Software de gestión de información de facultades para apoyar los procesos de calidad académica en la Universidad de Boyacá

Sergio Iván Martínez Pulido

Universidad de Boyacá
Facultad de Ciencias e Ingeniería
Programa de Ingeniería de Sistemas
Tunja
2023

Software de gestión de información de facultades para apoyar los procesos de calidad académica en la Universidad de Boyacá

Sergio Iván Martínez Pulido

Trabajo de grado para optar al título de:
Ingeniero de Sistemas

Directora:
Clara Patricia Avella Ibáñez
Ingeniera de Sistemas

Universidad de Boyacá
Facultad de Ciencias e Ingeniería
Programa de Ingeniería de Sistemas
Tunja
2023

Nota de aceptación:

_____

_____

_____

_____

_____

_____

Firma del presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Tunja, 31 de mayo de 2023

"Únicamente el graduando es responsable de las ideas expuestas en el presente trabajo". (Lineamientos constitucionales, legales e institucionales que rigen la propiedad intelectual).

Agradecimientos

Me gustaría agradecer a los distinguidos ingenieros del programa de Ingeniería de Sistemas, cuyas enseñanzas y guías me permitieron aplicar de manera eficiente los conocimientos adquiridos en este trabajo.

Asimismo, extiendo mi gratitud a la prestigiosa Universidad de Boyacá, institución académica que me brindó los recursos y las herramientas necesarias para llevar a cabo este proyecto con éxito. Su compromiso con la formación de profesionales altamente capacitados en el campo de la ingeniería de sistemas ha sido esencial en mi formación académica. Quiero hacerles llegar mi más profundo agradecimiento a todas aquellas personas y entidades que contribuyeron al desarrollo de este proyecto, ya que su invaluable apoyo ha sido clave en la consecución de este logro académico.

Contenido

Pág.

Lista de Figuras

Pág.

Lista de Anexos

Pág.

Glosario

Api rest: Una API REST (también conocida como API RESTful) es una interfaz de programación de aplicaciones (API o API web) que cumple con las restricciones del estilo arquitectónico REST y permite la interacción con servicios web RESTful. REST significa transferencia de estado representacional y fue creado por el científico de la computación Roy Fielding. (Red Hat, 2020).

Azure: es una plataforma de computación en la nube creada por Microsoft que ofrece servicios de alojamiento, almacenamiento, procesamiento y análisis de datos, entre otros (Microsoft, s.f.).

Backend: es la parte de un sistema o aplicación que se encarga de procesar los datos y la lógica de negocio. Es la parte que se ejecuta en el servidor y se comunica con el Frontend (Fitzgibbons, 2019).

Builds: se refiere al proceso de compilar el código fuente de una aplicación para crear un archivo ejecutable o paquete instalable.

Crud: es un acrónimo que significa Create, Read, Update y Delete. Se utiliza para describir las operaciones básicas de un sistema de gestión de base de datos (CodeAcademy, s.f.).

Dart: es un lenguaje de programación desarrollado por Google que se utiliza principalmente para crear aplicaciones móviles y web con Flutter.

Docker: es una plataforma de software que permite la creación y gestión de contenedores, que son entornos aislados para la ejecución de aplicaciones (Amazon, s.f.).

Endpoints: son los puntos de acceso a una API o servicio web.

Express js: es un framework de Node.js que se utiliza para crear aplicaciones web y API REST de forma rápida y sencilla (ExpressJs, s.f.).

Ftp: es un protocolo de transferencia de archivos que se utiliza para enviar y recibir archivos entre un cliente y un servidor (Kerner, s.f.).

Framework: es un conjunto de herramientas, librerías y convenciones de programación que se utilizan para desarrollar aplicaciones de software.

Frontend: es la parte de una aplicación que se ejecuta en el navegador o el dispositivo del usuario y que se encarga de la presentación y la interacción con el usuario.

Flutter: es un framework de desarrollo de aplicaciones móviles y web creado por Google que utiliza el lenguaje de programación Dart (Flutter, 2022).

Flutter widgets: son los componentes básicos de la interfaz de usuario en Flutter. Los widgets pueden ser personalizados y combinados para crear la interfaz de usuario de una aplicación.

Hosting: es el servicio de alojamiento de archivos y datos en un servidor remoto, que permite que una aplicación web o móvil esté disponible en línea (Rouse, 2012).

Json: es un formato de intercambio de datos que se utiliza para transmitir información entre sistemas y aplicaciones (w3schools, s.f.).

Landing page: es una página web diseñada para capturar la atención de los visitantes y llevarlos a realizar una acción específica, como suscribirse a un servicio o comprar un producto (Unbounce, s.f.).

Mongodb: es una base de datos NoSQL que se utiliza para almacenar datos en documentos JSON (MongoDB, s.f.).

NoSql: es un término que se refiere a los sistemas de bases de datos que utilizan modelos de datos no relacionales y no utilizan SQL como lenguaje de consulta (Couchbase, s.f.).

NodeJs: es un entorno de tiempo de ejecución de JavaScript que se utiliza para crear aplicaciones de servidor y API REST (NodeJS, s.f.)**.**

Post, Put o Delete: son verbos HTTP que se utilizan para realizar operaciones de creación, actualización y eliminación de datos en un servicio web o API REST.

Product backlog: es una lista de todas las funcionalidades y requisitos de un proyecto, ordenadas por prioridad y mantenidas por el equipo de desarrollo en colaboración con el cliente (Scrum.org, s.f.).

Scrum: es un marco de trabajo ágil para la gestión de proyectos de software que se centra en la colaboración, la comunicación y la entrega iterativa e incremental de funcionalidades (Scrum.org, s.f.).

Swagger: es una herramienta de documentación y prueba de API que permite describir las operaciones y los parámetros de una API de forma clara y estructurada (Swagger.io, s.f.).

Web service: es un servicio que se proporciona a través de Internet y que permite a diferentes sistemas y aplicaciones comunicarse entre sí mediante la transferencia de datos en formato XML o JSON (Ionos, s.f.).

Workspace: se refiere al entorno de trabajo en el que se desarrolla una aplicación, que incluye el código fuente, las herramientas de desarrollo y los recursos necesarios para la ejecución de la aplicación.

Resumen

Software de gestión de información de facultades para apoyar los procesos de calidad académica en la Universidad de Boyacá:

Este documento describe la creación de un aplicativo web para la Universidad de Boyacá, con el objetivo de gestionar la información académica generada por las facultades. El proyecto se enfocó en identificar requisitos, diseñar y desarrollar una base de datos NoSQL (MongoDB), crear el FrontEnd utilizando Flutter para web, programar el BackEnd con Node.js y documentarlo con Swagger. También se realizó la instalación y despliegue del aplicativo web y la Api Rest.

Este proyecto tiene como objetivo principal el desarrollo de una aplicación de software en ambiente web que permita la gestión de información generada por las facultades, para apoyar los procesos de calidad académica en la Universidad de Boyacá.

La metodología utilizada combina las etapas tradicionales de construcción de software con algunas prácticas del marco de desarrollo Scrum, adaptadas para ser ejecutadas por un solo desarrollador. Estas etapas incluyen el análisis, diseño, desarrollo, pruebas e implementación, y se realizan bajo el marco de desarrollo ágil.

En conclusión, la aplicación de software desarrollada en este proyecto brinda una solución tecnológica para la gestión de información de facultades que apoya los procesos de calidad académica en la Universidad de Boyacá. El uso de la metodología ágil y la adaptación de algunas prácticas de Scrum permitió el desarrollo eficiente y efectivo del proyecto por parte de un solo desarrollador.

Palabras claves: Construcción de software, Procesamiento de datos, Almacenamiento de datos, Interfaz de usuario, Api Rest, NodeJS, Framework de código abierto, Desarrollo ágil de software, Contenedores virtuales, Scrum.

Abstract

Faculty information management software to support academic quality processes at the Universidad de Boyacá:

This document describes the creation of a web application for the University of Boyacá, aimed at managing the academic information generated by the faculties. The project focused on identifying requirements, designing and developing a NoSQL database (MongoDB), creating the FrontEnd using Flutter for web, programming the BackEnd with Node.js, and documenting it with Swagger. The installation and deployment of the web application and the REST API were also performed.
The main objective of this project is to develop a web-based software application that allows the management of information generated by the faculties to support the academic quality processes at the Universidad de Boyacá.

This project has as its main objective the development of a web-based software application that enables the management of information generated by the faculties to support academic quality processes at the University of Boyacá.

The methodology used combines the traditional stages of software construction with some practices of the Scrum development framework, adapted to be executed by a single developer. These stages include analysis, design, development, testing, and implementation, and are carried out under the agile development framework.

In conclusion, the software application developed in this project provides a technological solution for the management of faculty information and supports academic quality processes at the Universidad de Boyacá. The use of agile methodology and the adaptation of some Scrum practices allowed for efficient and effective development of the project by a single developer.

Keywords: Software development, Data processing, Data storage, User interface, Rest Api, NodeJS, Open-source framework, Agile software development, Virtual containers, Scrum.

Introducción

El propósito general del proyecto que se presenta en este documento es el desarrollo de un sistema de información para la gestión de información de las facultades de la Universidad de Boyacá, necesario para los procesos de calidad académica. La característica principal del proyecto es el desarrollo de una infraestructura de software que cumpla con los requisitos establecidos. Se ha identificado que la Universidad de Boyacá carece de un módulo automatizado que capture la información generada por las facultades, que es necesaria para los procesos de autoevaluación realizados por la institución.

El proyecto se ha desarrollado para resolver el problema identificado en la División de Planeación y Acreditación de la Universidad de Boyacá y en las facultades, que consiste en la falta de un sistema informático web que permita gestionar la información requerida para los procesos de autoevaluación con fines de acreditación. El propósito del proyecto es desarrollar un sistema para capturar, almacenar y consultar la información generada por las facultades para utilizar en procesos de calidad académica.

Además de resolver este problema específico, el proyecto también tiene como objetivo explorar aspectos importantes del desarrollo de software web que son relevantes para la formación de ingenieros de sistemas. El documento sigue una estructura clara y organizada que comienza con la definición de los requisitos de la aplicación en el Product Backlog de Azure. A continuación, se procede al modelado de la base de datos y a la implementación de los componentes del aplicativo Frontend (Flutter Web) y Backend (NodeJs). Una vez completado este proceso, se dedica una sección especial a la documentación del proyecto. Seguidamente, se lleva a cabo el despliegue del aplicativo, incluyendo la configuración de la Web, el Web Service y la base de datos. Finalmente, se concluye con la implementación de SCRUM del proyecto.

1. Definición de requisitos de la aplicación en el product backlog de azure

1.1. Identificación de historias de usuario

Para llevar a cabo este proyecto, se identificó la necesidad de desarrollar un aplicativo web capaz de administrar registros, lo que implica la capacidad de agregar, actualizar, eliminar y consultar datos. Para alcanzar este objetivo, se desarrollaron dos capas que permitieran construir el aplicativo: Frontend y Backend, con el uso de una base de datos no relacional.

Para la capa de Frontend, se utilizó el Framework de Flutter web debido a su facilidad de implementación y existencia de una buena documentación. En cuanto al Backend, se optó por utilizar NODEJS debido a la fiabilidad de su ambiente de desarrollo para este tipo de proyectos. Finalmente, se definió la base de datos de MongoDB, ya que tiene la capacidad de administrar una gran concurrencia de datos, lo que es fundamental para este tipo de proyectos web.

Con base en lo expuesto, posteriormente se obtuvieron los requisitos de la aplicación para cada una de las capas del proyecto (frontend y backend), y a partir de estos, se identificaron las entidades del modelo de negocio, con el propósito de esquematizar la base de datos y comprender el correcto funcionamiento del proyecto.

Una vez que se esquematizó la base de datos, se procedió con el diseño del aplicativo web. Para lograr esto, se trabajó en el diseño de las interfaces necesarias para la aplicación, con el fin de determinar cómo sería la interacción del usuario con la misma. Como resultado de este proceso, se crearon tres pantallas principales que formarían la base del aplicativo. Además, se determinó que el aplicativo contaría con un cierto número de tablas y formularios asociados a ellas. En total se crearon 17 tablas y formularios con la representación de FACU_XX, que irían de FACU_01 a FACU_17. Estas tablas y formularios fueron diseñados para ser asociados por facultad y usuario, lo que implicó la necesidad de contar con un sistema de logueo para los usuarios, el cual permite la asociación de los mismos a su respectiva facultad. Con este enfoque, se busca garantizar la integridad de los datos y aumentar la seguridad en la aplicación. Por último, se definió una pantalla que permite a los usuarios ver su historial de uso de la aplicación. Esta pantalla proporcionará trazabilidad a los datos, lo que es útil para los administradores de la aplicación. De esta manera, es posible conocer cuáles fueron los registros creados, actualizados y eliminados de la aplicación, por un usuario específico.

Una vez que se definió el Frontend de la aplicación web, se procedió con la construcción del Backend para conectar la base de datos con el aplicativo web, lo cual se realizó mediante un Web Service. La construcción de este Web Service implicó la definición de una arquitectura de software adecuada para gestionar las solicitudes y respuestas de la aplicación web. Para lograr esto, se plantearon varios controladores que se encargarían de manejar la lógica de parte del servidor. Estos controladores incluyen el controlador de inicio de sesión (Sign in) y consulta de usuarios, que permitiría la autenticación de los usuarios y la verificación de sus credenciales. También se incluyó un controlador para el registro de tablas, que permitirá la creación de nuevas tablas en la base de datos cuando sea necesario. Además, se implementó un controlador para el historial, que permitiría registrar todas las interacciones del usuario con la aplicación, lo que proporcionará información valiosa para los administradores de la aplicación.

Después de definir la parte de diseño del aplicativo web y la lógica del servidor, y en cumplimiento al marco de trabajo Scrum, se plasmaron los requisitos del sistema en forma de Historias de Usuario. Estas historias de usuario son descripciones detalladas de las funcionalidades que se esperan del aplicativo web. Estas descripciones se utilizan como base para guiar el desarrollo de la aplicación y asegurar que se cumplan los requisitos del usuario y del negocio.

En este caso, las historias de usuario plasmadas fueron las siguientes:

- Como usuario del sistema requiero una pantalla de inicio de sesión
- Como usuario del sistema requiero una pantalla de inicio de sesión y una pantalla "Landing" donde se mostrarán las tablas y registros del sistema.
- Como usuario del sistema, requiero contar con formularios para agregar información a 17 tablas diferentes para captura de información generada por las facultades (FACU_01 a FACU_17).
- Como usuario del sistema, requiero una pantalla de historial donde pueda ver el recuento de los registros realizados por los usuarios.
- Como usuario de la plataforma, quiero tener acceso a mi historial de transacciones mediante una API
- Como autor del proyecto, necesito definir una estructura inicial de proyecto, para que pueda establecer una base sólida del proyecto

- Como desarrollador del proyecto, necesito una estructura inicial y bien organizada del proyecto y una estructura limpia y bien organizada para la API NodeJS, además del uso de una arquitectura limpia (Clean Architecture).
- Como autor del proyecto, quiero diseñar un esquema de base de datos que contemple todas las tablas y relaciones necesarias para el correcto funcionamiento de la aplicación.
- Como autor del proyecto, quiero preparar un entorno para despliegue según se requiera

Estas historias de usuario se plasmaron en el Product Backlog de Azure, que es una herramienta que permite la gestión de los requisitos del proyecto, los cuales se priorizan y se van desglosando en tareas más pequeñas para guiar el desarrollo de la aplicación, a medida que se realiza cada Sprint (Scrum.org, s.f.). De esta manera, se puede seguir un enfoque ágil en el desarrollo del proyecto y garantizar que se cumplan los requisitos del usuario y del negocio de manera eficiente y efectiva.

1.1.1. Generación del product backlog

El Product Backlog es una lista ordenada de requisitos que se deben implementar para desarrollar un producto o servicio en la plataforma de Azure. Cada requisito en el Product Backlog debe tener una descripción detallada que describa su funcionalidad y los objetivos que se deben cumplir. La descripción detallada de cada requisito es esencial para garantizar que el equipo de desarrollo comprenda los requisitos del producto y pueda trabajar en ellos de manera efectiva y eficiente. (Proyectosagiles.org, 2021).

Para lograr una mejor comprensión y efectividad en el desarrollo del proyecto, se proporciona una descripción detallada de cada requisito en el Product Backlog de Azure. En esta descripción se incluye el propósito de la historia de usuario, su alcance, los usuarios, los procedimientos relacionados, las precondiciones y las poscondiciones. Además, se incluye una descripción de la historia de usuario o procedimiento en sí. Esta información se proporciona tanto al desarrollador como al Product Owner con el fin de que ambos tengan una comprensión clara de cada requisito y puedan trabajar de manera efectiva para lograr los objetivos del proyecto.

La descripción detallada de las historias de usuario del product backlog, se puede revisar el Anexo 2 de este documento "***Levantamiento de requisitos***". Este anexo contiene información relevante y completa acerca de las historias de usuario incluidas en el backlog del proyecto.

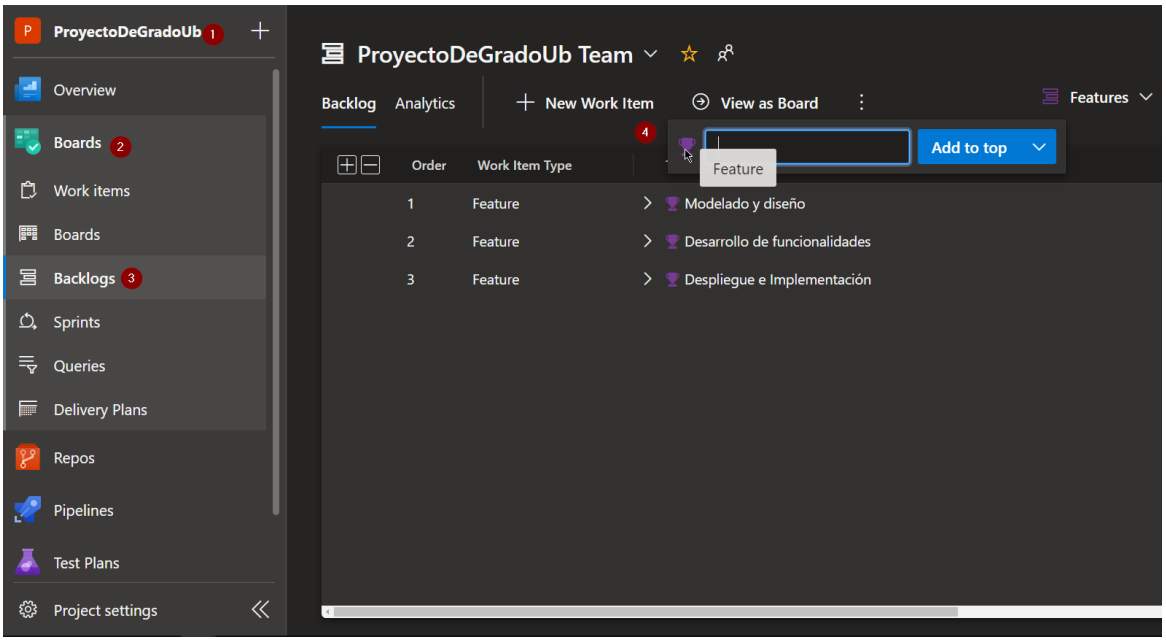
1.1.2. Incorporación de los requisitos validados al product backlog de azure

Una vez que se han definido las historias de usuario, es necesario agregarlas al Product Backlog de Azure, que es el registro completo de todas las características, funcionalidades y mejoras que se deben implementar en el producto final. En esta etapa, fue utilizado Azure DevOps como plataforma para gestionar el Product Backlog. Azure DevOps es una herramienta integral que permite no solo manejar el Product Backlog, sino también asociarlo con otros aspectos del proyecto, como la gestión de repositorios y la planificación de Sprints.

1.1.2.1. **Creación de features en azure backlogs.** En este proyecto se identificaron tres Features que representan las fases principales del proyecto. Cada Feature está compuesto por una serie de Historias de Usuario que se relacionan entre sí para lograr el cumplimiento del Feature correspondiente. Para crear estas Historias de Usuario, se utiliza el workspace del proyecto en Azure DevOps, accediendo al apartado de Boards y seleccionando la opción de Backlogs. A continuación, se agrega un nuevo work item de tipo Feature, tal como se muestra en la *Figura 1*, para establecer los objetivos específicos que se busca alcanzar con cada uno de los Features.

Figura 1

Proceso para agregar nuevo Feature en Azure DevOps

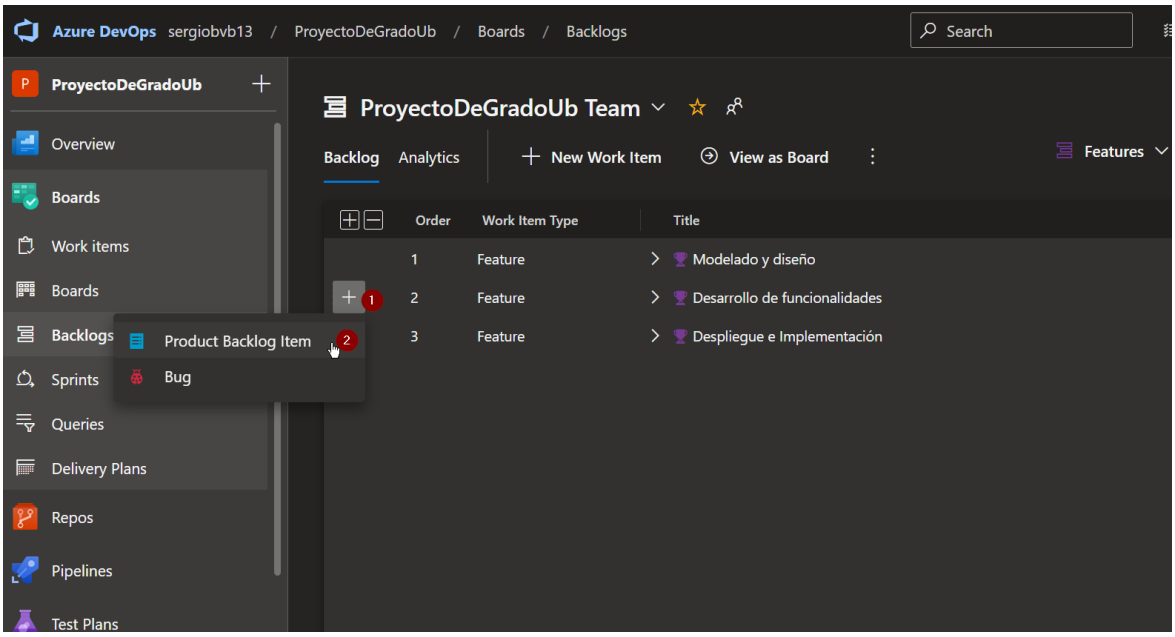


Fuente: Elaboración propia

1.1.2.2. **Creación de historias de usuario en Feature Azure Backlogs.** Una vez que se han definido las Historias de Usuario para el proyecto, es necesario agregarlas a los Features correspondientes. Para ello, se accede al workspace del proyecto en Azure DevOps y se ubica el Feature al que se desea agregar la Historia de Usuario. A continuación, se hace clic en el botón de "más opciones" y se selecciona la opción de "Product Backlog Item", que corresponde a una Historia de Usuario específica, tal como se ilustra en la *Figura 2*. De esta manera, se establece una relación clara entre las Historias de Usuario y los Features del proyecto, lo que permite una mejor organización y seguimiento del progreso del mismo**.**

Figura 2

Proceso para agregar nueva historia de usuario

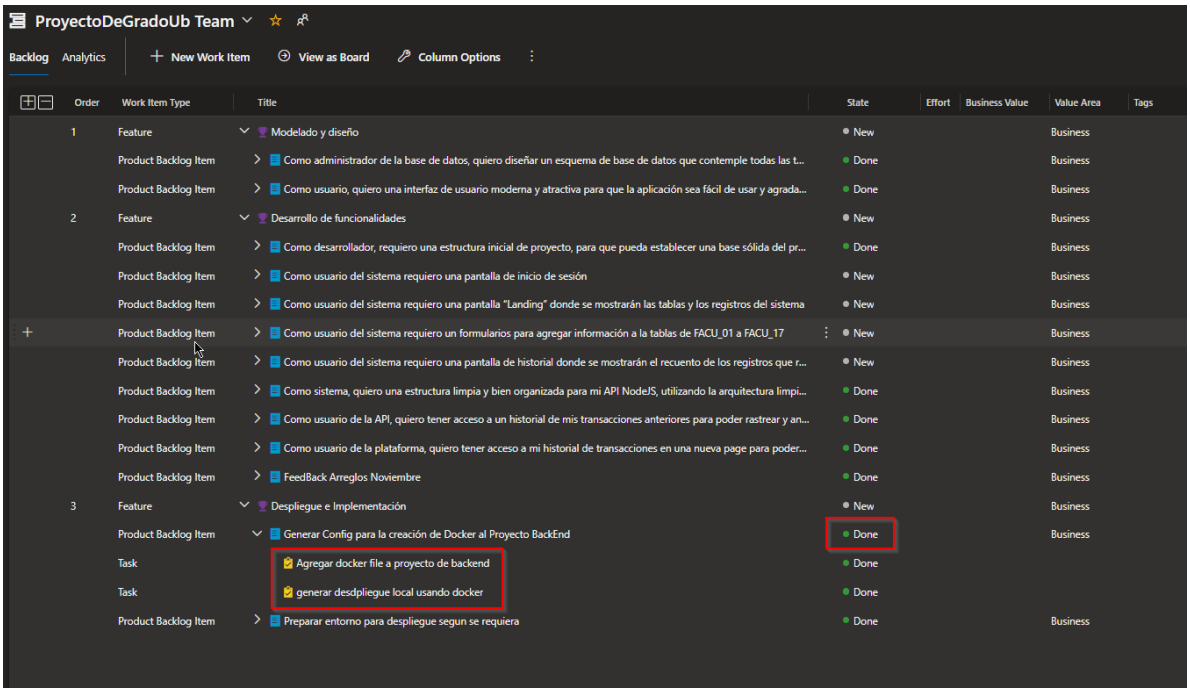


Fuente: Elaboración propia

De esta manera, se agregan las Historias de Usuario previamente definidas en el Product Backlog de Azure DevOps. Cada Historia de Usuario tiene un estado que permite identificar su seguimiento en el proyecto. Además, cada Historia de Usuario está compuesta por una serie de tareas específicas que se relacionan directamente con su cumplimiento, tal como se muestra en la Figura 3. Esta estructura de Historias de Usuario y sus tareas asociadas permite establecer una clara división de trabajo en el equipo, lo que facilita la gestión del proyecto y el seguimiento del progreso de cada elemento.

Figura 3

Historias de usuario en el Backlog de Azure DevOps



Fuente: Elaboración propia

2. Modelado de la base de datos

El modelado de bases de datos es un proceso importante en el diseño de bases de datos para comprender las entidades y las relaciones en diferentes bases de datos. El modelado es una técnica para representar los datos de un sistema de información y facilitar la comprensión de la estructura y los procesos del sistema. En general, el modelado de bases de datos se refiere a la creación de un diseño conceptual de la estructura de la base de datos que incluye entidades, atributos y relaciones (AppMaster, 2022).

2.1. Identificación y definición de los requisitos de la base de datos

Para este proyecto se ha seleccionado MongoDB como motor de base de datos, debido principalmente a su rapidez y facilidad de uso. Al tratarse de una base de datos no relacional, permite una mayor flexibilidad en el modelado y creación de objetos, lo que se ajusta perfectamente a las necesidades del proyecto en cuestión. Además, al ser una base de datos orientada a documentos, permite una escalabilidad horizontal más sencilla y eficiente, lo que es importante considerando que el proyecto puede experimentar un aumento en la cantidad de datos a manejar.

Para este proyecto se propuso la creación de 6 colecciones en la base de datos.

- **Tabla**: representa las 17 tablas en las que los usuarios pueden agregar registros pertinentes según corresponda a cada tabla. Esta colección cuenta con 4 atributos que permiten gestionar de manera eficiente la información almacenada en cada tabla y optimizar su acceso. Estos atributos son:
    1. **_id**: identificador único
    2. **descripción**: descripción de la tabla
    3. **indicadores**: hace referencia a información relacionada a la tabla pertinente
    4. **items**: es una lista de las columnas que posee esta tabla ejemplo: [AÑO, SEDE, PROGRAMA, etc.…]
- **Facultad:** representa las diferentes facultades de la universidad a las que se relacionan los registros, usuarios y programas. Esta colección cuenta con 3 atributos que permiten

organizar de manera eficiente la información almacenada en cada una de las facultades y establecer relaciones efectivas entre ellas. Estos atributos son:

1. **_id**: identificador único
2. **sigla**: sigla de la facultad
3. **descripción**: descripción de la facultad

- **Programa**: representa cada uno de los programas académicos asociados a cada facultad de la universidad. Esta colección es de gran utilidad para agregar registros relacionados con los programas y para gestionar la información de manera eficiente. Estos atributos son:
    1. **_id**: identificador único
    2. **sigla**: sigla del programa
    3. **facultad**: sigla de la facultad asociada
    4. **ciudad**: ciudad donde se imparte este programa ejemplo: Tunja, Sogamoso, Duitama
- **Usuario:** representa a los usuarios del sistema y sirve para almacenar información relevante de cada uno de ellos. Esta colección es esencial para el correcto funcionamiento del sistema, ya que se utiliza principalmente para el inicio de sesión de los usuarios y para identificar quién realiza cada uno de los registros. La colección de USUARIOS cuenta con 6 atributos que permiten almacenar de manera eficiente la información relevante de los usuarios. Estos atributos son:
    1. **_id**: identificador único
    2. **nombre**: nombre del usuario
    3. **email**: correo electrónico institucional con el que el usuario se loguea al sistema
    4. **facultad**: sigla de facultad asociada al usuario
    5. **rol**: rol que tiene el usuario en el sistema puede ser [usuario, admin]
    6. **Password**: contraseña del usuario esta debe estar encriptada ejemplo: "***$2b$10$30/0med5QgKMJGUQFeetQOE3fsUoktumT5NwVaM/0FpOmlt4kEyNC***"esta encriptación la hace el Backend
- **Registro:** representa los diferentes registros que los usuarios pueden hacer en el sistema. Esta colección está asociada a las colecciones de TABLA, USUARIOS y FACULTAD. La colección de REGISTRO cuenta con 5 atributos que permiten

gestionar de manera eficiente la información almacenada en cada registro. Estos atributos son:

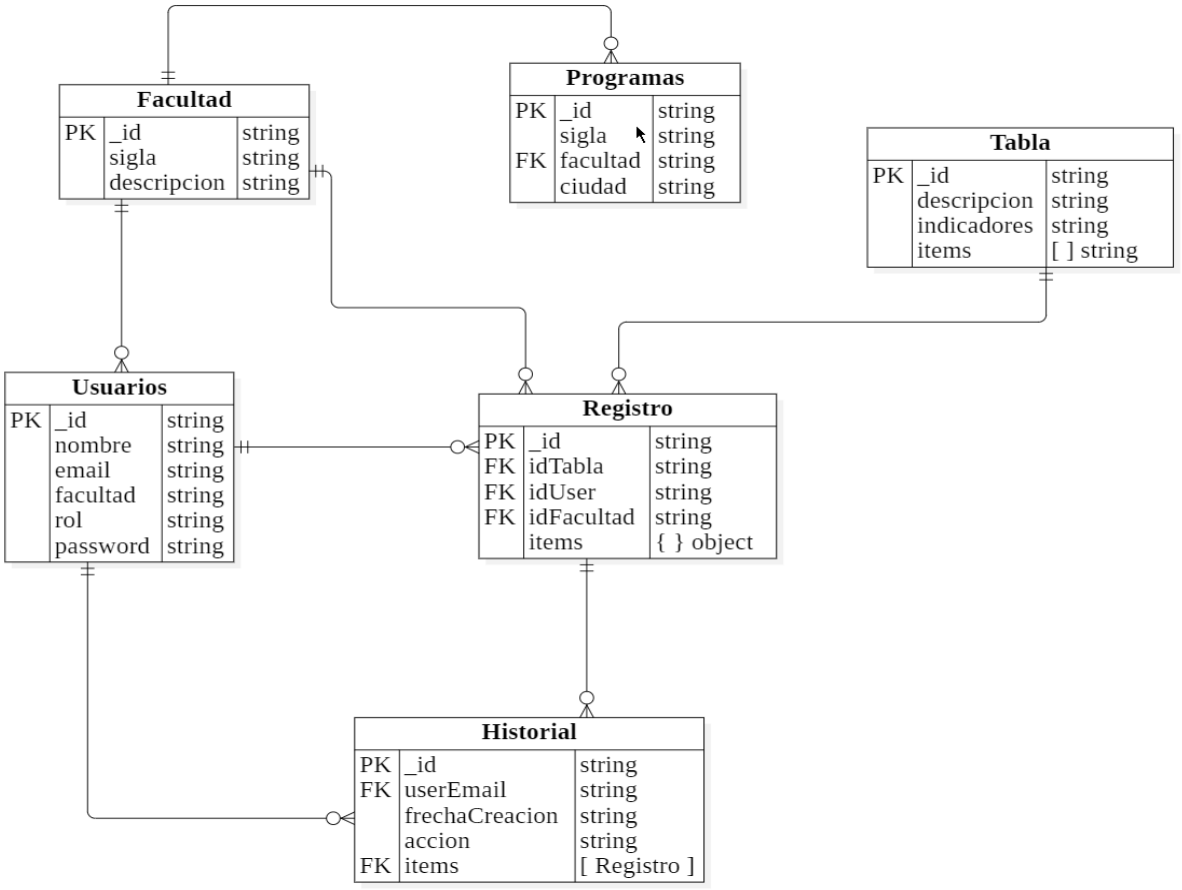
1. **_id**: identificador único
2. **idTabla**: identificador único de la tabla a la que se le va asociar este registro
3. **idUser**: identificador único de usuario a la que se le va asociar este registro
4. **idFacultad**: sigla de facultad que se le asocia al registro
5. **items**: es un objeto JSON que representa el registro, como tal no está normalizado debido a que cada tabla puede tener distintos valores en sus registros. Se hace uso de la flexibilidad de la base de datos para guardar estos registros, ejemplos de estos registros {JSON}.

- **Historial**: La colección de HISTORIAL representa el historial de las acciones realizadas en los registros, ya sea POST, PUT o DELETE. Esta colección está estrechamente asociada a la colección de REGISTRO, lo que permite identificar de manera efectiva la trazabilidad de los registros realizados por los usuarios. La colección de HISTORIAL cuenta con 5 atributos que permiten gestionar de manera eficiente la información almacenada en cada registro de historial. Estos atributos son:
    1. **_id**: identificador único
    2. **userEmail**: correo electrónico del usuario que realizar alguna acción sobre el registro
    3. **fechaCreacion**: fecha en la que el usuario interactúa con el registro
    4. **accion**: acción que realiza el usuario con el registro [POST, PUT, DELETE]
    5. **items**: registro en cuestión, hace una referencia directa al registro

Todas las entidades de la base de datos y sus atributos se representan en la *Figura 4*.

Figura 4

Modelo de base de datos para la aplicación



Fuente: Elaboración propia

3. Aplicativo frontend (flutter web)

En esta sección, se describe el proceso de diseño e implementación de la Interfaz Gráfica (UI – User Interface por su sigla en inglés) de la aplicación, así como la integración con el Backend de la misma. El propósito era crear una aplicación con una interfaz gráfica atractiva, fácil de usar y que brinde una experiencia de usuario óptima. Para ello, se utilizan herramientas para diseñar la interfaz gráfica y se desarrolla la lógica del Frontend con el uso de widgets y componentes de Flutter. Además, se integra el Frontend con el WebService Backend de la aplicación para permitir una comunicación fluida entre ambos.

3.1. Diseño UI y maquetación de cada página

Antes de comenzar con la maquetación de la aplicación, es esencial diseñar los bocetos de la interfaz de usuario (UI). Para ello, se utilizan herramientas de diseño como Figma, las cuales permiten crear diseños visuales precisos de las páginas, botones, campos de entrada de datos, entre otros elementos. En ocasiones se utilizan herramientas más sencillas como Paint o Excalidraw para realizar bocetos iniciales que sirvan como guía en el proceso de maquetación de las interfaces de usuario. Este enfoque garantiza que se tenga una idea clara de cómo se verá y funcionará la aplicación antes de avanzar en su desarrollo y ayuda a ahorrar tiempo y esfuerzo en el proceso de corrección de errores. En este proyecto se utilizó la herramienta de diseño Figma.

3.1.1. UI inicio de sesión

Para la pantalla de inicio de sesión, se buscó crear un diseño básico y fácil de usar, teniendo en cuenta que esta pantalla es crucial para el acceso del usuario al sistema. La pantalla presenta un formulario sencillo con dos campos de texto para ingresar el correo electrónico y la contraseña, respectivamente, y un botón "Login" que permite al usuario iniciar sesión en la aplicación.

Después de completar el boceto inicial, se procedió con la maquetación de la pantalla de inicio de sesión en Flutter, en la cual se mantuvo la mayoría de los elementos del prototipo inicial. Si bien hubo algunos cambios, estos fueron menores y se pueden observar en las *Figuras 5 y 6*

correspondientes. En general, se buscó mantener una interfaz intuitiva y fácil de usar para asegurar una experiencia de usuario positiva.

Figura 5

Boceto inicial pantalla inicio de sesión

Fuente: Elaboración propia

Figura 6

Pantalla definitiva de inicio de sesión

Fuente: Elaboración propia

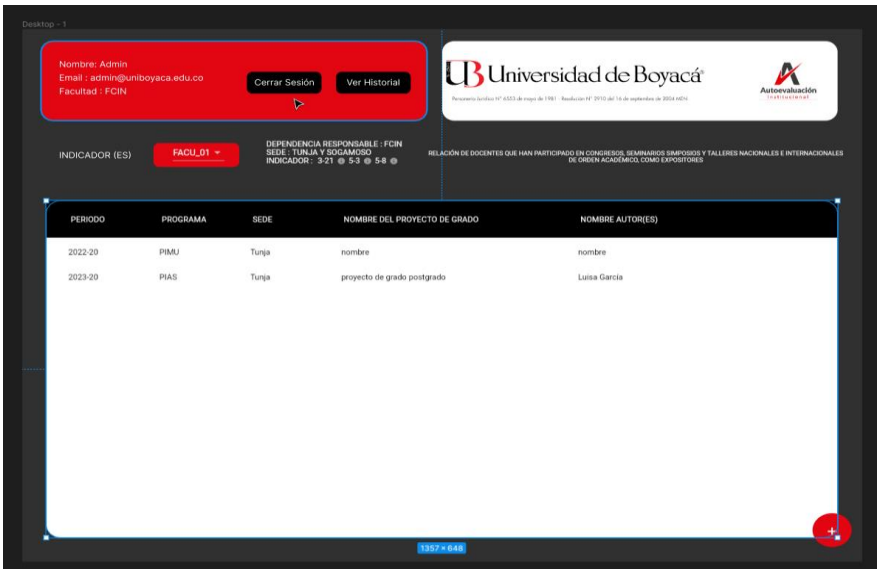
3.1.2. UI landing page

Al enfocarse en la pantalla de "Landing", se buscó crear un diseño básico y fácil de usar, aunque con algunas diferencias importantes respecto a la pantalla de inicio de sesión. En este caso, se utilizó la herramienta de diseño Figma para maquetar la pantalla y configurar aspectos clave del diseño como colores y tamaños. La pantalla presenta varias secciones de utilidad para el usuario. En la parte superior izquierda, se muestra información relevante sobre el usuario, como su nombre, correo electrónico y facultad, junto con dos botones que describen claramente su función: "Cerrar sesión" y "Ver historial".

En la parte inferior, se encuentra una sección de tablas que incluye un menú desplegable con todas las tablas disponibles y una breve descripción de cada una. A bajo del menú, se muestra una tabla dinámica que cambia según la tabla seleccionada. Esta tabla muestra filas para cada registro agregado por el usuario y permite acceder a los detalles de cada registro al hacer clic en la fila correspondiente. Finalmente, se incluye un botón con el icono de un "+", que abre un formulario para agregar nuevos registros a la tabla seleccionada. La misma se puede evidenciar en la *Figura 7.*

Figura 7

Diseño de la Landing Page



Fuente: Elaboración propia

Una vez finalizado el proceso de diseño del boceto para la pantalla de "Landing", se procedió a la maquetación de la interfaz en Flutter. En este proceso se buscó mantener la fidelidad al diseño inicial, lo que permitió conservar los elementos y características que se habían considerado más importantes durante la fase de diseño. Gracias a ello, la pantalla de "Landing" mantuvo su estructura original y se pudo asegurar que los usuarios pudieran acceder a toda la información relevante de manera clara y sencilla.

Uno de los principales objetivos en el diseño de la interfaz de usuario fue lograr una experiencia intuitiva y fácil de usar. Por ello, se trabajó en la disposición de los elementos y en la selección de los colores y tipografías más adecuados para garantizar una buena legibilidad. Además, se tuvo en cuenta la ubicación de los botones y las opciones de navegación para que fueran fácilmente accesibles para el usuario como se evidencia en la *Figura 8*.

Figura 8

Maquetación de la Landing Page

Fuente: Elaboración propia

3.1.3. UI página historial

En la pantalla de historial, a pesar de haber sido agregada posteriormente, se mantuvo el enfoque en la simplicidad y facilidad de uso que caracteriza a todo el proyecto. Aunque no se

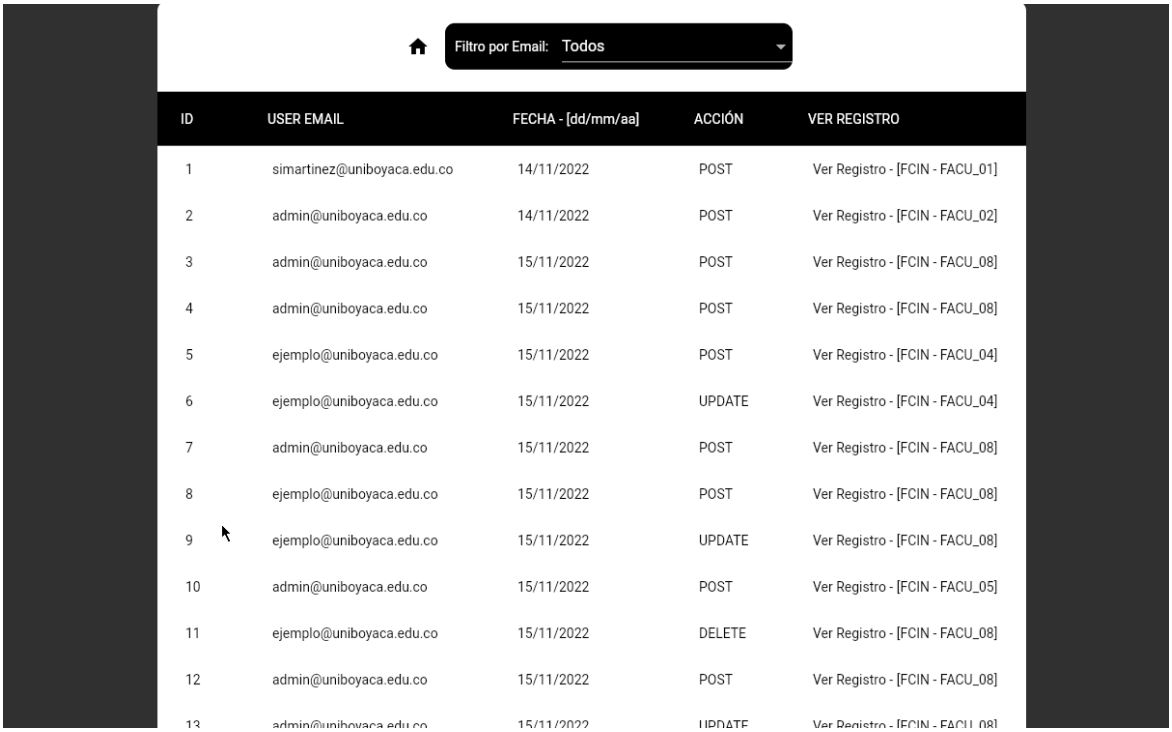
realizó un prototipo específico para esta pantalla, se tomó en cuenta la necesidad del Product Owner de tener un historial de los registros generados por los usuarios.

La pantalla se diseñó con una tabla que muestra información relevante como el correo del usuario, la fecha de creación del registro y la acción realizada por el usuario, lo que facilita la búsqueda de registros específicos. Además, se incluyó una opción para ver el registro en cuestión, lo que permite al usuario profundizar en la información del mismo.

Después de presentar la pantalla a los Product Owner, se recibió Feedback y del mismo se agregar un filtro de correo de los usuarios que se muestran en la tabla, lo cual se implementó para mejorar la usabilidad de la funcionalidad del historial. Este filtro permite al usuario buscar rápidamente registros específicos por correo de usuario, lo que se puede apreciar explicitamente en la *Figura 9*.

Figura 9

Maquetación página historial



Fuente: Elaboración propia

3.2. Maquetación de formularios

La maquetación de los formularios de cada tabla fue un proceso importante en el desarrollo de la aplicación. Se buscó crear un diseño intuitivo y fácil de usar para los usuarios, lo que llevó a la creación de formularios dinámicos que se ajustan en función de la tabla seleccionada. Los formularios fueron generados como componentes para garantizar la reutilización eficiente del código, lo que facilita el mantenimiento y la escalabilidad de la aplicación.

En cada formulario se presentan los campos necesarios para crear un registro en la tabla correspondiente, con diferentes tipos de campos como texto, número, fecha y opciones desplegables. Se agregaron validaciones en cada campo para asegurarse de que los usuarios ingresen información válida. Además, se agregaron botones para guardar y cancelar la creación del registro, lo que aumenta la usabilidad de la aplicación.

Cada formulario de tabla tiene un diseño específico que se ajusta a las necesidades de la tabla seleccionada, lo que mejora la experiencia del usuario al utilizar la aplicación. La modularización de los formularios permitió una fácil integración en otras partes de la aplicación, como lo puede ser el “Ver registro” de la página de historial, lo que agiliza el proceso de desarrollo y asegura la coherencia en la interfaz de usuario la misma se evidencia en la *Figura 10.*

Figura 10

Maquetación de formulario

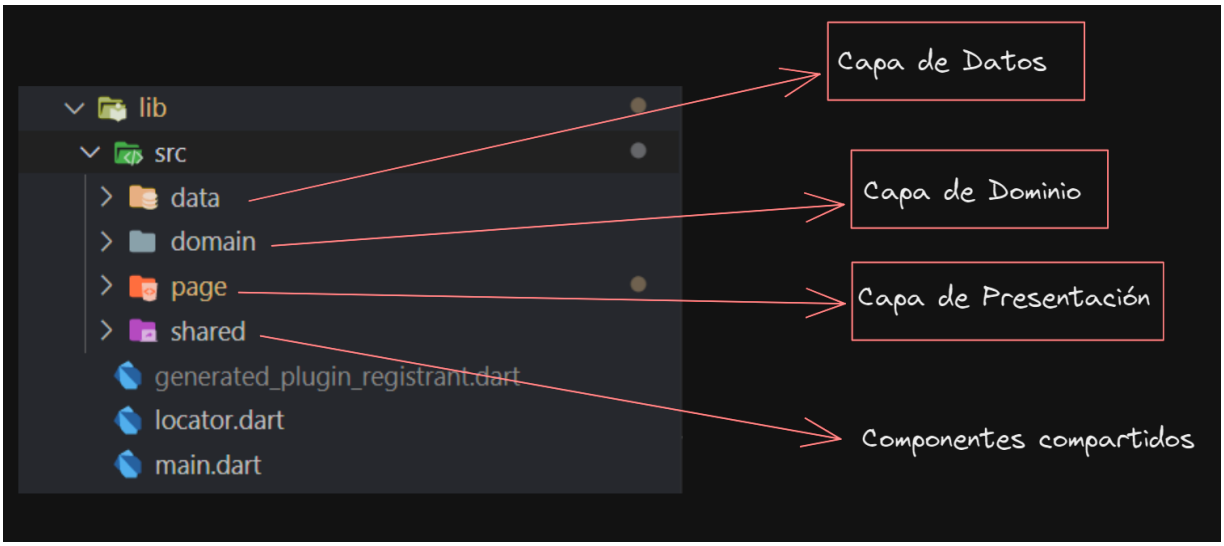
Fuente: Elaboración propia

3.3. Arquitectura del proyecto - flutter

En este proyecto de ingeniería de sistemas se ha desarrollado una arquitectura en Flutter Web basada en dos patrones de arquitectura: Modelo Vista Controlador (MVC) y Modelo Vista Vista Modelo (MVVM). Patrones de diseño muy utilizados en este Framework. Esta arquitectura se divide en tres capas: presentación, dominio y datos. Como se representa en la *Figura 11*.

Figura 11

Estructura de carpetas de la arquitectura



Fuente: Elaboración propia a partir de la captura de pantalla de la estructura de la aplicación

La capa de presentación es la encargada de tener todas las interfaces de usuario que se van a mostrar en la aplicación. En cada una de estas interfaces se aplica el patrón MVVM, que consiste en separar la lógica de la interfaz de usuario, lo que permite que cada pantalla tenga una página y su correspondiente controlador. De esta forma, se puede manejar la lógica asociada a cada pantalla de manera más ordenada y estructurada como se muestra en la *Figura 12*

Figura 12

Capa de presentación de la arquitectura

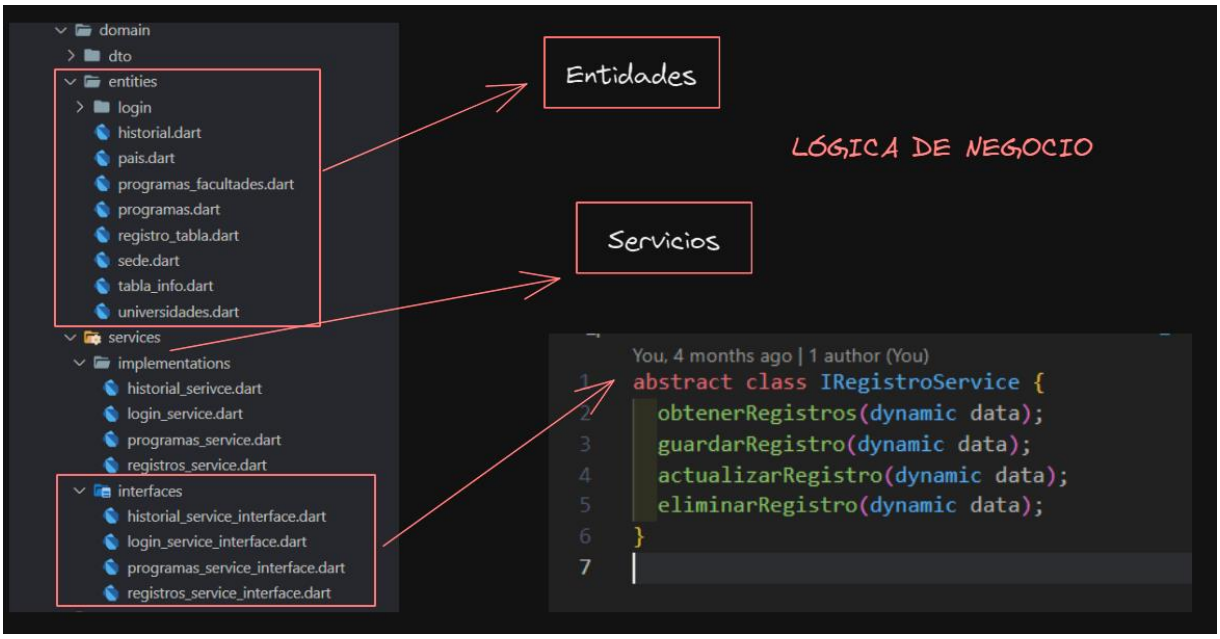
Fuente: Elaboración propia a partir de la captura de pantalla de la estructura de la aplicación

Por otro lado, la capa de dominio es donde se implementa toda la lógica de negocio. En esta capa se encuentran las entidades o modelos, los servicios de conexión a la capa de datos, los DTOs, entre otros componentes asociados a esta capa. Aquí se trabaja con el patrón MVC, que consiste en separar la lógica de negocio de la lógica de presentación. Esto permite que la lógica de negocio sea más independiente y pueda ser reutilizada en diferentes interfaces como se puede ver en la *Figura 13*.

Para conectar esta capa con las demás capas se usa el concepto de clases abstractas o bien llamadas interfaces, que las mismas son las que se inyectan donde se va a usar el servicio mediante el patrón de inyección de dependencias.

Figura 13

Capa de dominio de la arquitectura



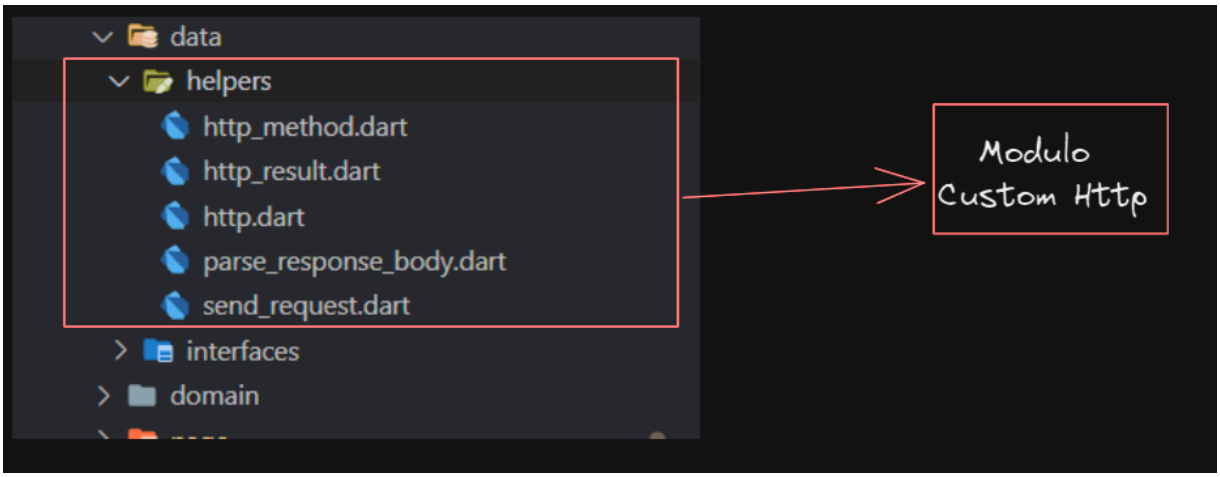
Fuente: Captura de pantalla de la estructura de la aplicación

Por último, la capa de datos actúa como un puente para conectar los servicios externos al aplicativo mediante interfaces. Esta capa se encarga de realizar las operaciones de lectura y escritura en la base de datos o la conexión con servicios externos como pueden ser Web Services (API REST) que es el aplicado a este proyecto, además de proveer la información necesaria para la capa de dominio. El mismo se muestra evidenciado en la *Figura 14.*

En este caso, la capa de datos proporciona un módulo personalizado de HTTP que permite que la capa de dominio solicite este módulo y le pase la información correspondiente a través de parámetros. Esto no solo promueve la reutilización de código, sino que también facilita la implementación del código.

Figura 14

Capa de datos de la arquitectura



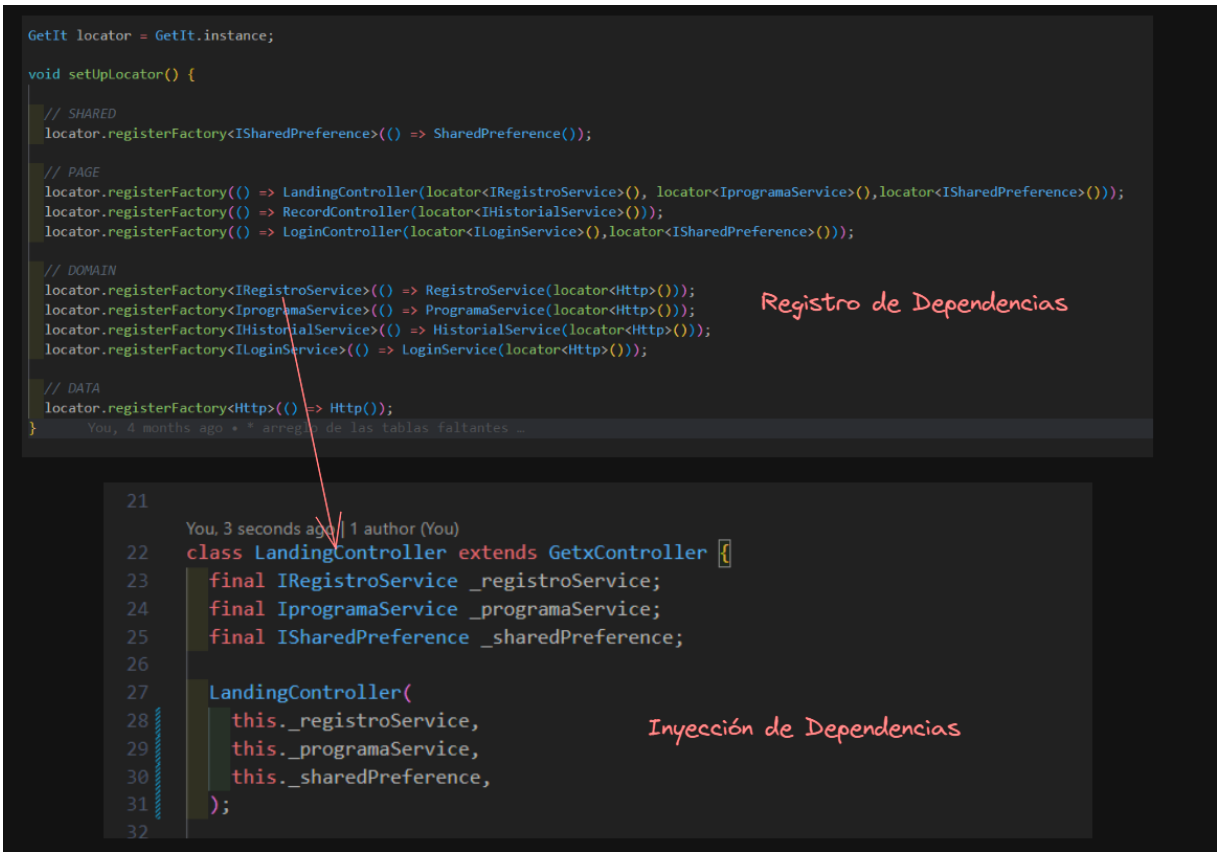
Fuente: Captura de pantalla de la estructura de la aplicación

Además de estos componentes de la arquitectura, se ha implementado el concepto de inyección de dependencias, el cual se registra mediante un "Service Locator". Este "Service Locator" registra todas las dependencias de las interfaces que se inyectan a los controladores, lo que permite una gestión más eficiente y estructurada de las dependencias.

Para este propósito, es necesario registrar las dependencias en un contenedor de dependencias. Este proceso implica instanciar las dependencias solo una vez en el proyecto, lo que a su vez permite su uso en cualquier momento, mediante la inyección de la interfaz correspondiente, en lugar de tener que inyectar todas las dependencias relacionadas. De esta manera, se promueve una mayor modularidad y escalabilidad del proyecto, lo que se traduce en un código más limpio y fácil de mantener. La *Figura 15* muestra cómo se realiza este proceso en la práctica.

Figura 15

Representación de registro e inyección de dependencias



Fuente: Captura de pantalla de la aplicación

En cuanto a los widgets, se ha agregado un apartado común que serán los widgets que se compartirán a lo largo de la aplicación, principalmente en la capa de presentación. De esta forma, se pueden reutilizar los widgets en diferentes pantallas de la aplicación, lo que ahorra tiempo y esfuerzo en el desarrollo.

4. Aplicativo backend (nodejs)

4.1. Arquitectura del proyecto nodejs

La arquitectura del proyecto Backend en Nodejs está basada en el patrón de diseño MVC (Modelo vista controlador), el cual se utiliza comúnmente en proyectos de API REST. Se ha implementado la arquitectura limpia, lo que significa que el proyecto está separado en carpetas para cada componente y se enfoca en la separación de responsabilidades. Además, se han incluido buenas prácticas de desarrollo como el uso de Docker y la documentación mediante Swagger.

4.2. Patrón modelo vista controlador

Además de la separación de carpetas mencionada anteriormente, es importante destacar que la separación de carpetas o responsabilidades es comúnmente utilizada en el desarrollo de aplicaciones REST, así como aplicaciones Web, ya que es la forma en la cual se implementa el patrón Modelo-Vista-Controlador (MVC). En esta arquitectura, los controladores actúan como intermediarios entre el modelo y las vistas que en este caso son los endponits que se exponen en el API, y se encargan de procesar las solicitudes y devolver repuesta a las mismas.

Por otro lado, la carpeta de modelos es donde se definen las estructuras de datos que se utilizan en el proyecto. Estas estructuras pueden ser objetos, colecciones de objetos, o cualquier otra entidad que sea necesaria para el correcto funcionamiento del proyecto. Es importante mencionar que los modelos no deben contener lógica de negocio, ya que esta responsabilidad recae en los controladores.

Finalmente, las rutas son un elemento crucial en cualquier aplicación API REST, ya que representan la manera en que el usuario interactúa con el servidor. En general, cada ruta representa un endpoint específico del proyecto, y está asociada con un controlador que se encarga de procesar las solicitudes relacionadas. Es importante destacar que la estructura y organización de las rutas puede variar significativamente dependiendo del Framework o lenguaje de programación utilizado en el proyecto, para el caso ExpressJs. Lo anterior se puede ver en la *Figura 16.*

Figura 16

Implementación del patrón MVC (separación de carpetas)

Fuente: Captura de pantalla de la estructura de la aplicación

La arquitectura también incluye la carpeta de configuración, que se encarga de manejar todas las configuraciones de la aplicación, incluyendo la conexión con la base de datos, la configuración de la autenticación y la configuración de los servicios de terceros. El archivo ***index.js*** es el punto de entrada principal del proyecto, que se encarga de iniciar la aplicación.

Además de la arquitectura limpia y la separación de carpetas, se han implementado buenas prácticas de desarrollo, como el uso de Docker para la creación de contenedores, lo que hace que la aplicación sea más fácil de manejar y escalar. También se ha incluido la documentación mediante Swagger, lo que facilita la comprensión de la API REST y la integración con otros servicios.

4.3. Documentación swagger

La documentación en Swagger es importante para proporcionar una descripción detallada de las rutas de la API y cómo se pueden utilizar. Esto puede incluir información como los parámetros de entrada, las respuestas esperadas, los códigos de estado y más. La documentación en Swagger también facilita la creación de pruebas automatizadas y el mantenimiento de la API a lo largo del tiempo. (Swagger.io, s.f.)

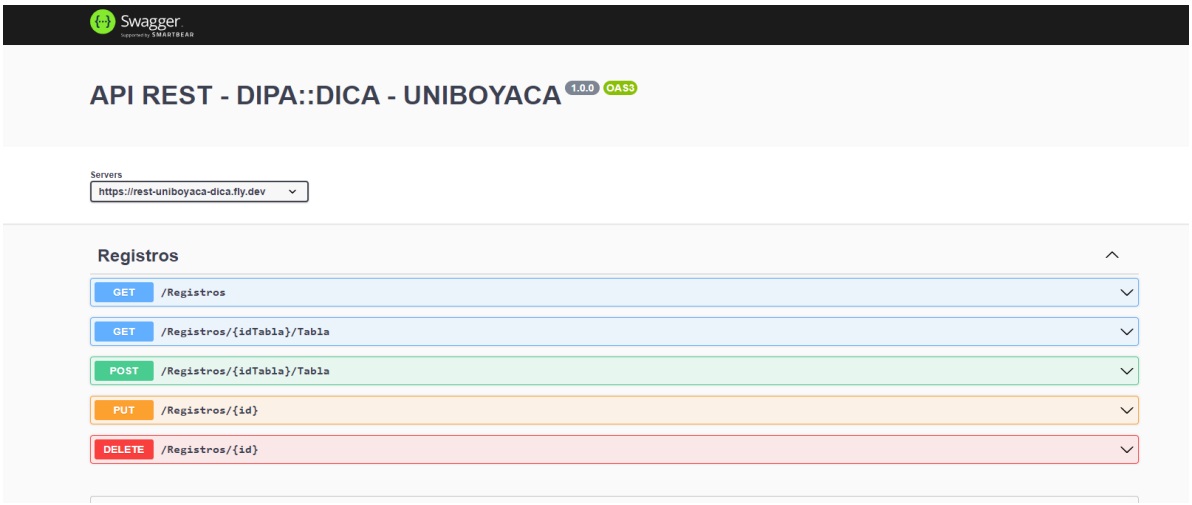
Para este proyecto, se decidió utilizar la documentación de Swagger debido a que es una forma efectiva de documentar el API, toda vez que esta herramienta permite interactuar directamente con los servicios que ofrece el API REST, exponiendo todos sus Endpoint para usarlos de forma directa. Esto se convierte en una alternativa a otras herramientas como Postman e Insomnia.

Para ejemplo, en la vista de registros, Swagger muestra cinco Endpoint que permiten interactuar con ellos, todos relacionados con registros. En este caso, se pueden realizar peticiones GET, POST, PUT y DELETE, lo que permite realizar en completo las operaciones CRUD. Además, Swagger identifica los esquemas utilizados en estos registros, lo que resulta muy útil para visualizar de forma gráfica los datos que se van a recibir y los que se pueden enviar en las peticiones.

Esto es especialmente beneficioso para el desarrollador, ya que le permite visualizar interactivamente todo lo relacionado con la API, ahorrando tiempo en el proceso de desarrollo. Lo anterior se puede evidenciar en la *Figura 17*.

Figura 17

Documentación Swagger



Fuente: Captura de pantalla de la documentación Swagger

5. Documentación

Este proyecto ha sido desarrollado con una arquitectura limpia y bien estructurada, la cual se basa en la separación de responsabilidades; en particular, se ha dividido en tres capas principales: la capa de presentación o interfaz de usuario (Frontend), la capa de dominio o lógica de negocio (back-end) y la capa de datos (db).

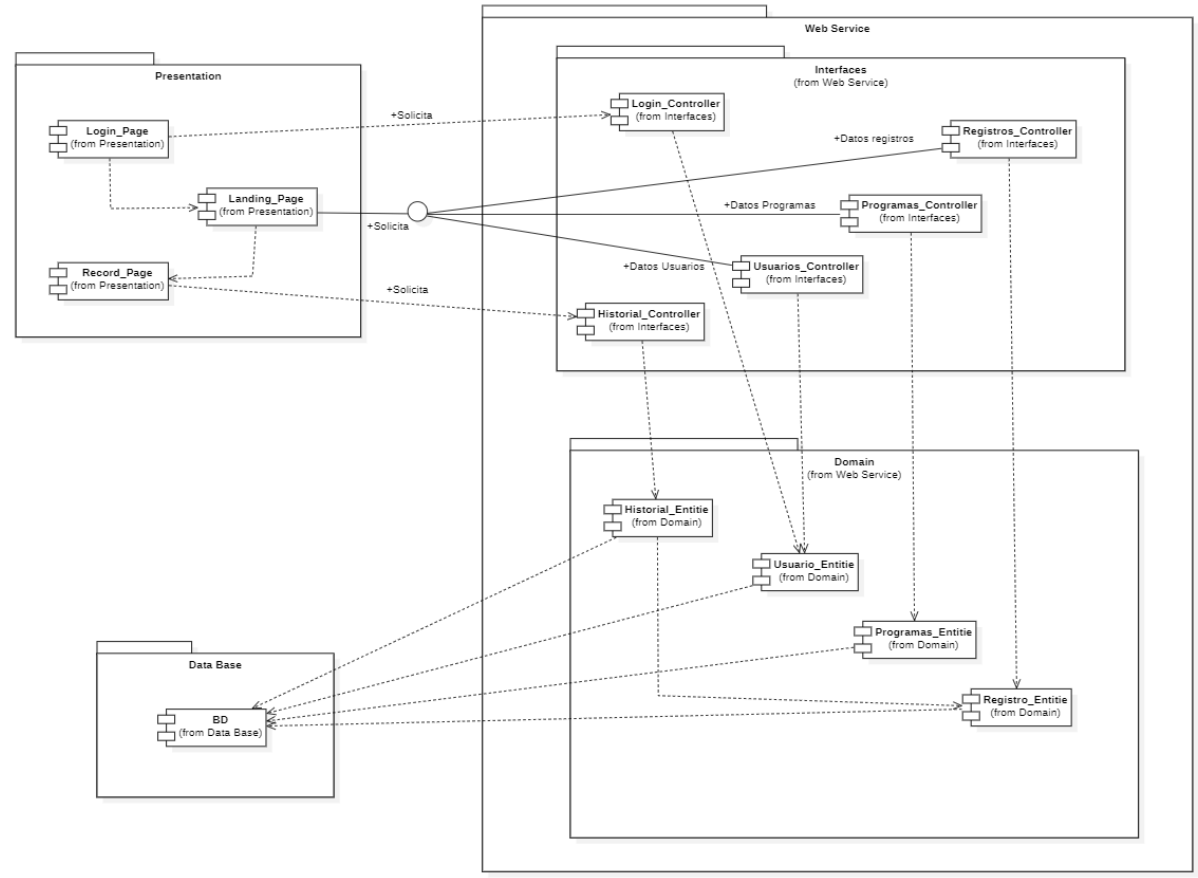
La capa de presentación se encarga de manejar la interacción con el usuario, ofreciendo una interfaz de usuario amigable y atractiva. Por su parte, la capa de dominio se encarga de la lógica de negocio, procesando y manipulando los datos de la aplicación. Finalmente, la capa de datos se encarga de almacenar los datos de la aplicación en una base de datos.

Esta arquitectura permite una separación clara de responsabilidades y un mayor modularidad del proyecto, facilitando el mantenimiento y escalabilidad de la aplicación. Además, al separar la lógica de negocio de la interfaz de usuario y el almacenamiento de datos, se puede garantizar una mayor flexibilidad y adaptabilidad de la aplicación a medida que evoluciona y crece.

La arquitectura de la aplicación desde el punto de vista de componentes se presenta en la *Figura 18*.

Figura 18

Arquitectura de la aplicación, diagrama de componentes



Fuente: Elaboración propia

La documentación es un aspecto fundamental en cualquier proyecto de software, ya que permite a los desarrolladores y usuarios entender cómo funciona el aplicativo y cómo realizar tareas específicas. Aunque la documentación en texto es una opción común y efectiva, también puede resultar tediosa para algunos usuarios revisar grandes cantidades de texto para encontrar la información necesaria.

Para la documentación de arquitectura de la aplicación en cuanto a sus capas, se optó por utilizar la documentación en video, ya que esto facilita la comprensión de los usuarios que realizarán modificaciones en el mismo. En estos videos se muestra una breve introducción a las herramientas de software utilizadas en el proyecto, su instalación, configuración y generación de Builds de cada proyecto para su despliegue, así como los espacios donde se llevará a cabo dicho despliegue. Se considera que esta es la mejor manera de realizar la transición con los

administradores interesados en usar el proyecto y con desarrolladores que quieran realizar mejoras al aplicativo. Los videos de documentación se adjuntan en un CD con tres videos (Anexo 1. Video Documentación), con los siguientes nombres:

- VideoDocumentacionFlutter.mkv
- VideoDocumentacionNodeJs.mkv
- VideoDocumentacionMongoDb.mkv

Se espera que la documentación en video permita una mejor comprensión y facilite la realización de cambios en el proyecto.

6. Despliegue del aplicativo web, web service y bd

6.1. Diagrama de despliegue

Para este proyecto se pueden identificar los 4 nodos principales:

- **PC cliente (navegador):** Este nodo representa la máquina del usuario final que accede a la aplicación web a través de un navegador.
- **Hosting de aplicación web**: Este nodo representa el servicio de alojamiento de la aplicación web. Además, incluye el servidor web y los componentes necesarios para que la aplicación pueda ser acceso ble desde Internet.
- **Web Service (API):** Este nodo representa el Backend (Api Rest) que es la interfaz que permite a los clientes comunicarse con el servidor y acceder a sus servicios y recursos.
- **Servidor de base de datos**: Este nodo representa el servicio de alojamiento de la base de datos. Incluye el servidor de base de datos y los componentes necesarios para que la aplicación pueda acceder y gestionar los datos almacenados.

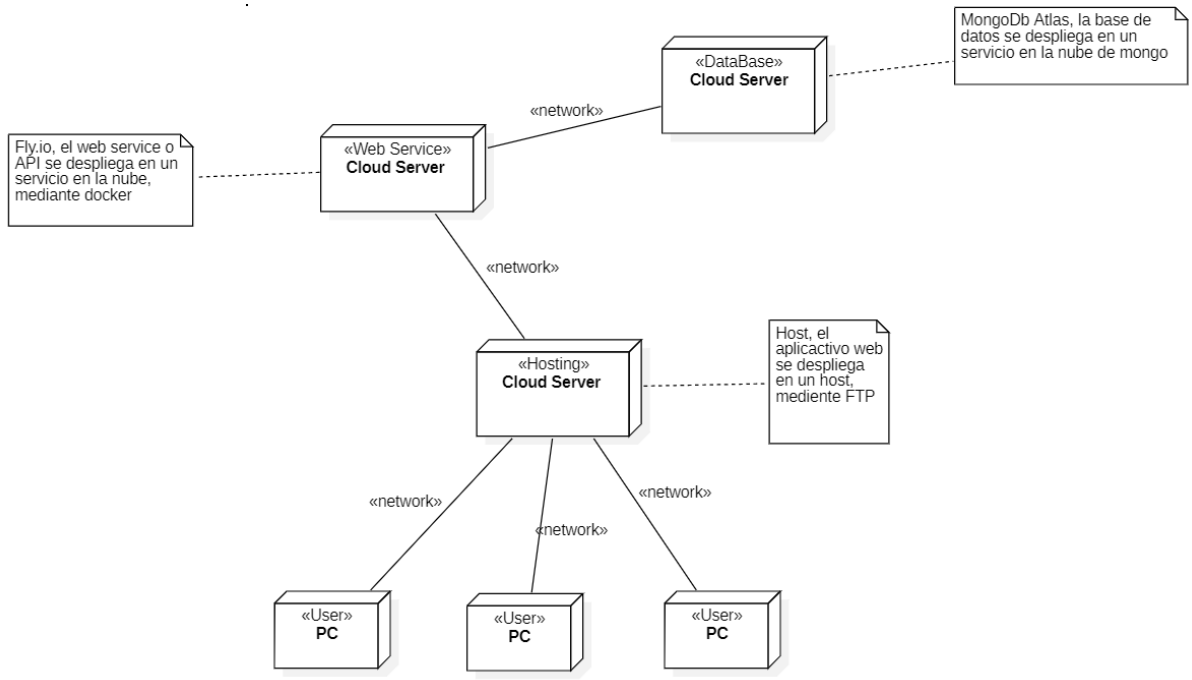
Las conexiones entre los nodos se realizan en el siguiente orden:

a) El PC Cliente se conecta al Hosting de la aplicación web a través de Internet, usando un protocolo de comunicación como HTTP o HTTPS.
b) El Hosting de aplicación web se conecta al web service (API) alojado en fly.io para realizar la lógica de negocio y conectar a su vez con la base de datos.
c) El Servidor de web server alojado en fly.io se conecta al Servidor de base de datos alojado en MongoDb Atlas para acceder y gestionar los datos almacenados en la base de datos.

El diagrama de despliegue se presenta en la *Figura 19.*

Figura 19

Diagrama de despliegue



Fuente: Elaboración propia

6.2. Preparación despliegue web

6.2.1. Build web flutter

Es importante entender que el despliegue de una aplicación web es un paso crucial en el proceso de desarrollo, ya que el mismo permite observar el proyecto en producción, lo cual significa que el usuario final podrá verlo. Para ello, una de las tareas es generar el build de producción de Flutter. Este proceso es importante porque permite compilar todo el código de Dart nativo en un conjunto de archivos que podrán ser subidos al Hosting que se tenga preparado.

Cuando se realiza el proceso de build, Flutter genera archivos que contienen la versión optimizada de la aplicación para la web. En este proceso, el código de Dart se convierte a lenguaje de la web, HTML, CSS y JavaScript. Esto significa que la aplicación podrá ser visualizada en cualquier navegador web.

Es importante destacar que el proceso de build de producción de Flutter tiene en cuenta las dependencias del proyecto, permitiendo que se generen los archivos necesarios para que la aplicación web funcione correctamente.

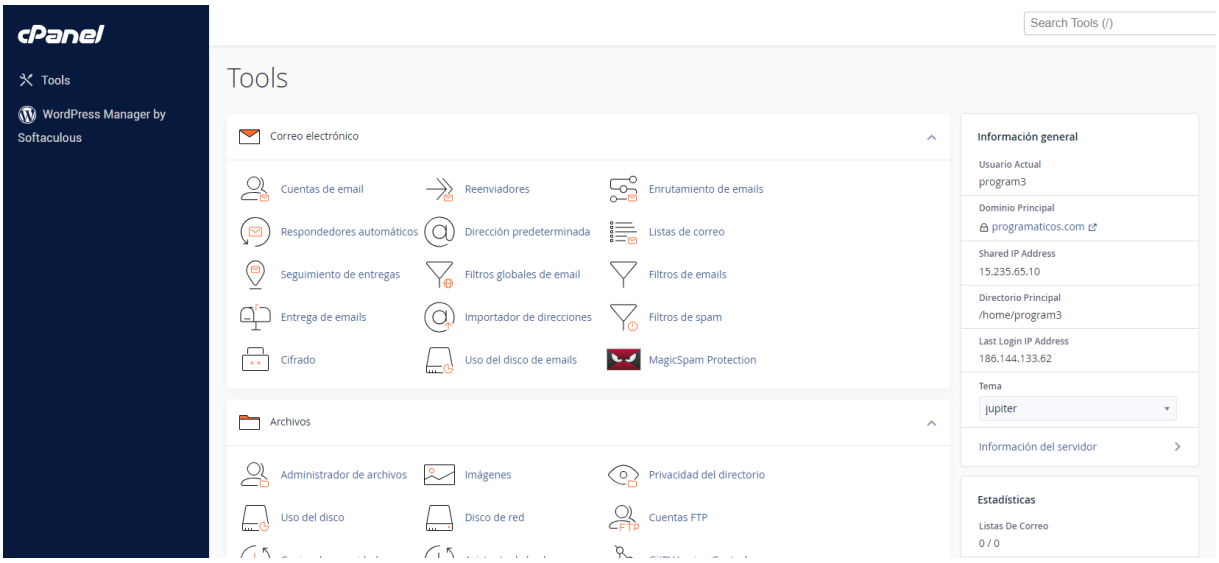
6.2.2. Hosting

Para poder publicar el aplicativo web desarrollado, es necesario contar con un servicio de Hosting. En el caso de este proyecto, se utilizó un servicio de hosting adquirido con ***latinoamericahosting.com,*** para realizar la prueba de instalación del aplicativo desarrollado. El servicio incluye un administrador Cpanel que permite subir los archivos generados en el build y servirlos en la web a través de la creación de un subdominio que hace referencia al proyecto actual. El subdominio cuenta con un espacio de almacenamiento donde se despliegan los archivos del aplicativo. En este proyecto, se decidió cargar los archivos directamente al espacio suministrado por el Hostin,g en lugar de utilizar la opción de FTP.

El administrador de Cpanel cuenta con una función que permite la instalación de un certificado SSL en el dominio correspondiente, el cual se trata de un certificado digital que garantiza la autenticidad del sitio web y encripta la información que se envía al servidor (Kaspersky, s.f.). La incorporación de esta medida de seguridad se puede apreciar en las *Figuras 20 y 21*, y es fundamental destacar que no solo protege la aplicación, sino que también aumenta la confianza de los usuarios al visitar la página.

Figura 20

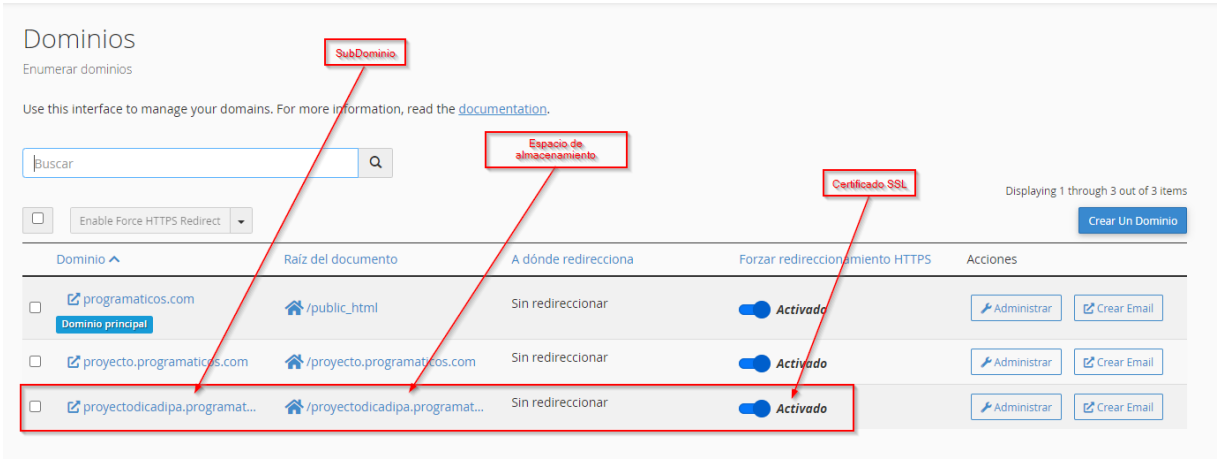
Cpanel Hosting



Fuente: Elaboración propia a partir de la pantalla de captura de Cpanel en el hosting de prueba Programaticos.com

Figura 21

Detalle dominio y sub-dominio hosting



Fuente: Elaboración propia a partir de la captura de pantalla de Cpanel en el hosting de prueba Programaticos.com

Para acceder a la aplicación, simplemente ingresa la siguiente URL: https://dicadipa.programaticos.com en el navegador web.

Para revisar el proceso de puesta en marcha del proyecto Frontend en Flutter y la creación del build correspondiente, se puede acceder al archivo de video (Anexo 2. Video Documentación) "VideoDocumentacionFlutter.mkv".

6.3. Preparación despliegue web service

6.3.1. Build docker

Durante el desarrollo del proyecto de software, se optó por utilizar la tecnología de Docker para el despliegue de la parte de Web Services. Esta elección se debió a que Docker permite empaquetar todas las dependencias necesarias para el correcto funcionamiento del Web Service, lo que facilita el proceso de despliegue en distintos servidores que contengan Docker.

La utilización de Docker ofrece la ventaja de que el equipo de desarrollo solo necesita preocuparse por generar la imagen de Docker con todos los archivos necesarios para ejecutar la aplicación, y posteriormente seleccionar el servidor en el que se llevará a cabo el despliegue. De esta manera, se reduce significativamente el tiempo y esfuerzo necesarios para el proceso de despliegue y se garantiza una mayor eficiencia en el mismo.

6.3.2. Desplegar web service

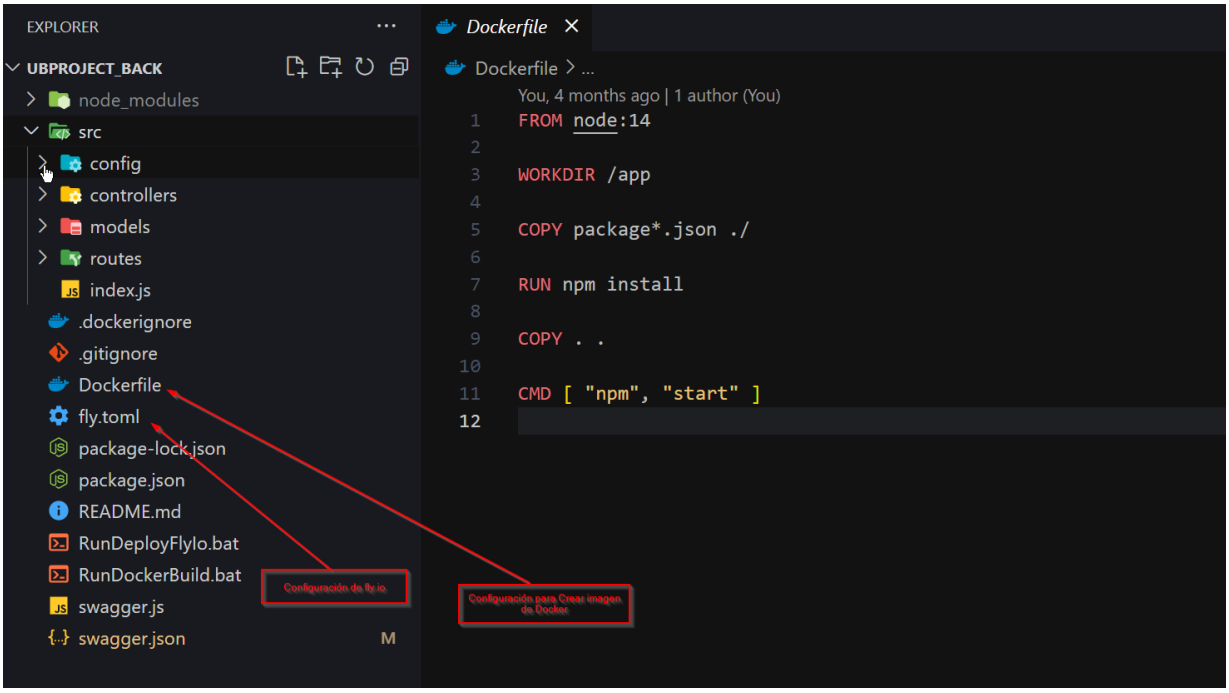
Para este proyecto se ha decidido emplear un servicio gratuito que facilita el despliegue de imágenes de Docker en un Servidor Virtual Privado (VPS - Virtual Private Server por sus siglas en inglés), denominado "*Fly.io*". Este servicio brinda la opción de desplegar imágenes de Docker y ajustar los recursos según la demanda del aplicativo, lo que posibilita una escalabilidad más sencilla y sin complicaciones.

Para usar dicho servicio, se creó una cuenta en la plataforma y se procedió a configurar los archivos necesarios para su despliegue, siguiendo la documentación proporcionada por el mismo servicio. La configuración detallada se encuentra en las *Figuras 22 y 23*. Además, Fly.io también proporciona herramientas para monitorear el uso de recursos y mejorar el rendimiento del aplicativo. De este modo, se logra un despliegue eficiente y escalable del web Service del proyecto.

Es importante tener en cuenta que el proceso de despliegue de la aplicación puede variar según el servidor, sin embargo, gracias a la utilización de la tecnología de Docker, las configuraciones necesarias para el funcionamiento del aplicativo permanecen invariables. Por lo tanto, se optó por esta tecnología para asegurar la consistencia del proceso de despliegue y para permitir que la aplicación se ejecute de manera uniforme en cualquier servidor que contenga Docker.

Figura 22

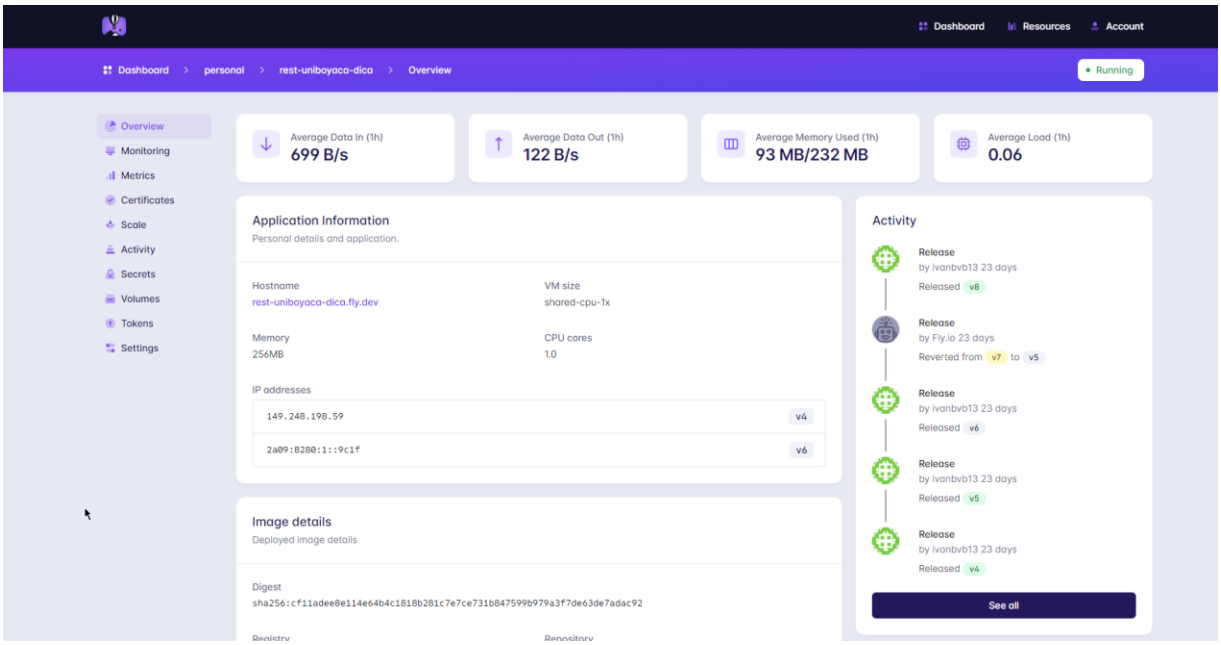
Configuración en proyecto docker y fly.toml



Fuente: Elaboración propia a partir de la captura de pantalla de la estructura de la aplicación

Figura 23

Panel servicio fly.io



Fuente: Captura de pantalla del panel de servicio Fly.io

La aplicación del Web Service no cuenta con una interfaz gráfica, ya que esta no fue diseñada con ese propósito. Sin embargo, para facilitar la comprensión de la aplicación del web Service, se ha implementado una interfaz de usuario en forma de documentación de Swagger. La documentación de Swagger se puede acceder a través de la siguiente URL https://rest-uniboyaca-dica.fly.dev/api-docs/ en un navegador web.

Para revisar el proceso de puesta en marcha del proyecto Backend en NodeJS y la creación del build correspondiente, se puede acceder al archivo de video (Anexo 3. Video Documentación) "VideoDocumentacionNodeJs.mkv".

6.4. Preparación despliegue bd

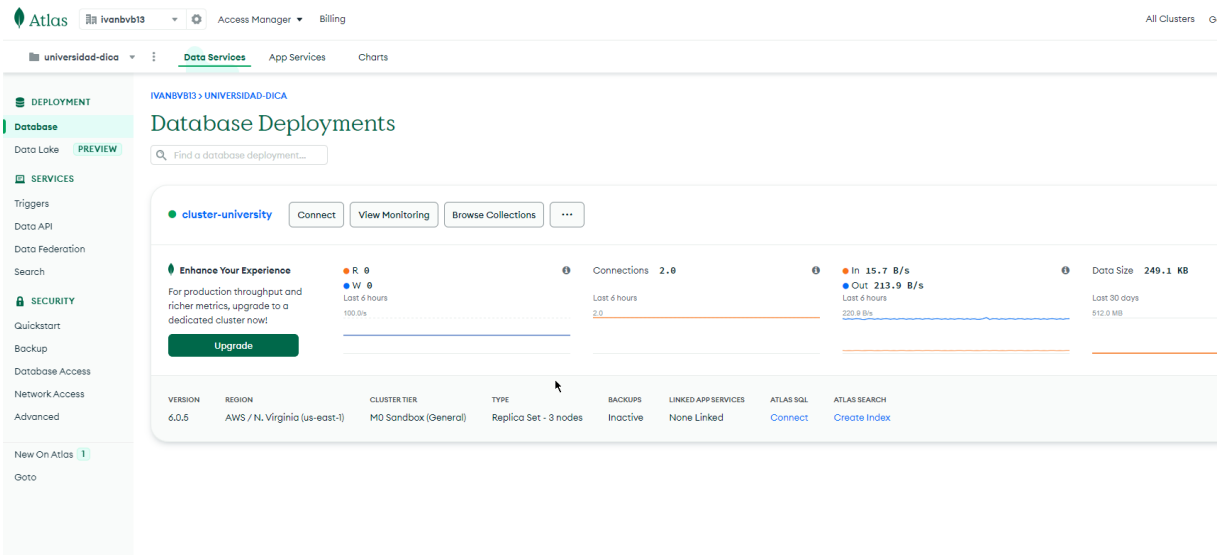
Como se mencionó anteriormente, para el desarrollo de este proyecto se eligió una base de datos no relacional como MongoDB. Para usar esta base de datos se ha utilizado un servicio gratuito ofrecido por la misma compañía, conocido como MongoDB Atlas. Este servicio consta de un clúster de base de datos en la nube, donde se puede desplegar una instancia de base de datos

MongoDB con configuraciones específicas que permiten la conexión a través de un String de conexión como se evidencia en la *Figura 24*.

Es importante tener en cuenta que el servicio gratuito presenta ciertas limitaciones. Sin embargo, si se desea desplegar una base de datos MongoDB, se puede instalar MongoDB en un servidor propio y exponer su puerto mediante una dirección IP pública, bajo ciertas restricciones. De esta manera, cualquier servicio o aplicación puede conectarse a la base de datos utilizando el String de conexión correspondiente.

Figura 24

Panel administrativo de MongoDb Atlas



Fuente: Captura de pantalla de MongoDb Atlas

Para revisar el proceso de puesta en marcha del proyecto Backend en NodeJS y la creación del build correspondiente, se puede acceder al archivo de video (Anexo 4. Video Documentación) "VideoDocumentacionMongoDb.mkv".

7. Implementación de scrum

7.1. Envisioning

La visión del Proyecto en Scrum es la primera etapa en la que se establecen los objetivos del proyecto y se define la visión del producto final. Durante esta fase, el equipo de Scrum trabaja con los interesados para entender sus necesidades y establecer el objetivo del proyecto. (Arrarte, s.f.)

La Visión del Proyecto es esencial para el éxito del proyecto ya que sirve como una guía para la toma de decisiones y ayuda al equipo a mantenerse enfocado en el objetivo final. En resumen, la fase de Visión del Proyecto en Scrum se trata de establecer una comprensión común de lo que se está construyendo, por qué se está construyendo y para quién se está construyendo. (Martins, s.f.)

¿Quiénes están implicados en el proceso?

- **Product Owner**: Marien Crhisty Pérez Avella, profesional de la División de Planeación y Acreditación de la Universidad de Boyacá (encargada de gestionar las tablas que generan las facultades para procesos de autoevaluación).
- **Scrum Master**: Clara Patricia Avella Ibáñez – Ingeniera de sistemas, directora del presente proyecto.
- **Desarrollador**: Estudiante Ingeniería de Sistemas Sergio Iván Martínez, autor del presente proyecto.

- **¿Cuál es el problema por resolver?**

  El problema por resolver es que la Universidad de Boyacá no cuenta con un módulo automatizado que capture la información generada por las facultades, necesaria para los procesos de autoevaluación que se realizan en la institución. Por lo tanto, se necesita desarrollar un sistema de información para la gestión de información de facultades que satisfaga estos requerimientos.

- **¿Qué impacto tiene sobre el negocio (Universidad)?**

  La implementación del sistema de información para la gestión de información de facultades tendría un impacto positivo en la universidad, ya que permitiría capturar, almacenar y consultar información necesaria para los procesos de autoevaluación con fines

de acreditación. Además, esto contribuiría a mejorar los procesos de calidad académica de la universidad.

- **¿Cuál es la propuesta de valor?**

  La propuesta de valor es proporcionar a la División de Planeación y Acreditación y a las diferentes facultades de la Universidad de Boyacá, un sistema informático web que les permita gestionar la información generada por las facultades para los procesos de autoevaluación con fines de acreditación. Esto sería una solución efectiva al problema planteado y contribuiría a los procesos de calidad académica, toda vez que se contará con herramientas automatizadas que apoyen la realización de estos.

- **¿Cuál es la funcionalidad clave?**

  La funcionalidad clave del sistema de información para la gestión de información de facultades es capturar, almacenar y permitir consultas de información necesaria para los procesos de autoevaluación con fines de acreditación en la universidad. Además, este sistema debe ser una infraestructura de software que satisfaga los requerimientos planteados y ser accesible a través de una plataforma web. También es importante que este sistema sea fácil de usar y manejar para todos los usuarios implicados en el proceso.

7.2. Sprints

A continuación, se presentan los Sprints que se realizaron para el desarrollo del sistema de información.

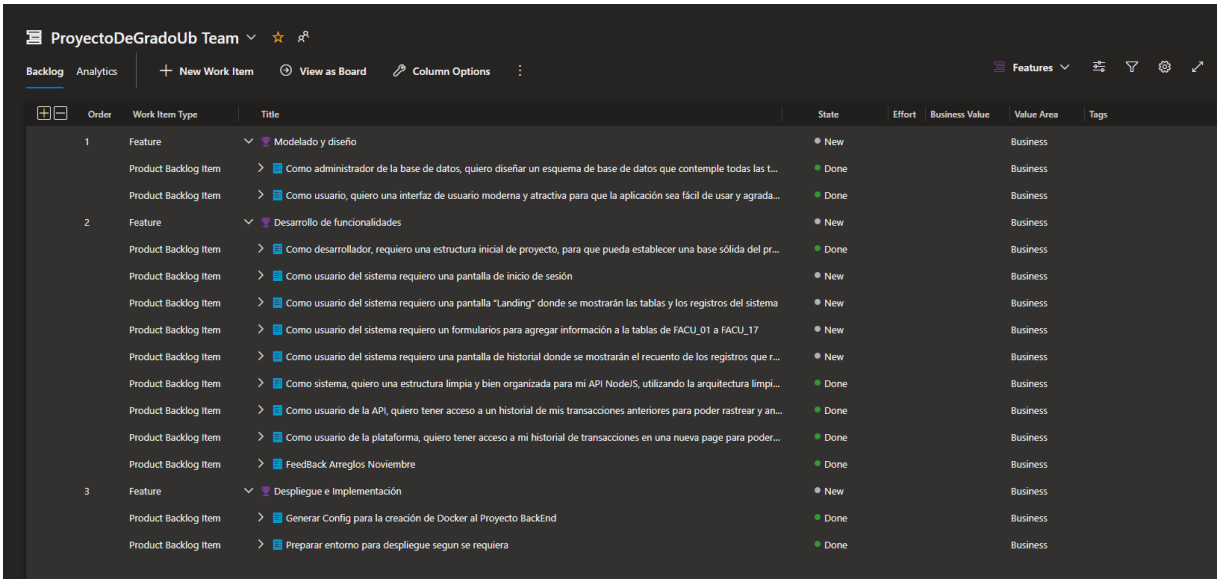
7.2.1. Sprint 0 - planeación

En este Sprint, se realizó el análisis de los requisitos del proyecto junto con el Product Owner y Scrum Master, y se documentaron en forma de historias de usuario, las cuales se pueden encontrar en el (*Anexo 2 " Especificación de Requisitos")*. Esta información se ha incorporado al Product Backlog de Azure, el cual es una herramienta que permite gestionar de manera ordenada y priorizada las funcionalidades que deben desarrollarse durante el proyecto.

Esto es parte de la fase de planificación, en la cual se establecen los objetivos del proyecto y se definen las tareas necesarias para alcanzarlos. Realizar este proceso fue fundamental para tener una comprensión clara de los requisitos del proyecto y planificar de manera efectiva la implementación de las funcionalidades requeridas en los siguientes Sprints.

Figura 25

Sprint 0 – planeación



Order	Work Item Type	Title	State	Effort	Business Value	Value Area	Tags
1	Feature	Modelado y diseño	New			Business	
	Product Backlog Item	Como administrador de la base de datos, quiero diseñar un esquema de base de datos que contemple todas las t...	Done			Business	
	Product Backlog Item	Como usuario, quiero una interfaz de usuario moderna y atractiva para que la aplicación sea fácil de usar y agrada...	Done			Business	
2	Feature	Desarrollo de funcionalidades	New			Business	
	Product Backlog Item	Como desarrollador, requiero una estructura inicial de proyecto, para que pueda establecer una base sólida del pr...	Done			Business	
	Product Backlog Item	Como usuario del sistema requiero una pantalla de inicio de sesión	New			Business	
	Product Backlog Item	Como usuario del sistema requiero una pantalla "Landing" donde se mostrarán las tablas y los registros del sistema	New			Business	
	Product Backlog Item	Como usuario del sistema requiero un formularios para agregar información a la tablas de FACU_01 a FACU_17	New			Business	
	Product Backlog Item	Como usuario del sistema requiero una pantalla de historial donde se mostrarán el recuento de los registros que r...	New			Business	
	Product Backlog Item	Como sistema, quiero una estructura limpia y bien organizada para mi API NodeJS, utilizando la arquitectura limpi...	Done			Business	
	Product Backlog Item	Como usuario de la API, quiero tener acceso a un historial de mis transacciones anteriores para poder rastrear y an...	Done			Business	
	Product Backlog Item	Como usuario de la plataforma, quiero tener acceso a mi historial de transacciones en una nueva page para poder...	Done			Business	
	Product Backlog Item	FeedBack Arreglos Noviembre	Done			Business	
3	Feature	Despliegue e Implementación	New			Business	
	Product Backlog Item	Generar Config para la creación de Docker al Proyecto BackEnd	Done			Business	
	Product Backlog Item	Preparar entorno para despliegue segun se requiera	Done			Business	

Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 0 – Azure backlog

7.2.2. Sprint 1

7.2.2.1. Planificación sprint

- **¿En qué estoy ahora mismo?**

  En este momento el proyecto se encuentra con los requisitos y los objetivos definidos.

- **¿Cuántos días tiene el sprint?**

  15 días

- **¿Dónde quiero estar al final del sprint?**

Al finalizar el sprint se contará con el modelo de base de datos con las entidades de Usuarios, Tablas, Registros, Programas y Facultad, además de diseño y maquetación para la página de Login.

- **¿Con qué personas se cuenta para el Sprint?**

  Estudiante: Sergio Iván Martínez Pulido

- **¿Cuál es el plan?**

Historia de Usuario: Como autor del proyecto, quiero diseñar un esquema de base de datos que contemple todas las tablas y relaciones necesarias para el correcto funcionamiento de la aplicación.

Tareas:

- Crear modelo para usuarios
- Crear modelos de tablas
- Crear modelo para registros de tabla
- Identificar los requisitos de la aplicación
- Establecer las entidades y relaciones
- Normalizar las tablas
- Definir las restricciones y reglas de integridad
- Crear modelo para programas

Historia de Usuario: Como desarrollador, requiero una estructura inicial de proyecto, para que pueda establecer una base sólida del proyecto

Tareas:

- Iniciar proyecto Flutter
- Configurar arquitectura limpia al proyecto, Crear la estructura de carpetas
- Configurar estructura basado en MVVM del proyecto
- Generar configuración de inyección de dependencias

Historia de Usuario: Como usuario del sistema requiero una pantalla de inicio de sesión

Tareas:

- Maquetar Login Page

- Diseño y maquetación de página de Login
- Responsive Login Page
- Se agrega servicio para login

- **Resultados:**

  Azure Backlog – Sprint 1 (*Figura 26*)

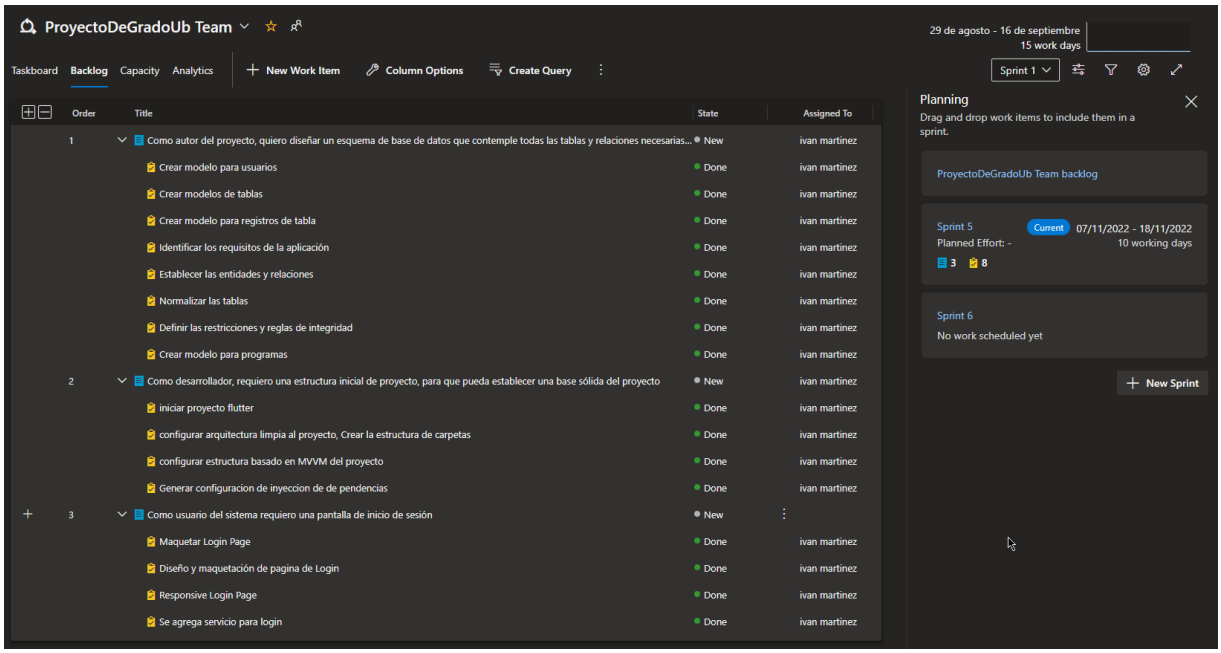
  Modelo de base de datos (*Figura 4*)

  Diseño y maquetación para la página de Login (*Figura 5*)

  Estructura del proyecto Flutter (Figura 27)

Figura 26

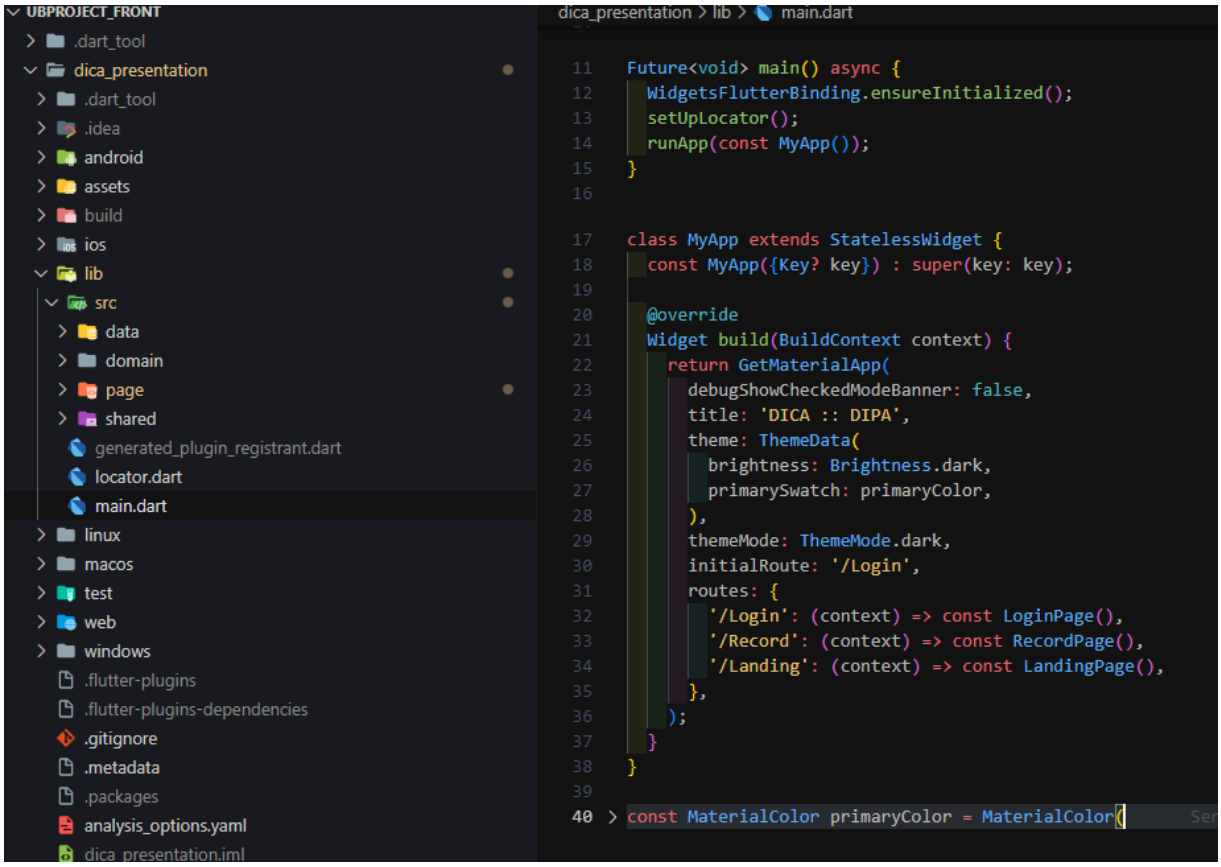
Azure Backlog Sprint 1



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 1 – Azure backlog

Figura 27

Estructura del proyecto Flutter



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 1 – Azure backlog

7.2.3. Sprint 2

7.2.3.1. Planificación sprint

- **¿En qué estoy ahora mismo?**

  En este momento el proyecto cuenta con modelo de base de datos, una estructura de proyecto en Flutter, página de Login diseñada y maquetada en Flutter.

- **¿Cuántos días tiene el sprint?**

  15 días

- **¿Dónde quiero estar al final del sprint?**

  Al finalizar el sprint se contará con la página de “Landing” diseñada y maquetada en Flutter, 5 de los 17 formularios de Registro para las Tablas FACU_01 – FACU_05.

- **¿Con qué personas se cuenta para el Sprint?**

  Estudiante: Sergio Iván Martínez Pulido

- **¿Cuál es el plan?**

  Historia de Usuario: Como usuario del sistema requiero una pantalla "Landing" donde se mostrarán las tablas y los registros del sistema

  Tareas:

  - Header Landing page
  - Maquetar Landing Page
  - Maquetar Widget para Las Tablas según indicador FACU_XX
  - Responsive Landing page
  - Añadir GetX Landing Page (Controller)

  Historia de Usuario: Como usuario del sistema requiero un formulario para agregar información a las tablas de FACU_01 a FACU_17

  Tareas:

  - Maquetar Dialog FACU_01
  - Maquetar Dialog FACU_02
  - Maquetar Dialog FACU_03
  - Maquetar Dialog FACU_04
  - Maquetar Dialog FACU_05

- **Resultados:**

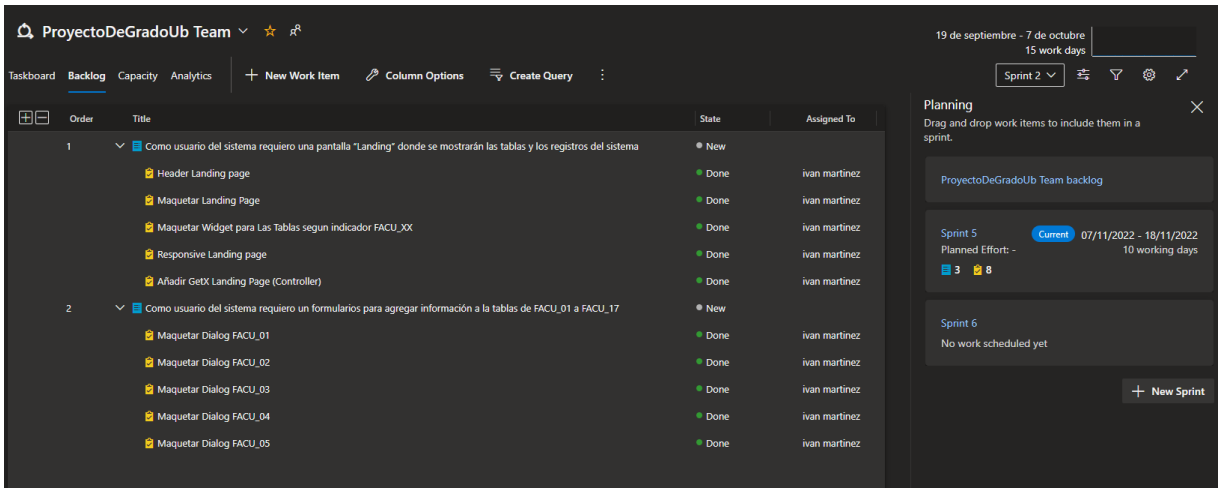
  Azure Backlog – Sprint 2 (*Figura 28*)

  Diseño y maquetación de la Landing Page (*Figuras 7 y 8*)

  Diseño y Maquetación Formularios FACU_01 a FACU_05 (*Figura 29*)

Figura 28

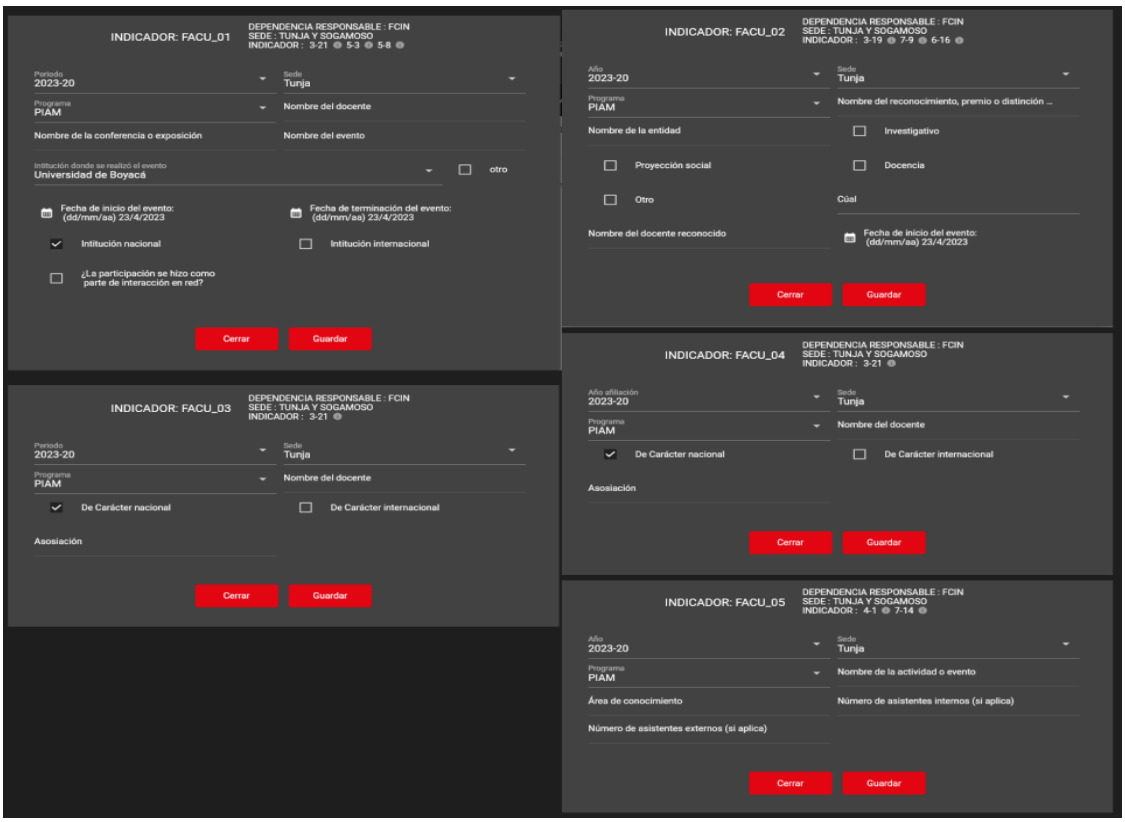
Azure Backlog Sprint 2



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 2 – Azure backlog

Figura 29

Diseño y maquetación formularios FACU_01 a FACU_05



Fuente: Captura de pantalla de los formularios de las tablas FACU_01 a FACU_05

7.2.4. Sprint 3

7.2.4.1. Planificación sprint

- **¿En qué estoy ahora mismo?**

  En este momento el proyecto cuenta con la página de "Landing" diseñada y maquetada en Flutter, 5 de los 17 formularios de Registro para las Tablas FACU_01 – FACU_05.

- **¿Cuántos días tiene el sprint?**

  15 días

- **¿Dónde quiero estar al final del sprint?**

  Al finalizar el sprint se contará con la finalización de los formularios de las tablas FACU_6 a FACU_17, y la conexión a sus servicios al API. Creación proyecto API REST en NodeJS.

- **¿Con qué personas se cuenta para el Sprint?**

  Estudiante: Sergio Iván Martínez Pulido

- **¿Cuál es el plan?**

  Historia de Usuario: Como usuario del sistema requiero un formulario para agregar información a las tablas de FACU_01 a FACU_17

  Tareas:

  - Header Landing page
  - Maquetar Dialogs de FACU_06 a FACU_17
  - agregar servicios REST para el manejo de CRUD de las tablas y los registros

  Historia de Usuario: Como sistema, quiero una estructura limpia y bien organizada para mi API NodeJS, utilizando la arquitectura limpia (Clean Architecture).

  Tareas:

  - Configurar Proyecto inicial
  - Crear modelo para registros según schema de mongoose
  - Crear controlador para registros

- Configurar el entorno de desarrollo y las herramientas necesarias para desarrollar la API
- Crear la estructura básica de la API, incluyendo las carpetas y archivos necesarios para cada capa.
- Crear los modelos de datos y las interfaces necesarias para la capa de dominio
- Implementar la capa de aplicación, incluyendo la lógica de negocio y las interfaces de usuario
- Implementar la capa de infraestructura, incluyendo la conexión a la base de datos y la configuración del servidor.
- Realizar pruebas de rendimiento y seguridad para asegurar la calidad de la API

- **Resultados:**

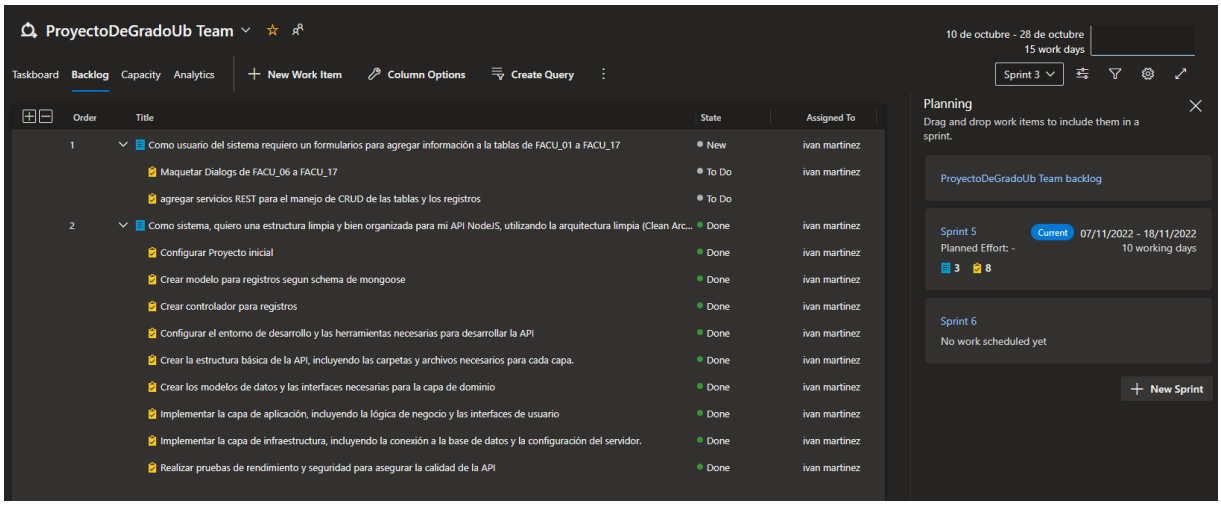
  Azure Backlog – Sprint 3 (*Figura 30*)

  Diseño y Maquetación Formularios FACU_06 a FACU_17 (*Figura 31*)

  Estructura Proyecto NodeJs (*Figura 32*)

Figura 30

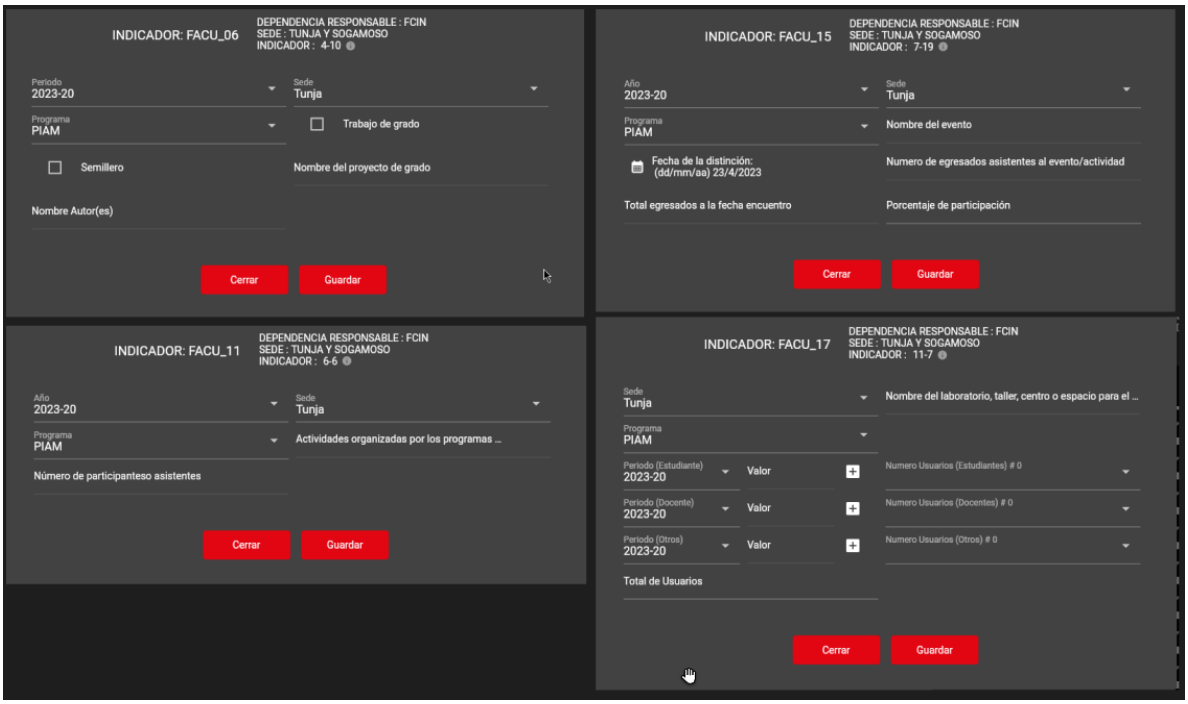
Azure Backlog Sprint 3



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 3 – Azure backlog

Figura 31

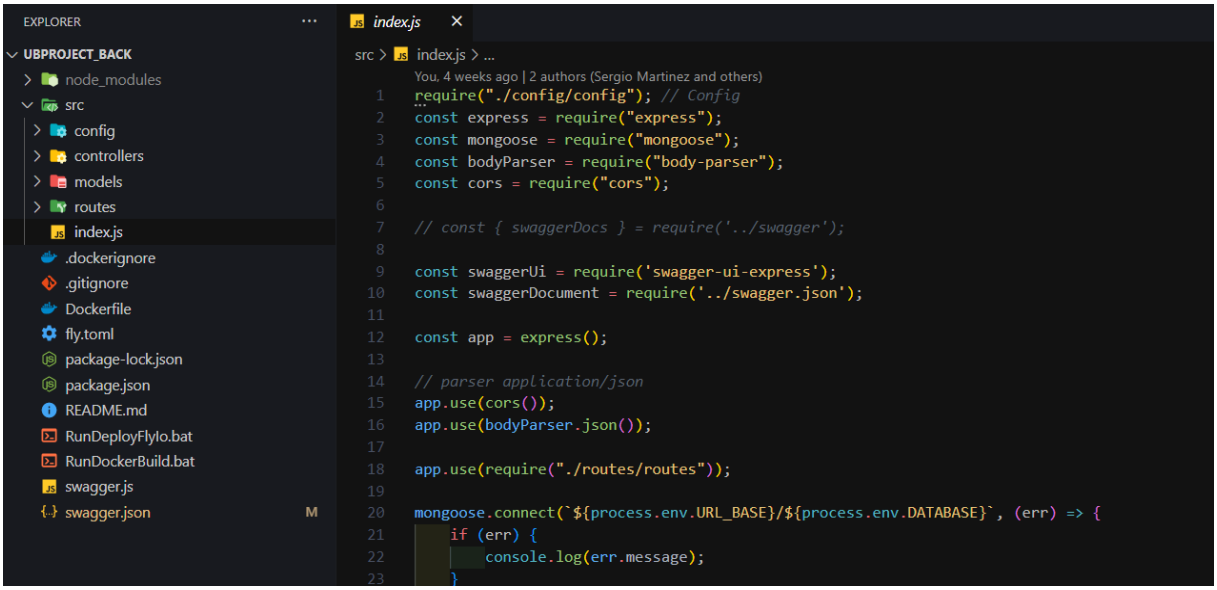
Diseño y maquetación, Tablas FACU_06 a FACU_17



Fuente: Captura de pantalla de algunos formularios de las tablas FACU_06 a FACU_17

Figura 32

Estructura proyecto NodeJs



Fuente: Captura de pantalla Proyecto NodeJs

- **Feedback**

Después de concluir este sprint, el equipo de desarrollo recibió solicitudes por parte del Product Owner para realizar cambios en el diseño, enfocándose principalmente en la página de Landing. Todas las observaciones realizadas por el Product Owner se pueden ver en el (Anexo 4 Feedback de Sprints).

7.2.5. Sprint 4

7.2.5.1. Planificación sprint

- **¿En qué estoy ahora mismo?**

  En este momento el proyecto cuenta con los formularios de las tablas FACU_6 a FACU_17, y la conexión a sus servicios al API. Además de la creación proyecto API REST en NodeJS.

- **¿Cuántos días tiene el sprint?**

  10 días

- **¿Dónde quiero estar al final del sprint?**

Al finalizar el sprint se contará con la solución de un nuevo requisito solicitado por el product owner lo cual es la pantalla de historial, y la configuración en Backend de historial.

- **¿Con qué personas se cuenta para el Sprint?**

  Estudiante: Sergio Iván Martínez Pulido

- **¿Cuál es el plan?**

  Historia de Usuario: Como usuario del sistema requiero una pantalla de historial donde se mostrará el recuento de los registros que realizan los usuarios

  Tareas:

  - Diseñar y maquetar página de historial
  - Maquetar responsive página historial
  - Crear tabla para mostrar historial
  - Agregar servicio REST para obtener historial

Historia de Usuario: Como autor del proyecto, quiero tener acceso a mi historial de transacciones mediante una API

Tareas:

- Agregar nuevo service para agregar los historiales
- Agregar controller para exponer los historiales creados
- Agregar historial a los Endpoint de crear actualizar eliminar registro
- Asignar usuario a servicio de historial

- **Resultados:**

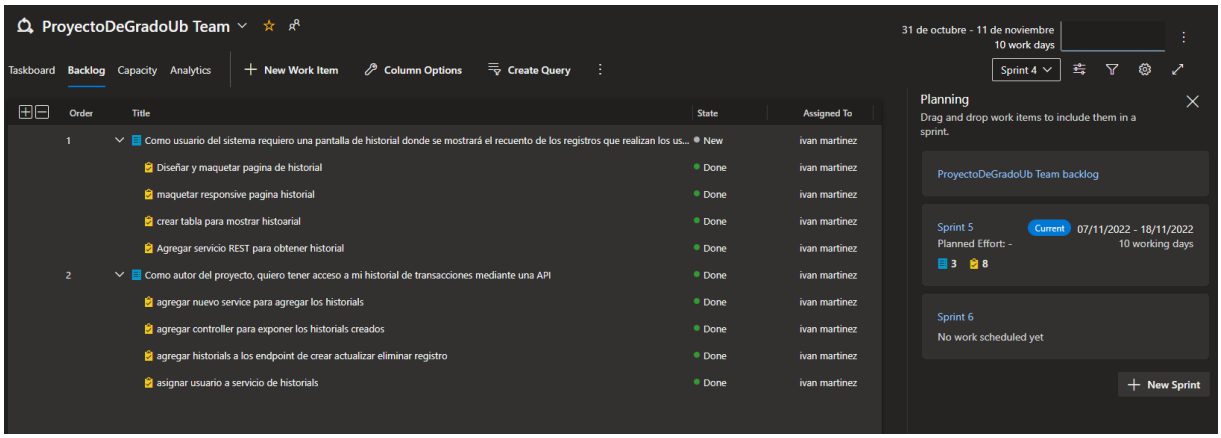
  Azure Backlog – Sprint 4 (*Figura 34*)

  Diseño y Maquetación Página Historial (*Figura 35*)

  Controlador del historial (*Figura 36*)

Figura 33

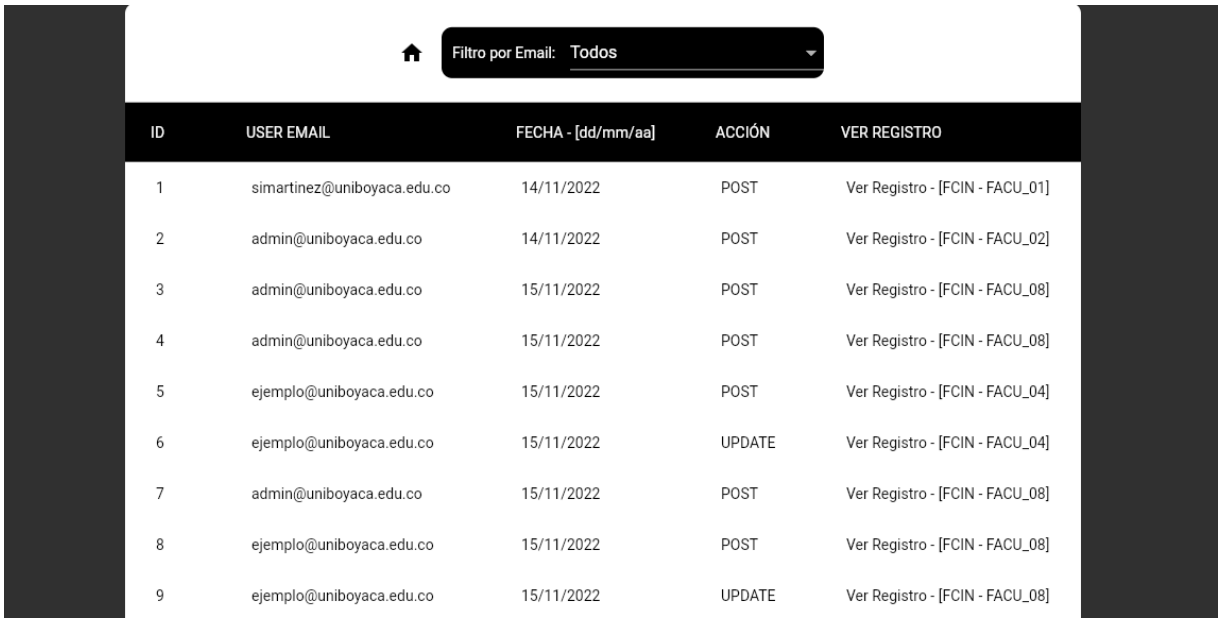
Azure Backlog Sprint 4



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 4 – Azure backlog

Figura 34

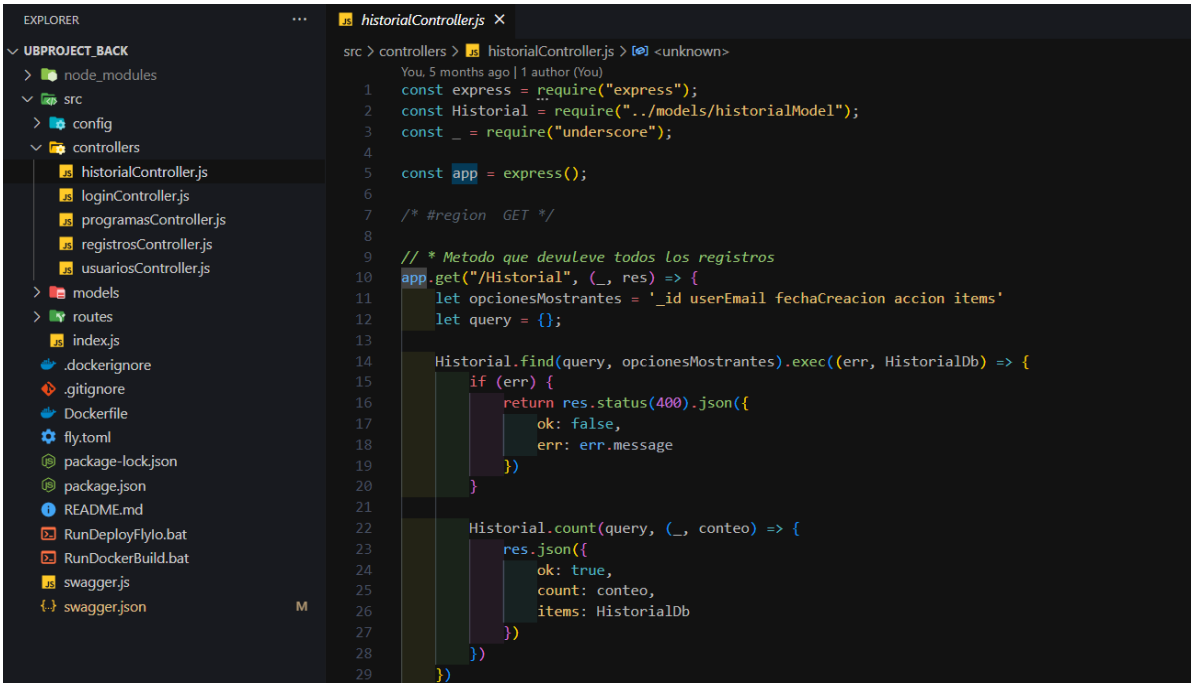
Diseño y Maquetación página historial



Fuente: Captura de pantalla del diseño y maquetación de la página historial

Figura 35

Controlador del Historial



Fuente: Captura de pantalla del del controlador del historial

7.2.6. Sprint 5

7.2.6.1. Planificación sprint

- **¿En qué estoy ahora mismo?**

  En este momento el proyecto cuenta con la pantalla de historial, y la configuración en Backend de historial.

- **¿Cuántos días tiene el sprint?**

  10 días

- **¿Dónde quiero estar al final del sprint?**

  Al finalizar el sprint se contará con feedback presentado por product owner sobre la tabla de registro FACU_08, y filtro de historial. Además, con el despliegue de las aplicaciones web, Backend y base de datos.

- **¿Con que personas se cuenta para el Sprint?**

  Estudiante: Sergio Iván Martínez Pulido

- **Cuál es el plan**

  Historia de Usuario: FeedBack Arreglos Noviembre

  Tareas:

  - Arreglar el de filtro historial
  - Agregar traza de item en Historial
  - Cambiar la tabla FACU_08

  Historia de Usuario: Generar Config para la creación de Docker al Proyecto BackEnd

  Tareas:

  - Agregar Docker file a proyecto de Backend
  - Generar despliegue local usando Docker

  Historia de Usuario: Preparar entorno para despliegue según se requiera

  Tareas:

  - Desplegar backEnd en servicio gratuito Fly.io haciendo uso de Docker
  - Desplegar base de datos mongo en servicio mongoDb atlas

- Desplegar web en servicio de host proporcionado en url programaticos.com

- **Resultados:**

  Azure Backlog – Sprint 5 (*Figura 36*)

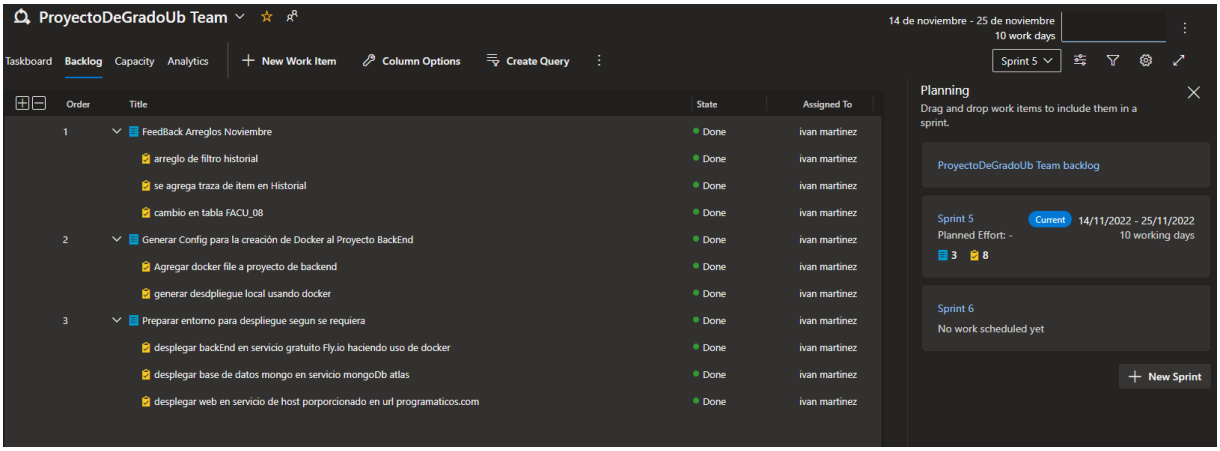
  Despliegue de página en el dominio Programaticos.com (*Figura 37*)

  Despliegue Backend en servicio Fly.io (*Figura 38*)

  Despliegue Base de datos en servicio MongoDb Atlas (*Figura 39*)

Figura 36

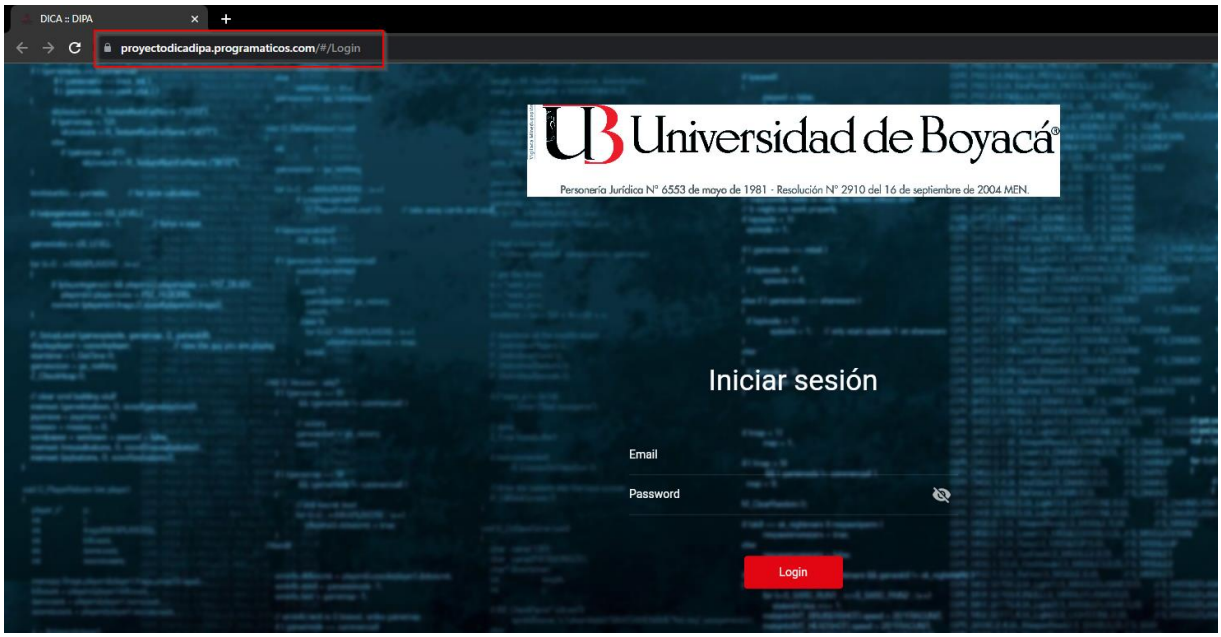
Azure Backlog Sprint 5



Fuente: Captura de pantalla de las historias de usuario identificadas en el Sprint 5 – Azure backlog

Figura 37

Despliegue de página en el dominio Programaticos.com

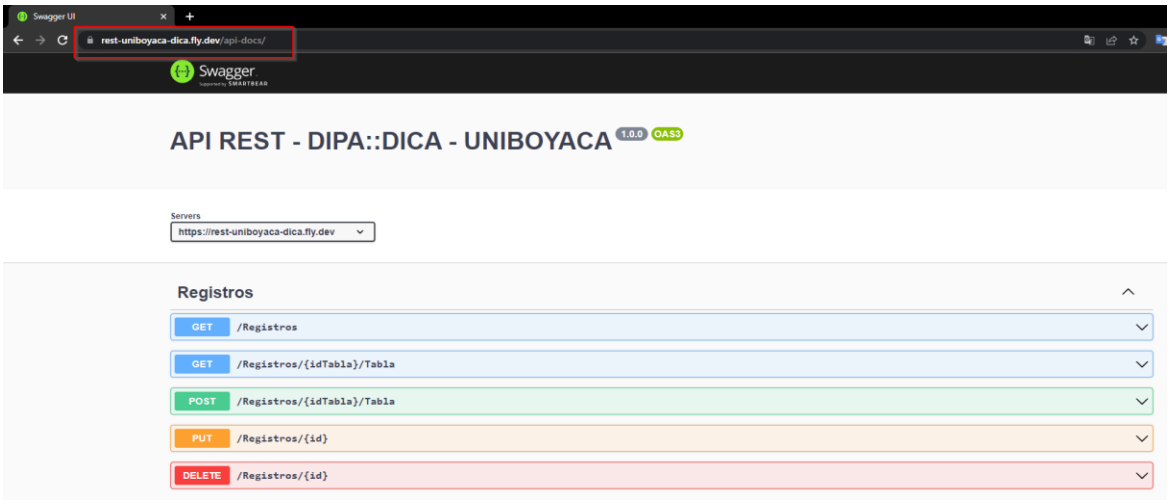


Fuente: Captura de pantalla del proyecto en funcionamiento

Nota: Representa el aplicativo web desplegado en el subdominio *proyectodicadipa.programaticos.com*, para el mismo se configuro el certificado SSL para su acceso sin restricciones desde la web.

Figura 38

Despliegue backend en servicio Fly.io

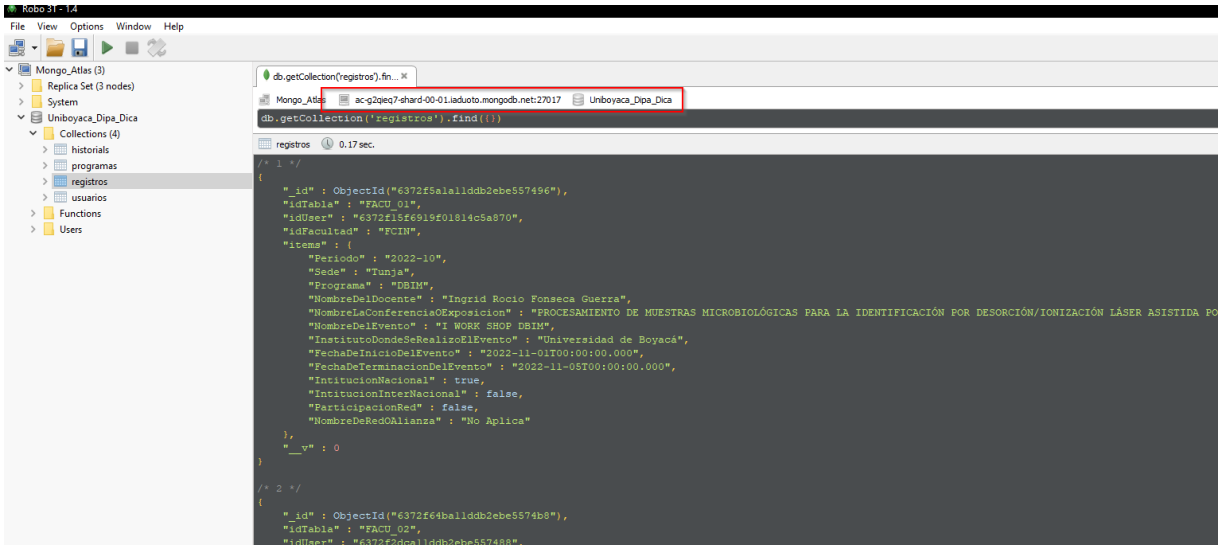


Fuente: Captura de pantalla del servicio Fly.io del proyecto

Nota: La figura representa el servidor web que aloja el Backend de la aplicación. Este servidor expone una página web con el servicio de Swagger, el cual permite probar los Endpoints del API Rest sin necesidad de configurar un servicio externo adicional. Es importante destacar que esta página se utiliza exclusivamente para la documentación del API Rest.

Figura 39

Despliegue Base de datos en servicio - MongoDb Atlas



Fuente: Captura de pantalla de la Base de Datos MongoDb en servicio

Nota: En la figura se muestra la base de datos del aplicativo en el gestor Robo 3T. Se encuentra subrayado en rojo el String de conexión a la base de datos, el cual puede ser accedido de manera remota desde cualquier PC. El gestor Robo 3T nos ayuda a interactuar con la base de datos de manera mucho más intuitiva.

En este capítulo se han explorado los aspectos fundamentales relacionados con el inicio de este proyecto y la creación de su Envisioning. Asimismo, se ha profundizado en la metodología ágil Scrum, centrándose en los Sprints y los valiosos resultados obtenidos a lo largo de este proceso. Además, se ha destacado la importancia de la retroalimentación por parte de los Product Owners, quienes han desempeñado un papel clave en la evolución del proyecto. Por último, se ha ahondado en las buenas prácticas del desarrollo ágil, resaltando cómo Scrum se ha convertido en una herramienta invaluable para impulsar la eficiencia y la calidad en el desarrollo de software.

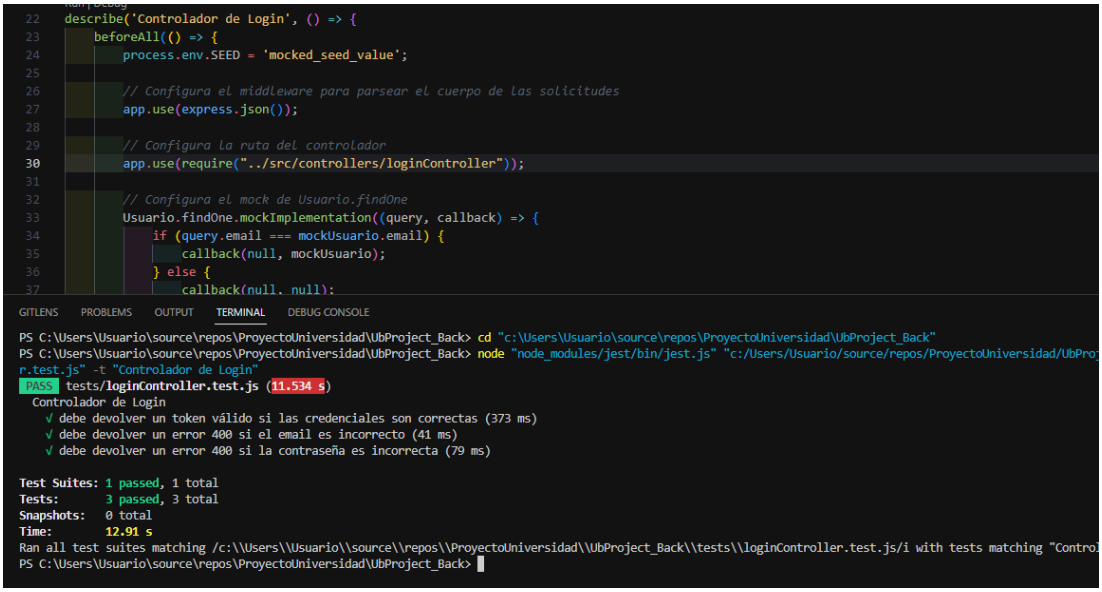
7.3. Pruebas unitarias

Una prueba unitaria es una forma de probar una unidad, la pieza más pequeña de código que se puede aislar lógicamente en un sistema. En la mayoría de los lenguajes de programación, eso es una función, una subrutina, un método o propiedad. (smartbear, s.f.)

Para este proyecto se definieron test unitarios tanto en el Font como en el Backend, los mismos se realizaron con librerías nativas de cada lenguaje como lo son mockito (Flutter) y jest (NodeJs). Estos test se plantearon para poder que verificar que el código haga lo que se supone que se espera de el a lo largo del tiempo, como lo puede ser el controlador de Login el cual se evidencia en la *Figura 40,* o de la petición de historial que se pide desde la página web como se evidencia en la *Figura 41*.

Figura 40

Test unitarios controlador login

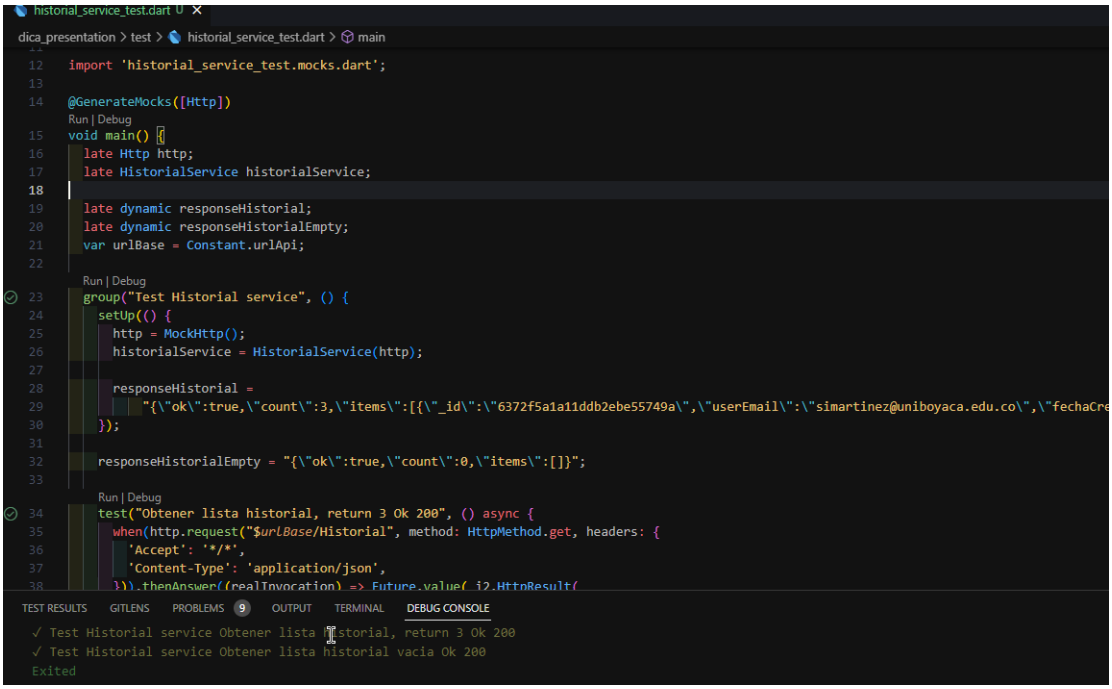


Fuente: Captura de pantalla código fuente Api Rest NodeJs

Nota: En la figura se muestra la implementación de test unitarios de tres casos posibles para el controlador de login en el Api Rest.

Figura 41

Test unitarios petición historial flutter



Fuente: Captura de pantalla código fuente Frontend Flutter

Nota: En la figura se muestra la implementación de test unitarios de dos casos posibles para la petición de historial en la página web, lo que se hace acá es generar una respuesta falsa y con la misma verificar que las líneas de código de la petición funcionen como se espera.

Durante el periodo de desarrollo se pudo evidenciar ciertos problemas de ejecución en preproducción, problemas que a posterior se gastó más tiempo en resolver. Al empezar a usar TDD (Desarrollo guiado por pruebas), dichos problemas fueron disminuyendo debido a que antes de crear la función ya se tiene abordado todos los casos posibles de dicha función. Esto es muy útil ya que se mitiga la posibilidad de que algo falle cuando llegue a produccion.

8. Conclusiones

Durante la realización de este proyecto de grado, se logró plasmar el conocimiento adquirido durante toda la formación académica en la resolución de un problema real. Además, fue posible adaptar las nuevas tecnologías del mercado para utilizarlas en el desarrollo del aplicativo.

La generación del aplicativo FrontEnd utilizando Flutter Web permitió crear una interfaz de usuario intuitiva y atractiva, brindando una experiencia de usuario óptima en la aplicación web. Asimismo, la generación del Web Service BackEnd utilizando Node.js, junto con la documentación en Swagger, aseguró la disponibilidad de una API robusta y bien documentada, facilitando la integración entre los componentes. Adema de esto el desafío del despliegue de los tres componentes de proyecto web, api y base de datos ofreció un buen punto de referencia para el entorno laboral.

La implementación del marco de trabajo SCRUM fue una de las claves del éxito del proyecto, porque ofrece técnicas y prácticas que ayudan a gestionar el desarrollo de software de manera ágil y eficiente, lo que permitió obtener los resultados esperados. La adaptación de esta metodología a un solo desarrollador permitió conocer en detalle todo el flujo de trabajo, lo que resultó en un alto desempeño y una mejor comprensión de la metodología.

La implementación de tecnologías de gran uso en la actualidad fue una de las fortalezas del proyecto, como lo fueron MongoDB, Flutter y NodeJS, y la documentación para su uso fue de fácil acceso, de tal manera que esto permitió su implementación sin contratiempos para el desarrollo del aplicativo.

Referencias

AppMaster. (2022, 19 de octubre). *Guía completa de diseño de bases de datos y herramientas* https://appmaster.io/es/blog/diagrama-de-diseno-de-la-base-de-datos-guia-completa

Amazon. (s.f.). *¿Qué es docker?*. https://aws.amazon.com/es/docker/

Arrarte, A. (s.f.). *Cómo usar las 5 fases de la metodología scrum en tus proyectos para mejorar la efectividad*. https://alvaroarrarte.com/fases-de-la-metodologia-scrum/

CodeAcademy. (s.f.). *What is CRUD?*. https://www.codecademy.com/article/what-is-crud

Couchbase. (s.f.). *NoSQL Databases, Why successful enterprises rely on NoSQL*. https://www.couchbase.com/resources/why-nosql

Fitzgibbons, L. (2019). *Front-end and Back-end.* https://www.techtarget.com/whatis/definition/front-end#:~:text=The%20back%20end%20refers%20to,one%20or%20more%20programming%20languages

Flutter. (s.f.). *Build apps for any screen*. https://flutter.dev/

Ionos. (s.f.). *Web Services, Services from machine to machine.* https://www.ionos.com/digitalguide/websites/web-development/web-services/

Kaspersky. (s.f.). *Qué es un certificado SSL:* https://latam.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate

Kerner, S. M. (s.f.). *What is FTP (File Transfer Protocol)?*. https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP

Martins, J. (s.f.). *¿Qué es scrum? Una guía completa para principiantes.* https://asana.com/es/resources/what-is-scrum

Microsoft. (s.f.). *What is azure?*. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/

MongoDB. (s.f.). *¿Qué es MongoDB?*. https://www.mongodb.com/es/what-is-mongodb

NodeJS. (s.f.). *About Node.js*. https://nodejs.org/en/about

proyectosagiles.org. (2021, Noviembre). *Lista de objetivos / requisitos priorizada (Product Backlog).* https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog/

Red Hat. (2020, 8 de mayo). *What is a REST API?:* https://www.redhat.com/en/topics/api/what-is-a-rest-api

Rouse, M. (2020,31 de agosto). *What Does Web Development Mean?*. https://www.techopedia.com/definition/23889/web-development

Scrum.org. (s.f.). *Scrum Glossary*. https://www.scrum.org/resources/scrum-glossary

Scrum.org. (s.f.). *What is a Product Backlog?* https://www.scrum.org/resources/what-is-a-product-backlog

Smartbear. (s.f.). *What Is Unit Testing?*. https://smartbear.com/learn/automated-testing/what-is-unit-testing/

Swagger.io. (s.f.). *What Is Swagger?*. https://swagger.io/docs/specification/2-0/what-is-swagger/

Unbounce. (s.f.). *What Is a Landing Page*?. https://unbounce.com/landing-page-articles/what-is-a-landing-page/

W3schools. (s.f.). *What is JSON?* https://www.w3schools.com/whatis/whatis_json.asp