual (include)
Resumen El caso de uso inicia cuando el jefe de departamento le solicita al profesor su autoevaluación del Plan de Trabajo Individual. El profesor realiza y envía su autoevaluación al jefe de departamento, este la analiza y procede a evaluar los resultados obtenidos en el Plan de Trabajo Individual, notificándole posteriormente los resultados de la evaluación. El caso de uso finaliza cuando no existen diferencias por ambas partes acerca de la evaluación del Plan de Trabajo Individual. El documento de evaluación queda firmado y	

archivado.			
Acción del actor		**Respuesta del negocio**	
1.	El jefe de departamento le solicita al profesor su autoevaluación del Plan de Trabajo Individual.		
		2.	Incluir CU “Autoevaluar Plan de Trabajo Individual”.
		3.	El profesor envía al jefe de departamento su autoevaluación.
4.	El jefe de departamento recibe y analiza la autoevaluación del		
5.	profesor. Procede a evaluar el cumplimiento del Plan de Trabajo		
6.	Individual del profesor. Notifica al profesor los resultados de la evaluación.		
		7.	El profesor analiza la evaluación de su Plan de Trabajo Individual. **Está de acuerdo y firma el documento de evaluación.**
8.	El jefe de departamento firma y archiva el documento de evaluación del Plan de Trabajo Individual del profesor.		
Cursos alternos **Línea 7:** si el profesor no está de acuerdo con la evaluación de su Plan de Trabajo Individual, le solicita una revisión al jefe de departamento. Este revisa nuevamente los resultados del Plan de Trabajo Individual del profesor, y en correspondencia, emite una evaluación del mismo.			

2.1.2 Actores del negocio

Un actor del negocio es cualquier individuo, grupo, entidad, organización, máquina o sistema de información externos; con los que el negocio interactúa para beneficiarse de sus resultados (Pressman, 2010).

Como anteriormente se mencionan, se identificaron como actores del negocio:

Tabla 3. *Actores del negocio*

Actor	Descripción
Jefe de departamento	Es el responsable de la organización, planificación y control del trabajo de los profesores universitarios del departamento o dirección que dirige, así como de su evaluación.
Profesor	Es el responsable de dar cumplimiento a las actividades planificadas en el Plan de Trabajo Individual.

2.1.3 Diagrama de caso de uso del negocio

Un diagrama de casos de uso del negocio representa gráficamente a los procesos del negocio y su interacción con los actores del negocio, permitiendo comprender mejor las descripciones del negocio que se está analizando.

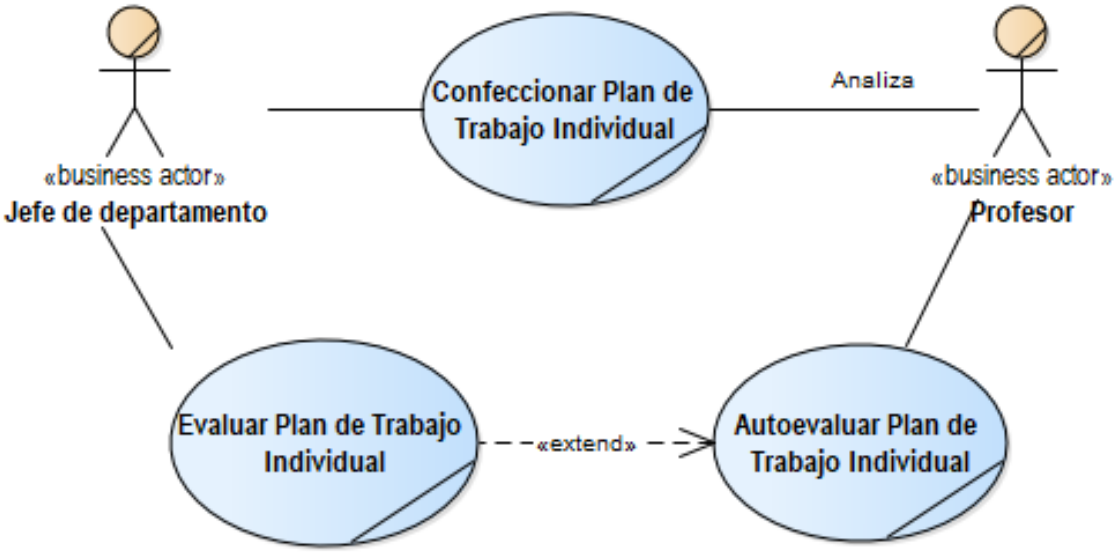
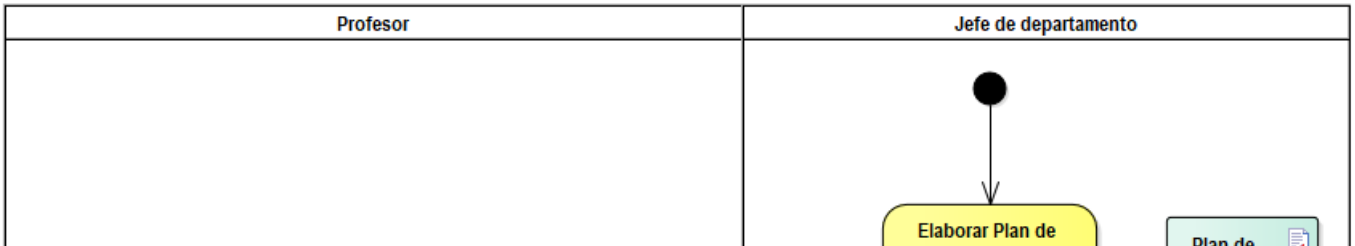
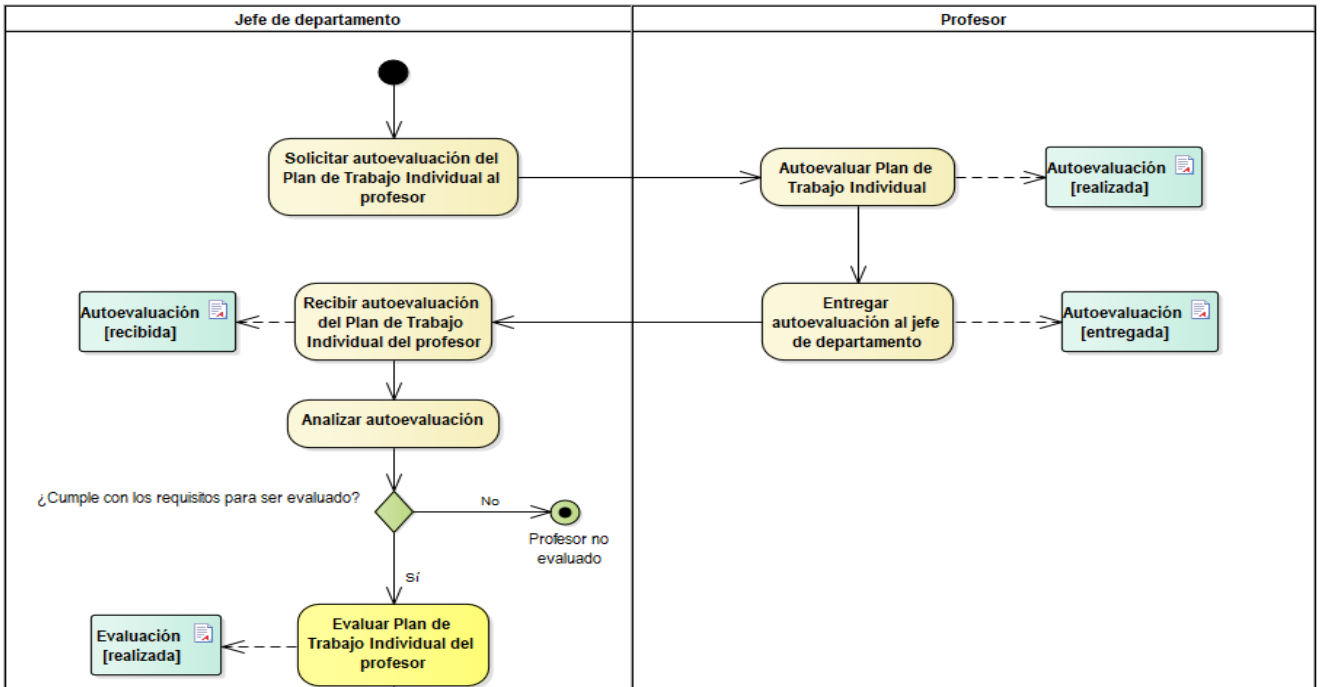


Ilustración 1. *Diagrama de CU del negocio*

2.1.

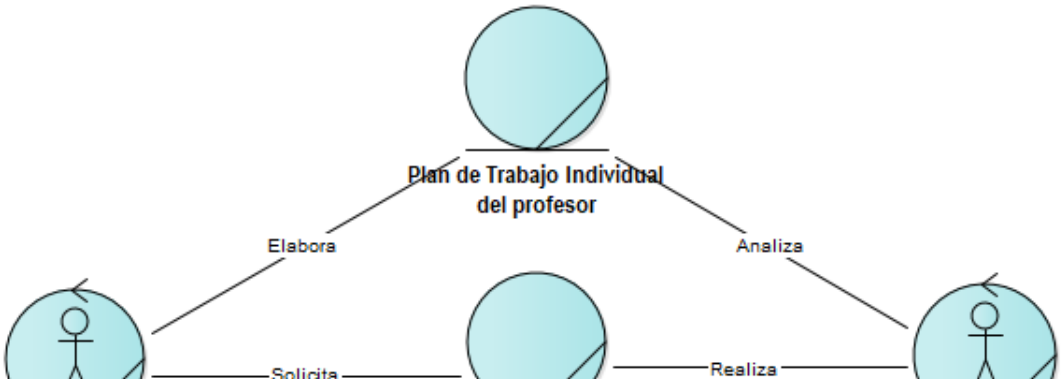
El diagrama de actividades proporciona una representación gráfica del flujo de acciones y decisiones dentro de un escenario específico (Pressman, 2010). A continuación, se muestran los procesos del negocio identificados representados en diagramas de actividades.




Jefe de departamento
Profesor
Solicitar autoevaluación del Plan de Trabajo Individual al profesor
Autoevaluar Plan de Trabajo Individual
Autoevaluación [realizada]
Autoevaluación [recibida]
Recibir autoevaluación del Plan de Trabajo Individual del profesor
Entregar autoevaluación al jefe de departamento
Autoevaluación [entregada]
Analizar autoevaluación
¿Cumple con los requisitos para ser evaluado?
No
Profesor no evaluado
Sí
Evaluación [realizada]
Evaluar Plan de Trabajo Individual del profesor

2.1.5 Modelo de objetos del negocio

El modelo de objetos del negocio describe cómo cada caso de uso del negocio es llevado a cabo por parte de un conjunto de trabajadores que utilizan un conjunto de entidades del negocio y de unidades de trabajo.



2.2 Requerimientos

Antes de desarrollar cualquier sistema software es necesario comprender qué deberá hacer y cómo dará soporte a las metas de los actores. Esta necesidad implica la comprensión del dominio de la aplicación, de las restricciones operacionales del sistema, de la funcionalidad requerida, y de las características no funcionales (Vallejo et al., 2020). A continuación, se definen los principales requisitos funcionales y no funcionales del sistema.

2.2.1 Requisitos funcionales

Para poder definir los requisitos de un sistema, es necesario entender el problema que deben resolver. La metodología RUP brinda una guía para encontrar, organizar, documentar, y seguir los cambios de los requisitos funcionales. Utiliza una notación de caso de uso para representar los requisitos, con la finalidad de especificar el

comportamiento deseado del sistema, así como describir qué debe hacer, pero no especifica cómo lo hace (Martínez & Martinez, 2000).

A continuación, se definen los principales requisitos funcionales identificados en el proyecto. Estos requisitos parten de la base de la autenticación de los usuarios y están orientados a la gestión del Plan de Trabajo Individual, sus actividades y evaluaciones.

Tabla 4. *Requisitos funcionales del sistema (RF)*

No.	Requisitos funcionales
RF01	**Autenticar usuario**
RF02	**Gestionar usuario**
2.1	Adicionar un usuario
2.2	Modificar un usuario
2.3	Eliminar un usuario
RF03	**Gestionar Plan de Trabajo Individual**
3.1	Adicionar un Plan de Trabajo Individual
3.2	Modificar un Plan de Trabajo Individual
3.3	Eliminar un Plan de Trabajo Individual
RF04	**Gestionar autoevaluación al Plan de Trabajo Individual**
4.1	Adicionar una autoevaluación al Plan de Trabajo Individual
4.2	Modificar una autoevaluación al Plan de Trabajo Individual
4.3	Eliminar una autoevaluación al Plan de Trabajo Individual
RF05	**Gestionar evaluación del Plan de Trabajo Individual**
5.1	Adicionar una evaluación al Plan de Trabajo Individual
5.2	Modificar una evaluación al Plan de Trabajo Individual
5.3	Eliminar una evaluación al Plan de Trabajo Individual
RF06	**Mostrar Plan de Trabajo Individual**
RF08	**Mostrar autoevaluación del Plan de Trabajo Individual**
RF09	**Mostrar evaluación del Plan de Trabajo Individual**

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades inherentes a este, como la fiabilidad, la respuesta en el tiempo y la capacidad de

almacenamiento (Pressman, 2010). A continuación, se mencionan los principales requisitos no funcionales a tener en cuenta para el funcionamiento correcto de la API.

- Debe garantizar el acceso a la información sin ningún inconveniente y al mismo tiempo.
- El sistema debe contar con un control de acceso que permita a cada usuario tener disponible solamente las opciones relacionadas con su rol.
- No se requiere de una capacidad de procesamiento alta, pues la aplicación no ejecutará algoritmos complejos.
- Se requiere un servidor de bases de datos con soporte de volúmenes medianos de información.
- Debe documentarse bien la aplicación para garantizar su soporte.
- Podrá ser utilizado en cualquier sistema operativo, a través de un navegador web.
- Para el servidor, se requiere de un servidor web con soporte para Node.js, y un servidor de base de datos PostgreSQL.

2.3 Modelo de casos de uso del sistema

El modelo de casos de uso del sistema permite definir las funciones del sistema para cada tipo de usuario y proporciona la entrada fundamental para el análisis, el diseño y las pruebas. A continuación, de acuerdo con lo planteado anteriormente, se describen cuáles serían los actores que van a interactuar con el sistema y los casos de uso que refieren las principales funciones del mismo.

Tabla 5. *Actores del sistema*

Actores	Descripción
Usuario	Es un actor genérico que hace uso de funcionalidades comunes para varios actores.
Administrador	Es el encargado de gestionar los usuarios que tendrán acceso al sistema.

Jefe de departamento	Es quien gestiona los Planes de Trabajo Individual de los profesores, y la evaluación del mismo.
Profesor	Es quien gestiona la autoevaluación del Plan de Trabajo Individual para su posterior valoración.

2.3.1 Diagrama de casos de uso del sistema

Un diagrama de casos de uso del sistema representa gráficamente a los procesos y su interacción con los actores.

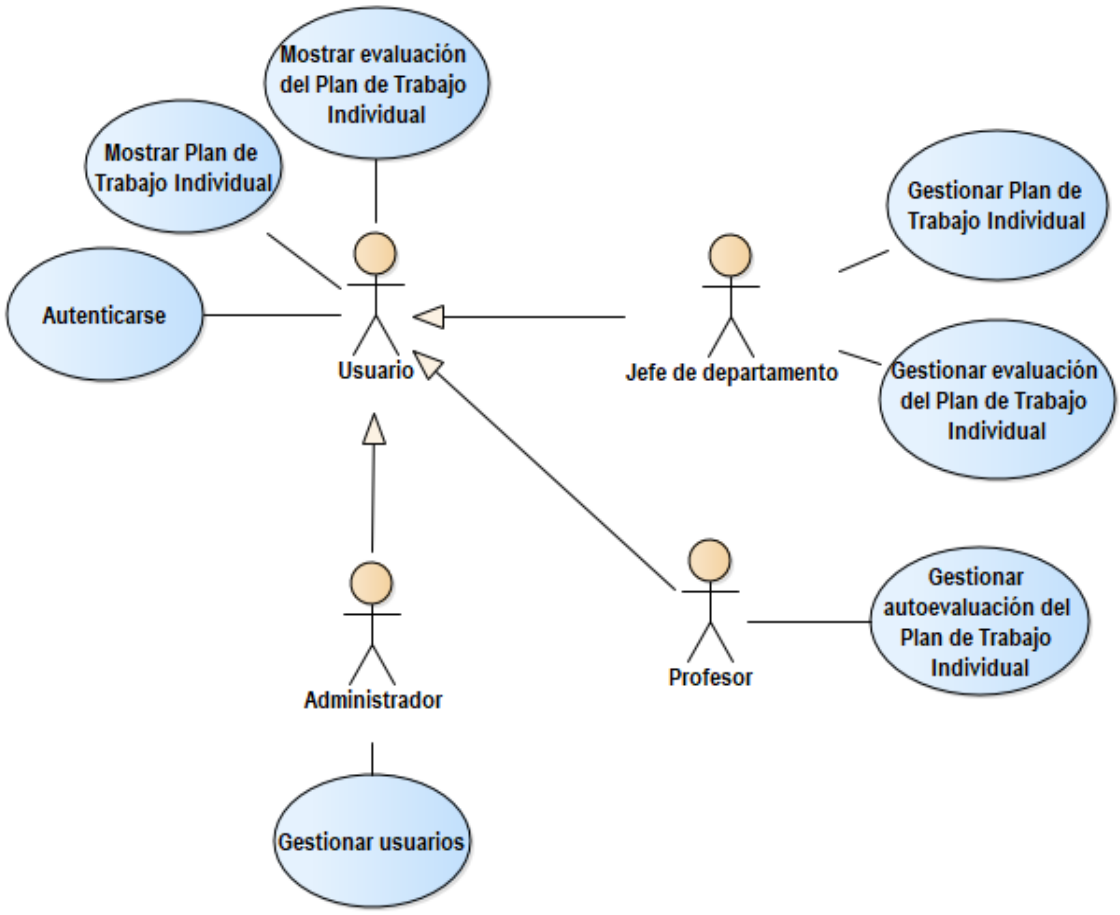


Ilustración 5. *Diagrama de casos de uso del sistema*

continuación, con más detalle, el flujo de acción entre el usuario y el sistema.

Tabla 6. *Descripción de CU del sistema: Autenticarse*

<table>
<tr><td colspan="2">Caso de uso</td><td colspan="2">Autenticarse</td></tr>
<tr><td colspan="2">Actores</td><td colspan="2">Usuario</td></tr>
<tr><td colspan="2">Propósito</td><td colspan="2">Proteger el acceso a la información del sistema.</td></tr>
<tr><td colspan="2">Resumen</td><td colspan="2">El caso de uso inicia cuando el usuario desea acceder al sistema. Para ello, introduce los datos requeridos y el sistema verifica que las credenciales sean correctas. Si los datos de entrada son válidos, el usuario podrá acceder a las opciones del sistema que le corresponden según sus permisos; de lo contrario, el sistema retornará un mensaje de error denegando el acceso.</td></tr>
<tr><td colspan="2">Referencias</td><td colspan="2">RF01</td></tr>
<tr><td colspan="2">Casos de uso asociados</td><td colspan="2"></td></tr>
<tr><td colspan="2">Precondiciones</td><td colspan="2">El sistema se encuentra disponible.</td></tr>
<tr><td colspan="2">Postcondiciones</td><td colspan="2">Usuario autenticado.</td></tr>
<tr><td colspan="4">Descripción</td></tr>
<tr><td colspan="2">Acción del actor</td><td colspan="2">Respuesta del sistema</td></tr>
<tr><td>1.
2.</td><td>El usuario accede a la URL del sistema.
Ingresa su nombre de usuario y contraseña.</td><td>

3.

4.</td><td>

El sistema verifica que le usuario esté registrado y que su contraseña esté correcta. Credenciales válidas
Permite el acceso al usuario según sus permisos.</td></tr>
<tr><td colspan="4">Cursos alternos
Línea 3: si las credenciales no son válidas, el sistema mostrará un mensaje de error.</td></tr>
<tr><td colspan="4">Observaciones
La URL para acceder a la API REST está compuesta por la dirección IP donde se encuentre alojada, el puerto de acceso y el nombre de la funcionalidad a realizar.
Ejemplo: http://localhost:3000/api/login

En caso de que los datos de entrada no sean correctos el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.</td></tr>
</table>

Tabla 7. *Descripción de CU del sistema: Gestionar Plan de Trabajo Individual*

Caso de uso		Gestionar Plan de Trabajo Individual	
Actores		Jefe de departamento	
Propósito		Adicionar, modificar o eliminar el Plan de Trabajo Individual del profesor.	
Resumen		El caso de uso inicia cuando el jefe de departamento necesita gestionar el Plan de Trabajo Individual del profesor. El sistema responderá según la petición, ya sea POST, PUT o DELETE. El caso de uso finaliza cuando haya realizado alguna de estas acciones o ninguna.	
Referencias		**RF3.1, RF3.2, RF3.3**	
Casos de uso asociados			
Precondiciones		El usuario debe estar autenticado.	
Postcondiciones			
Descripción			
Acción del actor		**Respuesta del sistema**	
1.	El jefe de departamento accede a la URL del sistema.	**2.**	El sistema ejecuta alguna de las siguientes acciones: **2.1** Si el jefe de departamento desea adicionar un nuevo Plan de Trabajo Individual, se ejecuta el FA1. **2.2** Si desea modificar los datos de un Plan de Trabajo Individual se ejecuta el FA2. **2.3** Si desea eliminar un Plan de Trabajo Individual existente se ejecuta el FA3.
Flujo alternativo		FA1. Adicionar Plan de Trabajo Individual	
1. **2.**	El jefe de departamento accede a la URL correspondiente a crear un nuevo Plan de Trabajo Individual. Ingresa los datos correspondientes al mismo.	**3.** **4.**	El sistema valida los datos de entrada. **Datos correctos** El sistema almacena los datos del plan en la base de datos y muestra un mensaje de información: “Plan

			de Trabajo Individual añadido correctamente".
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso y el nombre del recurso y funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/crear En caso de que los datos de entrada no sean correctos el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			
Flujo alternativo		FA2. Modificar Plan de Trabajo Individual	
1.	El jefe de departamento accede a la URL correspondiente a modificar un Plan de Trabajo Individual.		
2.	Ingresa los datos a modificar.		
		3.	El sistema verifica si el plan está almacenado en la base de datos.
		4.	**Datos correctos** El sistema valida los datos de entrada. **Datos correctos**
		5.	El sistema actualiza los datos del plan en la base de datos y muestra un mensaje de información: "Plan de Trabajo Individual actualizado correctamente".
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id del plan a modificar y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/modificar En caso de que los datos de entrada no sean correctos o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			
Flujo alternativo		FA3. Eliminar Plan de Trabajo Individual	
1.	El jefe de departamento accede a la URL correspondiente a eliminar un Plan de Trabajo Individual.		
		2.	El sistema verifica si el plan está almacenado en la base de datos. **Datos correctos**
		3.	El sistema elimina los datos del plan de la base de datos y muestra un mensaje de información: "Plan

			de Trabajo Individual eliminado correctamente".
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id del plan a eliminar y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/eliminar En caso de que los datos de entrada no sean correctos o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			

Tabla 8. *Descripción de CU del sistema: Gestionar autoevaluación del Plan de Trabajo Individual*

Caso de uso		Gestionar autoevaluación Plan de Trabajo Individual	
Actores		Profesor	
Propósito		Adicionar, mostrar, modificar o eliminar autoevaluación del Plan de Trabajo Individual.	
Resumen		El caso de uso inicia cuando el profesor necesita gestionar la autoevaluación del Plan de Trabajo Individual. El sistema responderá según la petición, ya sea POST, PUT o DELETE. El caso de uso finaliza cuando haya realizado alguna de estas acciones o ninguna.	
Referencias		**RF4.1, RF4.2, RF4.3**	
Casos de uso asociados			
Precondiciones		El usuario debe estar autenticado.	
Postcondiciones			
Descripción			
Acción del actor		**Respuesta del sistema**	
1.	El profesor accede a la URL del sistema.	**2.**	El sistema ejecuta alguna de las siguientes acciones: **2.1** Si el profesor desea adicionar una nueva autoevaluación del Plan de Trabajo Individual, se ejecuta el FA1. **2.2** Si desea modificar los datos de una autoevaluación del Plan de

			Trabajo Individual se ejecuta el FA2. **2.3** Si desea eliminar una autoevaluación del Plan de Trabajo Individual existente se ejecuta el FA3.
Flujo alternativo		FA1. Adicionar autoevaluación al Plan de Trabajo Individual	
1.	El profesor accede a la URL correspondiente a crear una nueva autoevaluación del Plan de Trabajo Individual.		
2.	Ingresa los datos correspondientes a la misma.		
		3.	El sistema verifica si el plan está almacenado en la base de datos. **Datos correctos**
		4.	El sistema valida los datos de entrada. **Datos correctos**
		5.	El sistema almacena los datos de la autoevaluación en la base de datos y muestra un mensaje de información: “Autoevaluación del Plan de Trabajo Individual añadida correctamente”.
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso y el nombre del recurso, el id del plan al que se le añadirá la autoevaluación y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/autoevaluacion/crear En caso de que los datos de entrada no sean correctos o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			
Flujo alternativo		FA2. Modificar autoevaluación del Plan de Trabajo Individual	
1.	El profesor accede a la URL correspondiente a modificar una autoevaluación del Plan de Trabajo Individual.		
2.	Ingresa los datos a modificar.		
		3.	El sistema verifica si la autoevaluación y el plan está almacenado en la base de datos. **Datos correctos**
		4.	El sistema valida los datos de

			entrada. **Datos correctos** **5.** El sistema actualiza los datos de la autoevaluación del plan en la base de datos y muestra un mensaje de información: “Autoevaluación del Plan de Trabajo Individual actualizada correctamente”.
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id del plan a modificar y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/autoevaluacion/:autoevaluacionId/modificar En caso de que los datos de entrada no sean correctos o la autoevaluación o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			
Flujo alternativo			FA3. Eliminar autoevaluación del Plan de Trabajo Individual
1.	El profesor accede a la URL correspondiente a eliminar una autoevaluación del Plan de Trabajo Individual.	**2.** **3.**	El sistema verifica si la autoevaluación y el plan está almacenado en la base de datos. **Datos correctos** El sistema elimina los datos de la autoevaluación del plan de la base de datos y muestra un mensaje de información: “Autoevaluación del Plan de Trabajo Individual eliminada correctamente”.
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id del plan a eliminar y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/autoevaluacion/:autoevaluacionId/eliminar En caso de que los datos de entrada no sean correctos o la autoevaluación o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			

Tabla 9. *Descripción del CU del sistema: Gestionar evaluación del Plan de Trabajo Individual*

Caso de uso		Gestionar evaluación del Plan de Trabajo Individual	
Actores		Jefe de departamento	
Propósito		Adicionar, modificar o eliminar evaluación del Plan de Trabajo Individual del profesor.	
Resumen		El caso de uso inicia cuando el jefe de departamento necesita gestionar la evaluación del Plan de Trabajo Individual del profesor. El sistema responderá según la petición, ya sea POST, PUT o DELETE. El caso de uso finaliza cuando haya realizado alguna de estas acciones o ninguna.	
Referencias		**RF5.1, RF5.2, RF5.3**	
Casos de uso asociados			
Precondiciones		El usuario debe estar autenticado.	
Postcondiciones			
Descripción			
Acción del actor		**Respuesta del sistema**	
1.	El jefe de departamento accede a la URL del sistema.	**2.**	El sistema ejecuta alguna de las siguientes acciones: **2.1** Si el jefe de departamento desea adicionar una nueva evaluación del Plan de Trabajo Individual, se ejecuta el FA1. **2.2** Si desea modificar los datos de una evaluación del Plan de Trabajo Individual se ejecuta el FA2. **2.3** Si desea eliminar una evaluación del Plan de Trabajo Individual existente se ejecuta el FA3.
Flujo alternativo		FA1. Adicionar evaluación del Plan de Trabajo Individual	
1.	El jefe de departamento accede a la URL correspondiente a crear una nueva evaluación del Plan de Trabajo Individual.		
2.	Ingresa los datos correspondientes		

	a la misma.	**3.** **4.** **5.**	El sistema verifica si el plan está almacenado en la base de datos. **Datos correctos** El sistema valida los datos de entrada. **Datos correctos** El sistema almacena los datos de la evaluación del plan en la base de datos y muestra un mensaje de información: “Evaluación del Plan de Trabajo Individual añadida correctamente”.
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso y el nombre del recurso, junto con el id del plan al que se añadirá la evaluación y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/:planId/evaluacion/crear En caso de que los datos de entrada no sean correctos o el plan no se encuentre almacenado en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.			
Flujo alternativo		FA2. Modificar evaluación del Plan de Trabajo Individual	
1. **2.**	El jefe de departamento accede a la URL correspondiente a modificar una evaluación del Plan de Trabajo Individual. Ingresa los datos a modificar.	**3.** **4.** **5.**	El sistema verifica si la evaluación está almacenada en la base de datos. **Datos correctos** El sistema valida los datos de entrada. **Datos correctos** El sistema actualiza los datos de la evaluación del plan en la base de datos y muestra un mensaje de información: “Evaluación del Plan de Trabajo Individual actualizada correctamente”.
Observaciones La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id de la evaluación a modificar y la funcionalidad a realizar. **Ejemplo:** http://localhost:3000/api/plan/evaluacion/:evaluacionId/modificar En caso de que los datos de entrada no sean correctos o la evaluación no se			

<table>
<tr><td colspan="4">encuentre almacenada en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.</td></tr>
<tr><td colspan="2">Flujo alternativo</td><td colspan="2">FA3. Eliminar evaluación del Plan de Trabajo Individual</td></tr>
<tr><td>1.</td><td>El jefe de departamento accede a la URL correspondiente a eliminar una evaluación del Plan de Trabajo Individual.</td><td>2.
3.</td><td>El sistema verifica si la evaluación está almacenada en la base de datos. Datos correctos
El sistema elimina los datos de la evaluación del plan, de la base de datos y muestra un mensaje de información: “Evaluación del Plan de Trabajo Individual eliminada correctamente”.</td></tr>
<tr><td colspan="4">Observaciones
La URL para acceder a la API REST está compuesta por la dirección IP o DNS donde se encuentre alojada, el puerto de acceso, el nombre del recurso junto con el id de la evaluación a eliminar y la funcionalidad a realizar.
Ejemplo: http://localhost:3000/api/evaluacion/:evaluacionId/eliminar

En caso de que los datos de entrada no sean correctos o la evaluación no se encuentre almacenada en la base de datos, el sistema no ejecutará la acción y mostrará un mensaje de información correspondiente.</td></tr>
</table>

2.4 Análisis y diseño

El análisis consiste en obtener una visión del sistema, que se preocupa de ver qué hace, de modo que solo se interesa por los requisitos funcionales. Por otro lado, el diseño es un refinamiento del análisis, que tiene en cuenta los requisitos no funcionales, en definitiva, cómo cumple el sistema sus objetivos (Pressman, 2010).

2.4.1 Diagramas de colaboración

La colaboración es el conjunto de acciones y sistemas que tienen conexiones entre sí y trabajan juntos para realizar cualquier tarea. Los diagramas de colaboración describen cómo se comunican los objetos para ejecutar un flujo específico de eventos en un caso

de uso. Se crean identificando los elementos estructurales necesarios para llevar a cabo la funcionalidad de una interacción. A continuación, se muestran los diagramas de colaboración del caso de uso “Gestionar Plan de Trabajo Individual”.

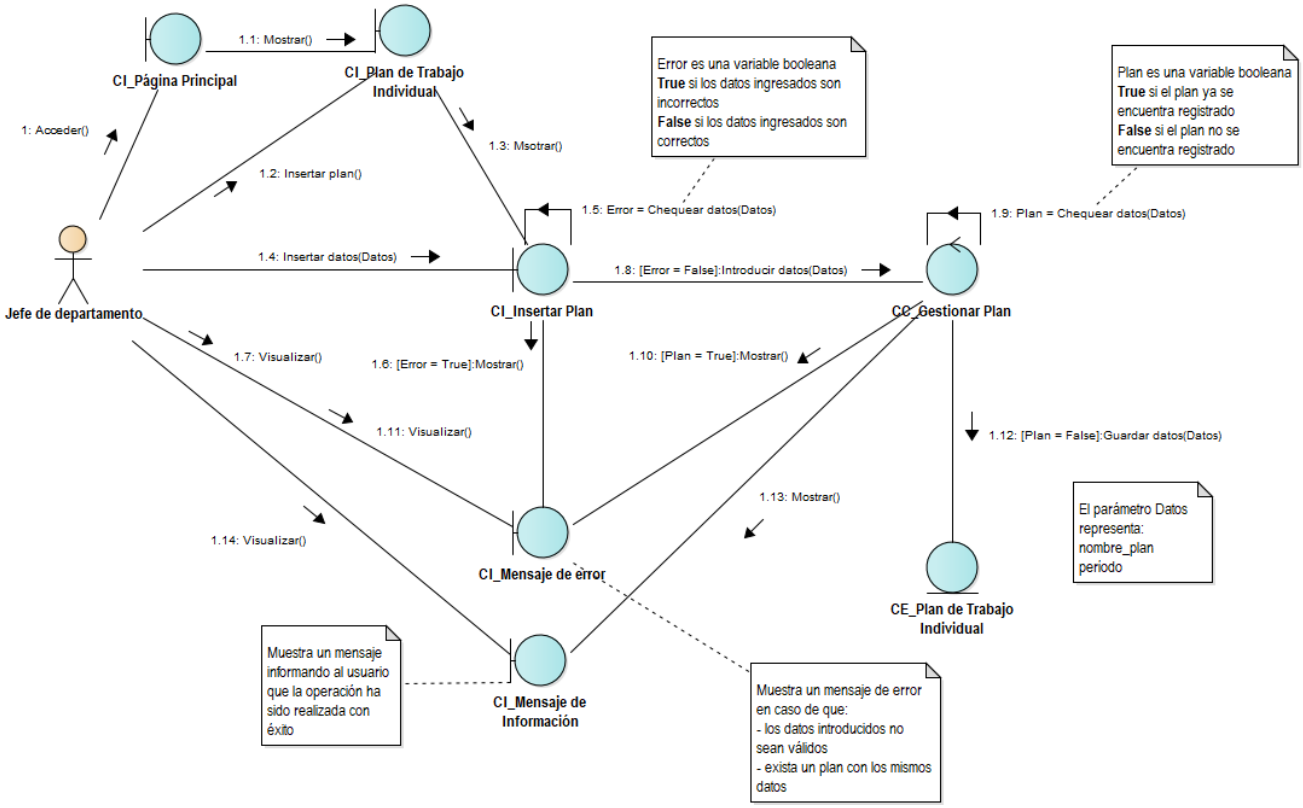
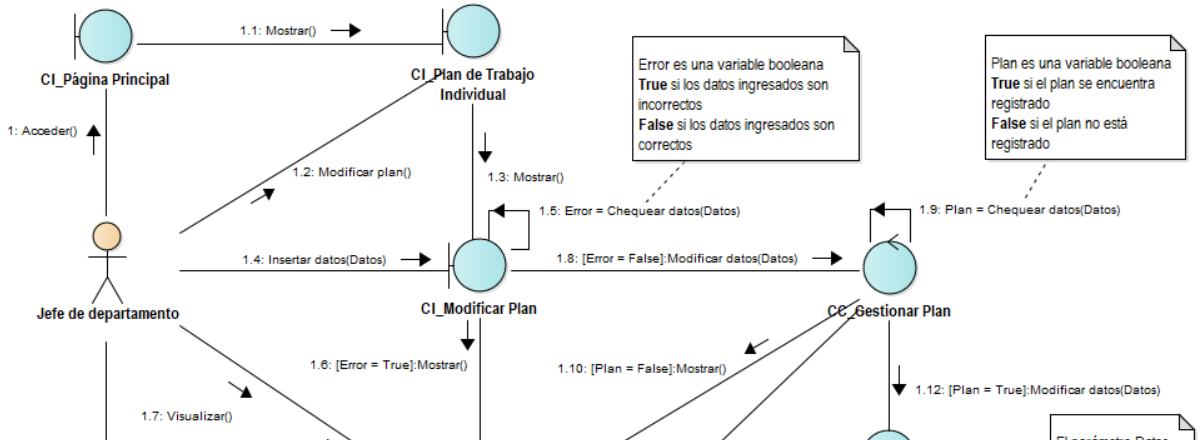


Ilustración 6. *Diagrama de colaboración. Insertar Plan de Trabajo Individual*



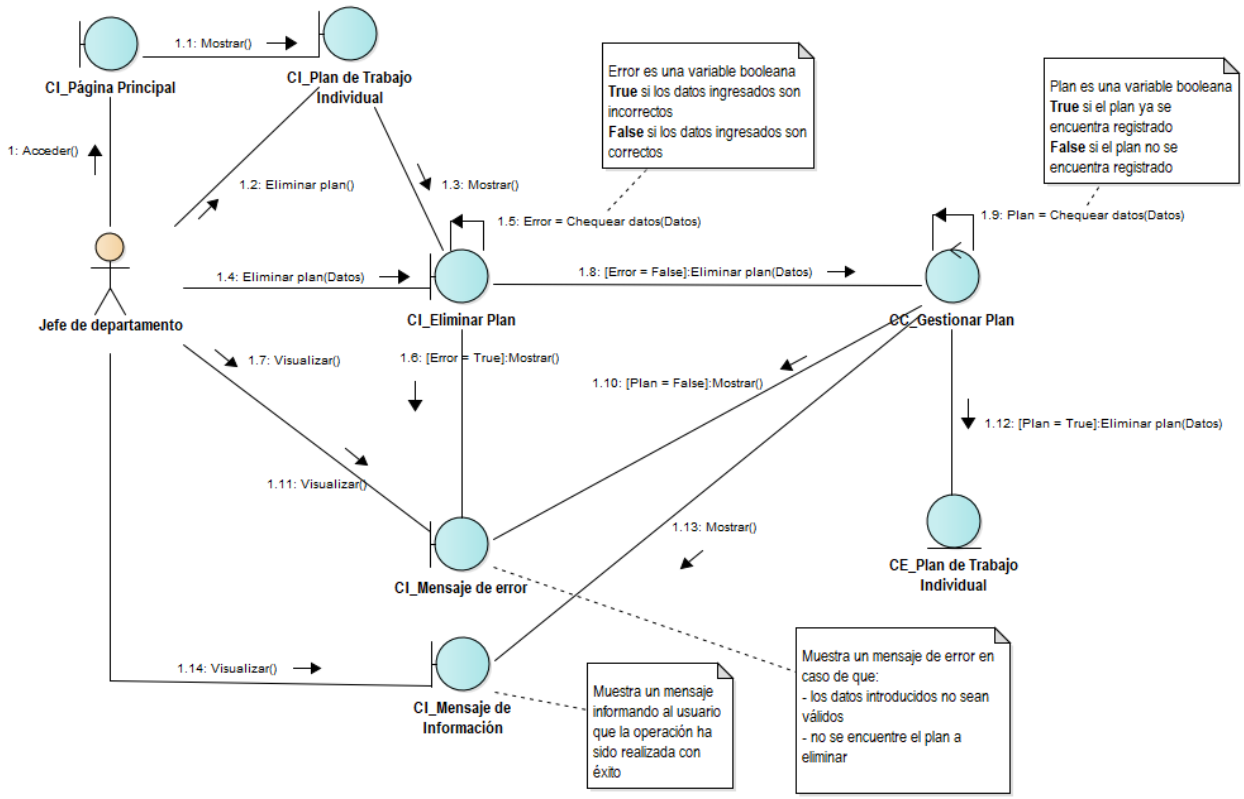
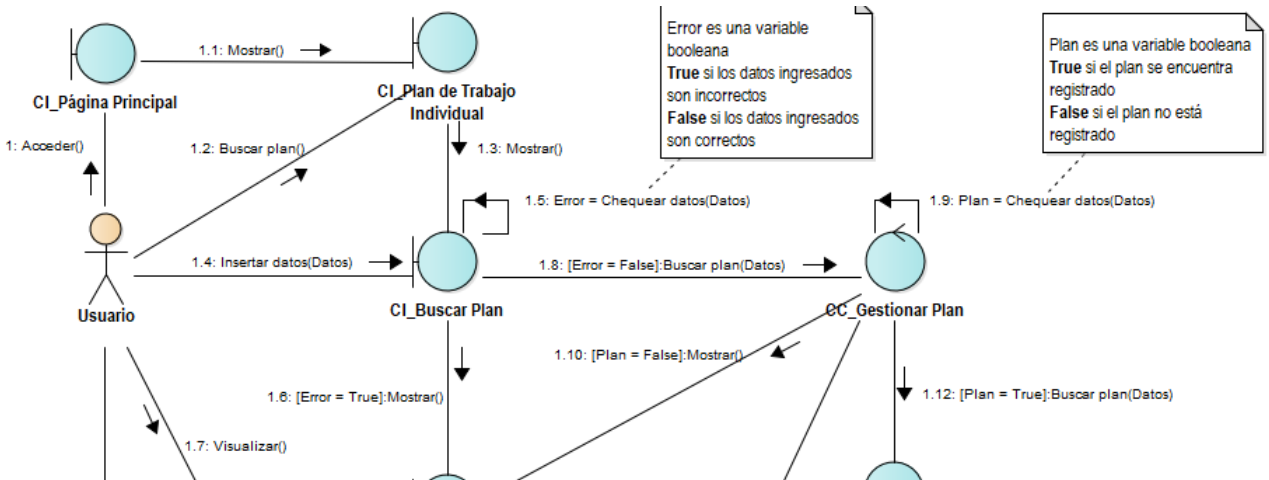
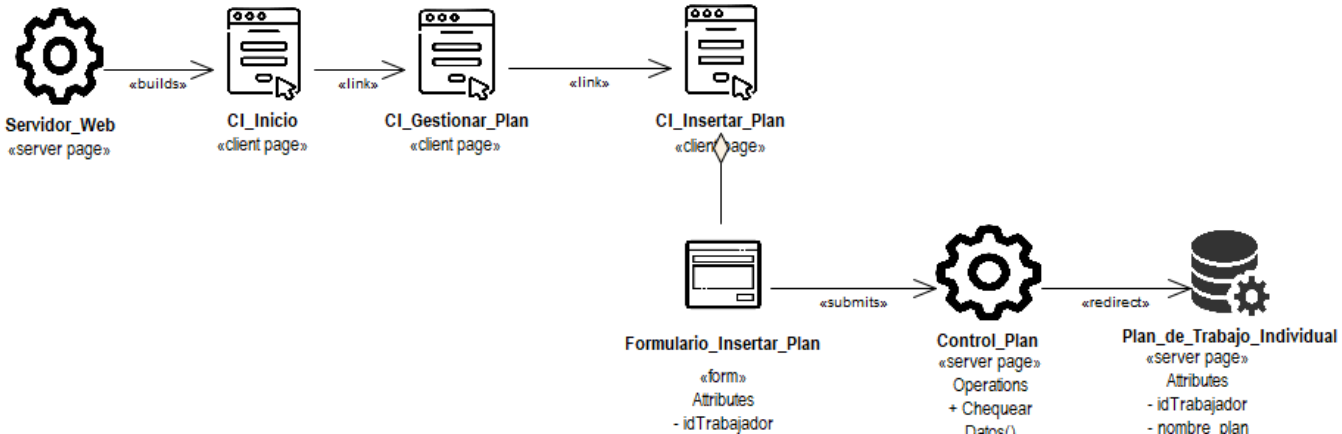


Ilustración 8. *Diagrama de colaboración. Eliminar Plan de Trabajo Individual*



2.4.2 Diagramas de clases de diseño

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación, así como la relación entre ambas. A continuación, se muestran los diagramas de clases de diseño desde la perspectiva de una web que implemente la API REST.



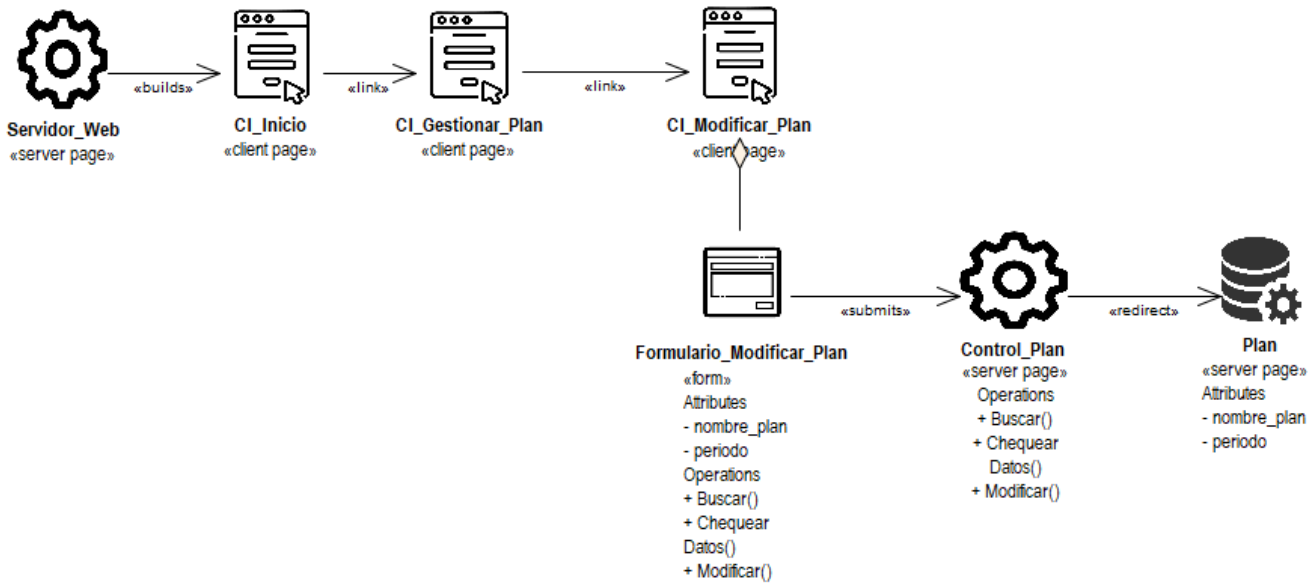
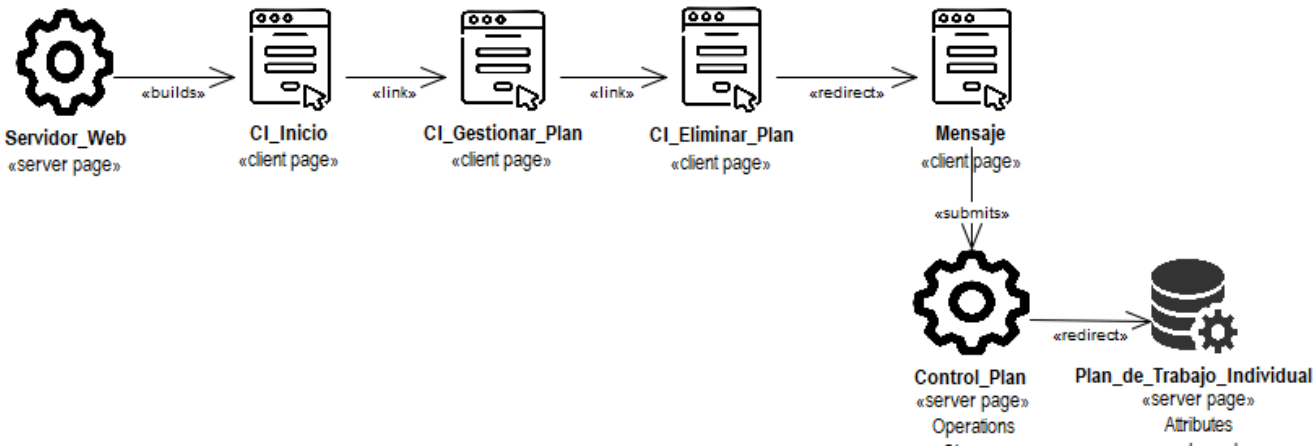


Ilustración 11. *Diagrama de clase de diseño. Modificar Plan de Trabajo Individual*



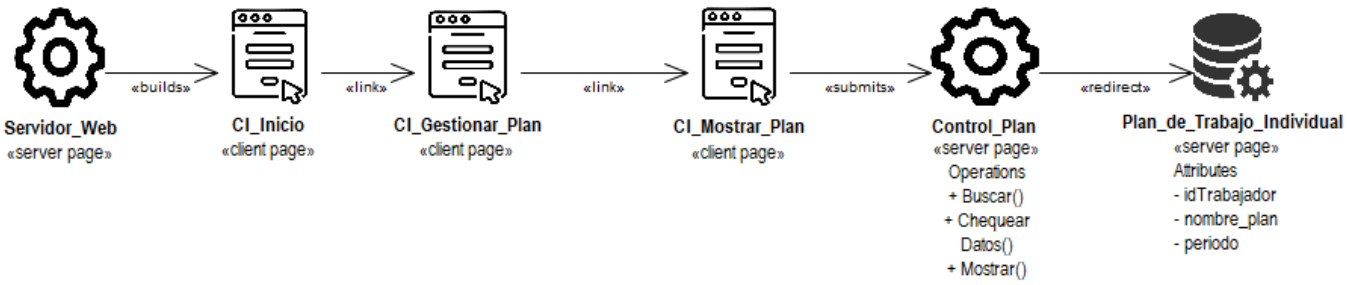
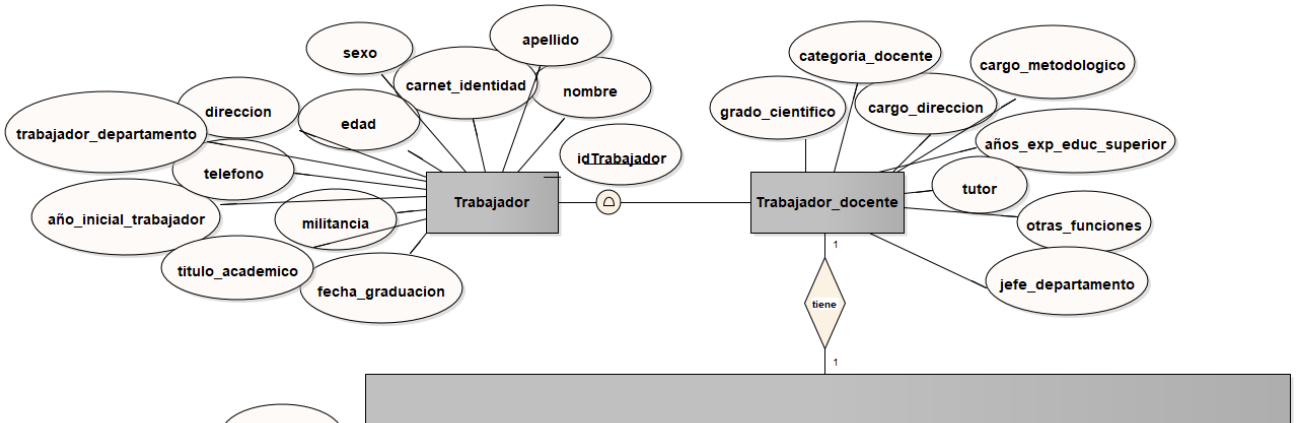


Ilustración 13. *Diagrama de clase de diseño. Mostrar Plan de Trabajo Individual*

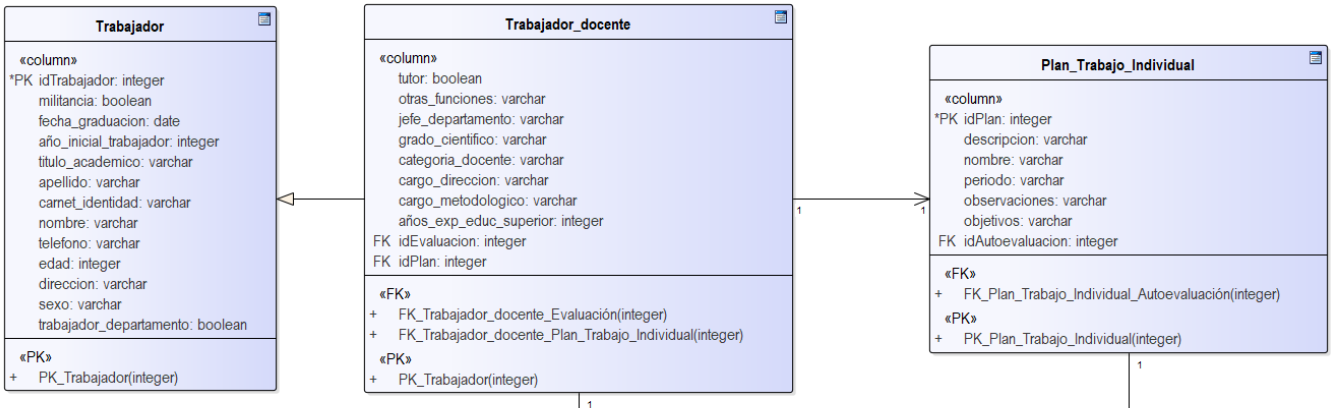
2.5 Modelo de datos

El modelo de datos describe la representación lógica y física de los datos usados por la aplicación. Se define como la estructuración y ordenamiento de un conjunto de datos para que puedan ser procesados correctamente y sin mucha dificultad por una base de datos. A continuación, se define el modelo entidad relación y el modelo físico de la base de datos del sistema propuesto.

2.5.1 Diagrama de Entidad-Relación de la base de datos



2.5.2 Modelo físico de la base de datos



2.6 Conclusiones parciales

En este capítulo, apoyados en la metodología RUP, se identificaron los principales procesos de negocio, los actores que intervienen, y las reglas del negocio inherentes a los mismos. Se logró definir las funciones a cumplir por el sistema al identificar los requisitos funcionales y no funcionales, y la relación de los actores con cada funcionalidad a implementar. Se diseñó también, el modelo de base de datos del sistema.

CAPÍTULO 3: IMPLEMENTACIÓN DE UNA API QUE CONTRIBUYA A LA GESTIÓN DEL PLAN DE TRABAJO INDIVIDUAL DEL PROFESOR UNIVERSITARIO

En este capítulo se especifica el proceso de desarrollo de la API REST, teniendo en cuenta temas de seguridad, tratamiento de errores y documentación. Se modelan los diagramas de componentes y de despliegue, y, por último, se realizan las pruebas de integración para verificar el correcto funcionamiento de la API.

3.1 Seguridad, documentación y tratamiento de errores

3.1.1 Seguridad

La seguridad de las API es un componente central de la seguridad de aplicaciones web. La mayoría de las aplicaciones web modernas se basan en las API para funcionar, y las API introducen un riesgo adicional en una aplicación al permitir que partes externas accedan a ella. Para garantizar la seguridad de la API se utilizó JSON Web Token.

Los tokens web JSON son una forma abierta y estándar de representar la identidad de su usuario de forma segura durante una interacción entre dos partes. Cuando dos sistemas intercambian datos, pueden usar un token web JSON para identificar al usuario sin tener que enviar credenciales privadas en cada solicitud.

La aplicación de usuario o cliente envía una solicitud de inicio de sesión. Esencialmente, un nombre de usuario, una contraseña o cualquier otro tipo de credenciales de inicio de sesión que el usuario proporcione. Una vez verificada, la API creará un token web JSON y lo firmará con una clave secreta. Luego, la API devolverá ese token a la aplicación cliente. Finalmente, la aplicación cliente recibirá el token, lo verificará en su propio lado para asegurarse de que sea auténtico y luego lo usará en

cada solicitud posterior. Por lo tanto, puede autenticar al usuario sin que este tenga que enviar más sus credenciales.

Por otra parte, para la autorización, el usuario contará con un atributo rol del que dependerá su acceso a ciertas acciones en la API.

3.1.2 Documentación

Una documentación bien estructurada facilita el proceso de aprendizaje sobre el nuevo entorno que se desarrollará basándose en la API. La documentación es una de las piezas básicas que determinan la experiencia de un desarrollador al hacer uso de las API, por lo que debe ser interactiva, mostrando la estructura y la organización que le facilitará el trabajo, de modo que no tenga que pasar tiempo investigando las funciones o perderlo en pruebas totalmente improductivas.

Existen diversas herramientas, una de las más conocidas es Swagger. Esta herramienta es extremadamente útil para describir, producir, consumir y visualizar API. El principal objetivo es unificar el sistema de documentación API con el propio código desarrollado, para que esté sincronizado en cada cambio. Como resultado final proporciona una interfaz web donde se pueden testear los endpoints API a la vez que se está consultando la documentación, como se puede apreciar en la siguiente imagen.

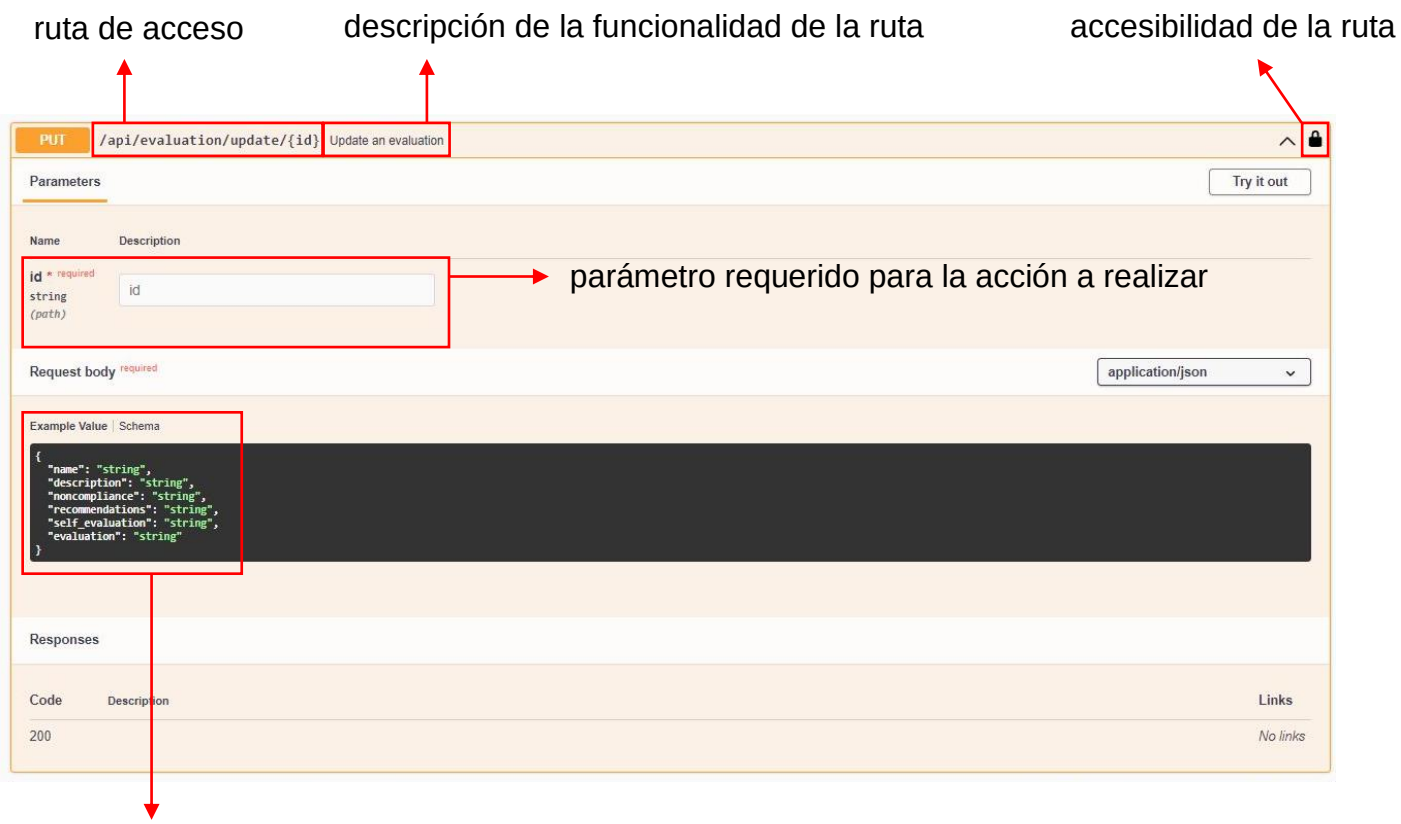


Ilustración 16. *Documentación de la API con Swagger*

3.1.3 Tratamiento de errores

Es una buena práctica validar la exactitud de los datos enviados a una aplicación. Para validar automáticamente las solicitudes entrantes, NestJS proporciona varias herramientas para esta función, entre ellas, ValidationPipe.

ValidationPipe proporciona un enfoque conveniente para hacer cumplir las reglas de validación para la entrada de datos, donde las reglas de validación se declaran con decoradores del paquete class-validator. Estas reglas pueden determinar que un campo no esté vacío, o que sea un tipo de datos específico, entre otras funciones. En

caso de que no se cumplan estas reglas, se mostrará un mensaje correspondiente, como se muestra en la siguiente imagen.

3.2 Implementación

3.2.1 Diagrama de componentes

Los componentes son partes modulares de un sistema, sean estos códigos fuente, binarios o ejecutables (Bernstein & Booch, 2020). Los diagramas de componentes representan las relaciones entre los componentes del sistema mediante una vista de diseño. Las relaciones indican que un componente utiliza los servicios ofrecidos por otro componente (Fernández & Cadelli, 2014).

A continuación, se muestra el diagrama de componentes para la propuesta de solución del proyecto.

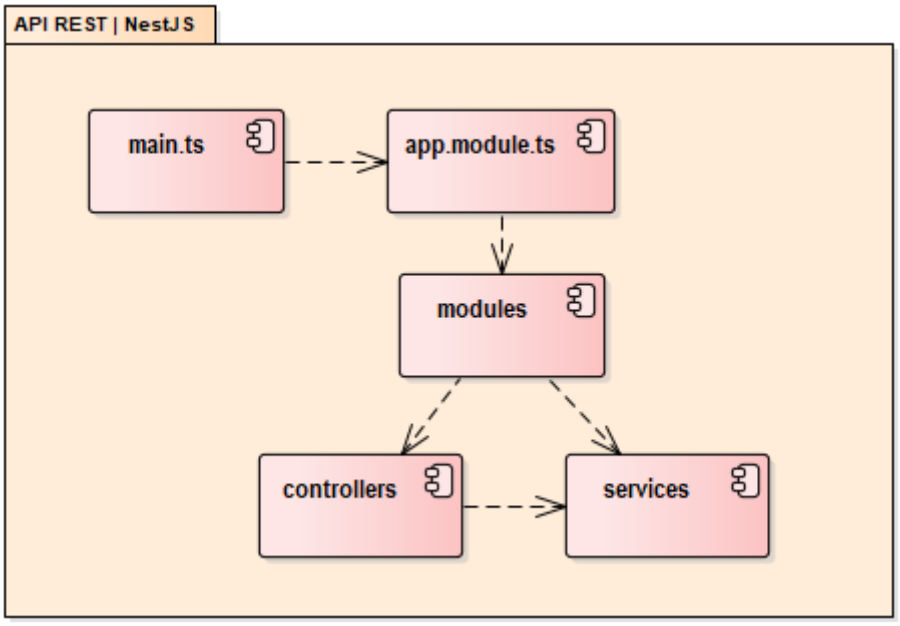


Ilustración 17. *Diagrama de componentes*

Descripción de los componentes

main.ts: es el archivo de entrada de la aplicación que utiliza la función central NestFactory para crear una instancia de la aplicación Nest.

app.module.ts: es el módulo raíz de la aplicación, que conecta con los demás módulos y con la base de datos.

modules: proporciona metadatos que Nest utiliza para organizar la estructura de la aplicación.

controllers: son responsables de manejar las solicitudes entrantes y devolver las respuestas al cliente.

services: contiene toda la lógica del negocio, por ejemplo, las operaciones CRUD, que son llamadas por los controllers para devolver una respuesta al cliente.

3.2.2 Diagrama de despliegue

Los diagramas de despliegue permiten modelar la disposición física o topología de un sistema; muestran las relaciones de los distintos nodos que lo componen y el reparto de los componentes sobre dichos nodos (Sommerville, 2005).

A continuación, se muestra el diagrama de despliegue para la propuesta de solución del proyecto.

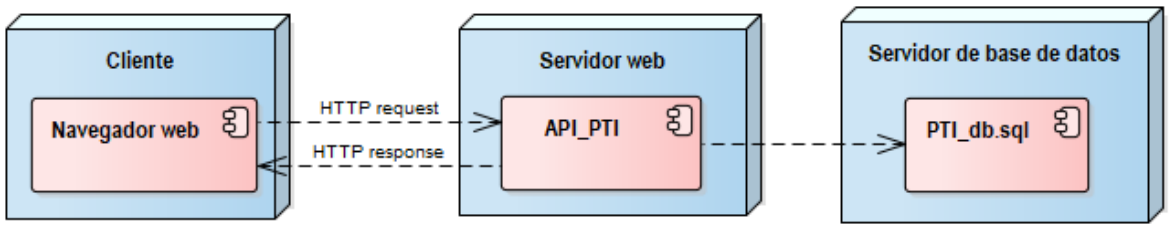


Ilustración 18. *Modelo de despliegue*

Descripción de los nodos

Cliente: referido a PC, laptop o dispositivo móvil que actuará como cliente y se comunicará mediante un navegador web con la API.

Servidor web: constituye el servidor donde estará alojada la API.

Servidor de base de datos: contendrá la información almacenada en la base de datos.

3.3 Pruebas

Las pruebas de software son un conjunto de actividades que se realizan con el objetivo de detectar fallos y evaluar las características del software; regulan la ejecución de los proyectos y garantizan la calidad del producto desarrollado. Incluyen técnicas y métodos específicos del diseño de casos de prueba (Sommerville, 2005).

Para la API REST del proyecto, desarrollado bajo los procedimientos y técnicas definidos por la metodología RUP, se establece como escenario de prueba la aplicación de pruebas de integración, con el objetivo de demostrar que el correcto funcionamiento de las acciones de la API en conjunto.

3.3.1 Pruebas de integración

Estas pruebas permiten comprobar la integración de los controladores con los servicios, la integración de nuestro sistema con algo de infraestructura (bases de datos, archivos, E/S). Pueden hacer llamadas a servicios externos y normalmente cubren la prueba de un sistema aislándolo del resto (Sánchez Salas, 2022). NestJS proporciona integración con Jest y Supertest listos para usar en pruebas unitarias, pruebas de integración y e2e.

3.4 Conclusiones parciales

En este capítulo se describió cómo se realizó el tratamiento de los errores y seguridad del sistema, y se diseñaron los diagramas de componentes y despliegue para facilitar la comprensión de las relaciones entre los componentes de software y hardware. Finalmente se emplearon las pruebas de integración para comprobar el correcto flujo de trabajo de la solución.

CONCLUSIONES GENERALES

Al finalizar la investigación se arribó a las siguientes conclusiones:

- Mediante el estudio de los fundamentos teóricos, metodológicos y tecnológicos se determinó para la documentación de desarrollo del software, el uso de la metodología RUP. Se seleccionaron como tecnologías para la implementación de la API, el framework NestJS que provee una estructura bien organizada para el desarrollo de software; y como gestor de base de datos PostgreSQL.
- Se diseñó una API para contribuir a la gestión del Plan de Trabajo Individual de los profesores de la Universidad de Sancti Spíritus “José Martí Pérez” mediante el estudio y modelado del negocio, la definición los principales requerimientos funcionales, el flujo de acciones y el modelo de base de datos.
- Se implementó una API para la gestión del Plan de Trabajo Individual del profesor universitario atendiendo a las necesidades del cliente, el tratamiento de errores y el control de acceso a la misma. Las pruebas y técnicas aplicadas al producto de software final permitieron evaluar el correcto funcionamiento del sistema.

RECOMENDACIONES

Considerando los resultados derivados de todo el proceso de la investigación y el desarrollo del sistema se recomienda:

- Desarrollar una solución que permita definir el Plan de Trabajo Individual del profesor universitario en dependencia de la categoría docente que ostente.
- Desarrollar el frontend para la API empleando marcos de trabajo modernos.

REFERENCIAS BIBLIOGRÁFICAS

Abreu, J. M., Rosales, L. C. de L., Herrera, A. L. G., & Pérez-Carrion, N. B. (2018). Desarrollo de la informatización en la Universidad de Ciencias Médicas de Matanzas. *Revista Médica Electrónica*, *40*(6), Art. 6.

Aguilera Martínez, D. (2019). *Servicio web para la gestión de información sobre bases de datos* [BachelorThesis]. https://e-archivo.uc3m.es/handle/10016/30296

Ahmed, S., & Mahmood, Q. (2019). An authentication based scheme for applications using JSON web token. *2019 22nd International Multitopic Conference (INMIC)*, 1-6. https://doi.org/10.1109/INMIC48123.2019.9022766

Alcibar, M. F., Monroy, A., & Jiménez, M. (2018). Impacto y Aprovechamiento de las Tecnologías de la Información y las Comunicaciones en la Educación Superior. *Información tecnológica*, *29*(5), 101-110. https://doi.org/10.4067/S0718-07642018000500101

Bautista, M., Lujo Aliaga, Z., Cedeño Galindo, L. V., Sosa Rivero, L. A., Pérez Céspedes, A., & Megna Alicio, A. (2018). Propuesta de sistema informático para la gestión del plan de trabajo individual de los profesores. *Revista de Investigación en Tecnologías de la Información: RITI*, *6*(11), 67-71.

Bernstein, D., & Booch, G. (2020). The UML and the Rational Unified Process. *IEEE Software*, *37*(06), 12-12. https://doi.org/10.1109/MS.2020.3019539

Cabrera Sarmiento, D. G. (2021). Análisis, diseño y desarrollo de una aplicación móvil para fomentar el estudio en internos y residentes de las facultades de medicina del Perú. *Universidad de Piura*. https://pirhua.udep.edu.pe/handle/11042/5268

Caqui Vargas, J. G., & Pareja Limaco, J. P. (2018). *Estudio comparativo entre las bases de datos relacional y no relacionales: Una revisión de la literatura científica* [Trabajo de investigación, Universidad Privada del Norte]. https://hdl.handle.net/11537/27388

Castanedo Abay, A. (2019). Modelo conceptual descriptivo para ejecutar una eficaz gestión por procesos, con garantía de calidad, en la Universidad del siglo xxi. *Revista Cubana de Educación Superior*, *38*(2). http://scielo.sld.cu/scielo.php?script=sci_abstract&pid=S0257-43142019000200011&lng=es&nrm=iso&tlng=pt

Coral Quinto, M. J. (2018). *Diseño e implementación de base de datos mediante el uso de web services con integración de Unity 3D para apoyo de aplicaciones lúdicas en la materia de Fundamentos de Programación.* [Thesis, Universidad de Guayaquil. Facultad de Ingeniería Industrial. Carrera de Licenciatura en Sistemas de Información.]. http://repositorio.ug.edu.ec/handle/redug/36403

De, B. (2017). API Documentation. En B. De (Ed.), *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization* (pp. 59-80). Apress. https://doi.org/10.1007/978-1-4842-1305-6_4

del Pino Sarduy, J. A., & Fernández Álvarez, D. (2021). GPLAN: Sistema Informático para la gestión de los Planes de Desarrollo Individual. *Sociedad & Tecnología*, *4*(1), Art. 1. https://doi.org/10.51247/st.v4i1.72

Demashov, D., & Gosudarev, I. (2019). *Efficiency Evaluation of Node.js Web-Server Frameworks*. 8.

Fernández Fontao, I. (2018). *Simplificación de la gestión de red mediante el uso de APIs* [Bachelor thesis, Universitat Politècnica de Catalunya]. https://upcommons.upc.edu/handle/2117/125739

Fernández, J. M., & Cadelli, S. (2014). *Convivencia de metodologías: Scrum y Rup en un proyecto de gran escala* [Tesis de grado, Universidad Nacional de La Plata]. http://sedici.unlp.edu.ar/handle/10915/47082

González Reyes, A., & Febles Estrada, A. (2021). Principales áreas de impacto y resultados de la informatización de la sociedad cubana. *UCIENCIA 2021*. https://repositorio.uci.cu/jspui/handle/123456789/9866

Haro, E., Guarda, T., Zambrano Peñaherrera, A. O., & Ninahualpa Quiña, G. (2019). *Desarrollo backend para aplicaciones web, Servicios Web Restful: Node.js vs Spring Boot*. *E17*, 309-321.

Hernández Pozo, J., Escobedo Nicot, M., & Calderín Calzadilla, I. (2022). SisCO: sistema de información para el servicio de guardia en la Universidad de Oriente. *Revista Universidad y Sociedad*, *14 (S1)*, 487-499.

Jhones, A. R., & Larramendi, J. V. (2019). La Informatización de las Universidades: Reflexiones desde Una Experiencia Cubana. *Library Trends*, *67*(4), 669-682. https://doi.org/10.1353/lib.2019.0017

Kroll, P., & Kruchten, P. (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley Professional.

Lenawati, M., & Sari, E. R. N. (2022). Pemanfaatan BPMN Dan Sparx Systems Enterprise Architect Dalam Penyusunan Peta Proses Bisnis Program Studi Sistem Informasi. *RESEARCH : Journal of Computer, Information System & Technology Management*, *5*(2), Art. 2. https://doi.org/10.25273/research.v5i2.13661

Márquez Martín, C. (2022). *Gestión de información georreferenciada y API para web y apps*. https://ddd.uab.cat/record/264188

Marrero, L., Thomas, P. J., Pasini, A. C., Bertone, R. A., Ibañez, E. J., Aguirre, V., Olsowy, V., Tesone, F., & Pesado, P. M. (2020). *Aspectos de ingeniería de software, bases de datos relacionales y bases de datos no relacionales para el desarrollo de sistemas de software en escenarios híbridos*. XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020, El Calafate, Santa Cruz). http://sedici.unlp.edu.ar/handle/10915/104026

Martínez, A., & Martinez, R. (2000). *Guía a Rational Unified Process*.

Monago Ruiz, A. (2019). *Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle*. Escuela Técnica Superior de Ingeniería Universidad de Sevilla.

Nakano, T., Garret, P., Vásquez, A., & Mija, Á. (2016). La integración de las TIC en la educación superior: Reflexiones y aprendizajes a partir de la experiencia PUCP. *En Blanco y Negro*, *4*(2). https://revistas.pucp.edu.pe/index.php/enblancoynegro/article/view/8936

Navarro, Y. de la C., Díaz Sánchez, I., & García Puja, L. M. (2018). Sistema de gestión de la información para la actividad docente: Una opción de mejoramiento.

Ventana Informática, *39*, 83-93. https://doi.org/10.30554/ventanainform.39.3313.2018

Pacheco Laje, J. L. (2018). *ESTUDIO COMPARATIVO ENTRE UNA ARQUITECTURA CON MICROSERVICIOS Y CONTENEDORES DOCKERS Y UNA ARQUITECTURA TRADICIONAL (MONOLÍTICA) CON COMPROBACIÓN APLICATIVA*. Universidad de Guayaquil.

Pham, A. D. (2020). *Developing back-end of a web application with NestJS framework. Case: Integrify Oy's student management system* [Bachelor of Business Administration, LAB UNIVERSITY OF APPLIED SCIENCES LTD]. https://www.theseus.fi/bitstream/handle/10024/353200/Pham_Duc.pdf?sequence=2

Prayogi, A. A., Niswar, M., Indrabayu, & Rijal, M. (2020). Design and Implementation of REST API for Academic Information System. *IOP Conference Series: Materials Science and Engineering*, *875*(1), 012047. https://doi.org/10.1088/1757-899X/875/1/012047

Pressman, R. S. (2010). *Ingeniería del Software un Enfoque Práctico* (7ma ed.). McGraw-Hill Interamericana Editores S.A. de C.V.

Puciarelli, L. (2020). *Node JS - Vol. 1: Instalación - Arquitectura - node y npm*. RedUsers.

Ríos, J. R. M., Ordóñez, M. P. Z., Segarra, M. J. C., & Zerda, F. G. G. (2017). Estado del arte: Metodologías de desarrollo en aplicaciones web. *3c Tecnología: glosas de innovación aplicadas a la pyme*, *6*(3), 54-71.

Romeu, N. (2020). *Implementación de un cotizador on line para el transporte de carga internacional* [Tesis, Universidad Nacional de La Plata]. http://sedici.unlp.edu.ar/handle/10915/118499

Ruíz Jhones, A., & Vidal Larramendi, J. (2019). La Informatización de las Universidades: Reflexiones desde una Experiencia Cubana. *Library Trends*, *67*(4), 669-682. https://doi.org/10.1353/lib.2019.0017

Saltos Pérez, L. A. (2022). *Estudio comparativo entre bases de datos relacional y no relacional* [BachelorThesis, Babahoyo: UTB-FAFI. 2022]. http://dspace.utb.edu.ec/handle/49000/11669

Sánchez Martínez, I., Ríos Rodríguez, L. R., & Sánchez Prado, C. (2017). SISTEMA DE GESTIÓN DE INFORMACIÓN DEL DEPARTAMENTO DE INGENIERÍA INFORMÁTICA DE SANCTI SPÍRITUS. *Pedagogía y Sociedad*, *20*(50). http://revistas.uniss.edu.cu/index.php/pedagogia-y-sociedad/article/view/603

Sánchez Salas, R. (2022). *Diseño y desarrollo de API y servicios web RESTful para la gestión de Chatbot multiplataforma: La UEx como caso de estudio* [BachelorThesis]. https://dehesa.unex.es:8443/handle/10662/15840

Sommerville, I. (2005). *Ingeniería del software* (7ma ed.). PEARSON ADDISON WESLEY.

Tacilla Ludeña, J. L. (2016). Sistema informático web de gestión de incidencias usando el framework AngularJS y Node.js para la empresa Redteam Software LLC. *Universidad Privada Antenor Orrego*. https://repositorio.upao.edu.pe/handle/20.500.12759/3416

Tihomirovs, J., & Grabis, J. (2016). Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. *Information Technology and Management Science*, *19*(1), Art. 1.

Torres Vivanco, M., & Castellanos Gutiérrez, A. (2021). Sistema informático para la gestión del plan de trabajo mensual de los docentes universitarios. *Serie Científica de la Universidad de las Ciencias Informáticas*, *14*(2), Art. 2.

Vallejo, J. E. C., Riascos, J. S. G., & Salazar, A. B. (2020). Esencialización de la práctica gestión de requisitos de rup. *ReCIBE, Revista electrónica de Computación, Informática, Biomédica y Electrónica*, *9*(1), Art. 1. https://doi.org/10.32870/recibe.v9i1.169