ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRÍCA Y ELECTRÓNICA

DESARROLLO DE PROTOTIPO SOFTWARE PARA LA GESTIÓN DE LA BOLSA DE EMPLEOS DE UNA JUNTA PARROQUIAL

DESARROLLO DE UN PROTOTIPO DE APLICATIVO MÓVIL PARA LA GESTIÓN DE LA BOLSA DE EMPLEOS DE UNA JUNTA PARROQUIAL

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGIAS DE LA INFORMACIÓN

ERNESTO JAVIER ANDRADE SALGADO

ernesto.andrade@epn.edu.ec

DIRECTOR: XAVIER ALEXANDER CALDERÓN HINOJOSA

xavier.calderon@epn.edu.ec

DMQ, abril 2024

CERTIFICACIONES

Yo, ERNESTO JAVIER ANDRADE SALGADO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

_____

ERNESTO JAVIER ANDRADE SALGADO

Certifico que el presente trabajo de integración curricular fue desarrollado por ERNESTO JAVIER ANDRADE SALGADO, bajo mi supervisión.

_____

XAVIER ALEXANDER CALDERÓN HINOJOSA

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

_____

ERNESTO JAVIER ANDRADE SALGADO

_____

XAVIER ALEXANDER CALDERÓN HINOJOSA

DEDICATORIA

Este trabajo se lo dedico a mis padres, Sofía Salgado y Fredy Andrade, quienes, gracias a su esfuerzo, dedicación y paciencia, han logrado impulsarme día a día a conseguir mis metas. Este logro no es más que un reflejo de todos y cada uno de los días que ustedes han velado por mí.

A mis amigos, quienes han sido participes de innumerables recuerdos llenos de felicidad durante este largo trayecto de convertirnos en profesionales.

AGRADECIMIENTO

A mis padres, por todo el cariño y apoyo que me han brindado durante mi vida. Es gracias a su ejemplo y enseñanzas que llegado a ser quien soy.

Deseo expresar mi profundo agradecimiento a mis amigos Samuel y Joseph, quienes han sido un apoyo constante en mi vida universitaria, compartiendo momentos de alegría y felicidad, así como apoyándome en situaciones de presión. Su amistad ha enriquecido enormemente mi experiencia académica.

A mi amiga y compañera en este proyecto, así como en gran parte de mi vida, quiero expresarle mi sincero agradecimiento. Tu amistad, apoyo y respaldo han sido fundamentales tanto en mí vida académica como personal.

Por último, deseo expresar mi gratitud a mi director de Trabajo de Integración Curricular, Xavier Calderón, quien ha demostrado una notable paciencia y ha brindado un seguimiento constante, ofreciendo valiosos consejos y orientación a lo largo de la elaboración de este trabajo.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente Trabajo de Integración Curricular tiene como propósito general el diseño e implementación de un prototipo para la gestión de la bolsa de empleos de una Junta Parroquial. Este prototipo está conformado por una aplicación móvil que consume los servicios de una REST API empleando métodos HTTP. El presente documento se encuentra conformado por tres capítulos:

En el primer capítulo se abordan conceptos relacionados a las aplicaciones móviles, la metodología Kanban y el patrón de diseño MVC. Adicionalmente, se proporciona información detallada sobre las herramientas y tecnologías que se utilizaran en las diferentes fases componen el proceso de desarrollo del prototipo.

En el segundo, el cual se encuentra dividido en dos apartados, se detalla el proceso de Diseño y la Implementación del prototipo. En la sección de Diseño, se describe el proceso de obtención de requisitos funcionales y no funcionales, se establecen los roles de usuario, así como sus privilegios, se diseñan módulos del aplicativo y la RES API. Por otro lado, la Implementación implica la codificación de lo previamente definido en la fase de Diseño.

El último capítulo, principalmente, está enfocado en la realización de pruebas para verificar el cumplimiento de los requisitos funcionales y no funcionales, así como de los objetivos establecidos para el prototipo. Adicionalmente, se exponen las conclusiones y recomendación obtenidas al realizar este Trabajo de Integración Curricular.

PALABRAS CLAVE: Modelo-Vista-Controlador, REST API, aplicación móvil, Kanban, Junta Parroquial, Flutter, Mongo DB.

ABSTRACT

The general purpose of this Curricular Integration Project is the design and implementation of a prototype for the management of the employment exchange of a Parish Council. This prototype consists of a mobile application that makes use of the services of a REST API using HTTP methods. This document consists of three chapters:

The first chapter tackles concepts related to mobile applications, the Kanban methodology and the Model-View-Controller design pattern. Additionally, detailed information on the tools and technologies is provided, which will be used in the different phases of the prototype development process.

The second one, which is divided into two sections, details the Design and Implementation process of the prototype. The Design section describes the process of obtaining functional and non-functional requirements, establishing user roles and privileges, designing application modules and the REST API. On the other hand, the Implementation involves the codification of what was previously defined in the Design phase.

The last one is mainly focused on testing to verify compliance with functional and non-functional requirements, as well as with the objectives established for the prototype.

KEYWORDS: Model-View-Controller, REST API, mobile application, Kanban, Parish Council, Flutter, Mongo DB.

1. INTRODUCCIÓN

En un mundo cada vez más conectado y en constante evolución, la búsqueda y oferta de empleo se ha convertido en un aspecto esencial de la vida moderna. La gestión eficiente de la bolsa de empleo es crucial para facilitar la inserción laboral y fomentar el desarrollo económico de una comunidad. En este contexto, el presente trabajo tiene como objetivo central la concepción y creación de un prototipo de aplicación móvil diseñado específicamente para optimizar la gestión de la bolsa de empleo en una Junta Parroquial.

Este proyecto surge de la necesidad de modernizar y agilizar el proceso de búsqueda y publicación de oportunidades laborales dentro de una comunidad local. La implementación de esta solución tecnológica busca brindar a los residentes y empresas de la Junta Parroquial una plataforma accesible y eficaz para conectarse en el ámbito laboral. La aplicación móvil se desarrollará con un enfoque centrado en el usuario, incorporando funcionalidades clave para satisfacer las demandas de empleadores y solicitantes de empleo por igual.

A lo largo de este trabajo, exploraremos en detalle el proceso de diseño y desarrollo de esta aplicación, abarcando desde la identificación de requisitos hasta la implementación y pruebas del prototipo.

Este trabajo representa un paso hacia adelante en la modernización de la gestión de empleo local, alineando la tecnología con las necesidades de una comunidad en constante evolución.

1.1. Objetivo general

Desarrollar un aplicativo móvil que permita la gestión y publicación de la bolsa de empleos de una Junta Parroquial.

1.2. Objetivos específicos

1. Diseñar los módulos del aplicativo móvil a partir de entrevistas con los miembros de la Junta Parroquial.
2. Implementar las interfaces de los módulos de la aplicación móvil.
3. Realizar pruebas a los diferentes módulos desarrollados del aplicativo móvil.

1.3. Alcance

Los usuarios del aplicativo contarán con una fuente de información que será actualizada de forma continua con datos acerca de nuevas vacantes de empleos que se hagan

públicos dentro de la Junta Parroquial. Para ello, se implementarán las siguientes funcionalidades en cada uno de los diferentes módulos con los que contará el aplicativo:

Módulo Login:

- Permitirá a los usuarios registrarse e iniciar sesión de forma segura.
- Se manejarán tres tipos de usuarios: administradores, empresas y usuarios. Los cuales tendrán diferentes niveles de privilegios.

Al contar con tres tipos de usuarios dentro de la aplicación, se manejarán diferentes niveles de privilegios para cada uno, como se detalla a continuación para cada módulo:

Módulo de empleos:

Usuario tipo Administrador:

- Podrá visualizar y eliminar los empleos disponibles dentro de la aplicación.
- Podrá visualizar una lista de postulaciones a los empleos.
- Podrá visualizar los perfiles de empresas.

Usuario tipo Empresa:

- Podrán publicar nuevos empleos, los cuales serán visibles para la empresa que los publicó, los administradores y los usuarios que no pertenezcan al tipo de empresa. Además, únicamente la empresa creadora y los usuarios con el rol de Administrador tendrán la capacidad de realizar modificaciones o eliminar dichos empleos.
- Podrá visualizar la lista de postulaciones que hayan recibido sus empleos.
- Podrá visualizar los perfiles de los usuarios que hayan postulado a sus empleos y tomar la decisión de aceptar o negar dichas postulaciones adicionando el motivo de rechazo.

Usuario tipo cliente:

- Visualizar la lista de empleos disponibles.
- Postular a los empleos disponibles.

Módulo de datos de usuarios

Usuario tipo Administrador:

- Podrá crear, visualizar, actualizar y eliminar los usuarios y empresas registradas.
- Podrá visualizar los perfiles de todos los usuarios y empresas registradas.

Usuario tipo Empresa:

- Podrá visualizar y editar la información que se presentará en su perfil.
- Podrá visualizar los perfiles de los demás usuarios y empresas que se encuentren registrados.

Usuario cliente:

- Podrá visualizar y editar la información que se encontrará expuesta en su perfil.
- Podrá visualizar los perfiles de los demás usuarios y empresas que se encuentren registrados.

Módulo de reportes

Este módulo contará con las siguientes funcionalidades y solo estará disponible para el usuario Administrador:

- Se mostrará un resumen del número de usuarios y empresas registrados, así como el número de postulaciones en función del estado en el que se encuentren.
- Se mostrarán gráficos estadísticos que representen la cantidad de postulaciones en espera, aprobadas, y rechazadas, así como los motivos de rechazo

Para la implementación del aplicativo, se tendrán en cuenta las siguientes fases:

Fase de diseño

- En esta fase se llevará a cabo la recolección de los requerimientos del aplicativo por medio de reuniones y entrevistas con la presidente de la Junta Parroquial.
- Se analizarán los requerimientos proporcionados y se elaborarán diagramas UML para explicar de forma más detallada el funcionamiento de la aplicación.
- Se definirá la arquitectura del sistema, incluyendo la estructura del proyecto basado en el stack MERN (Mongodb, Express.JS, React, Node.JS) y la configuración de la base de datos relacional, así como la utilización del modelo MVC.
- Se estructurarán las API REST que permitirán la comunicación entre el backend y el frontend.
- Se bocetearán las pantallas y flujos de la aplicación utilizando Flutter, considerando la integración de los estilos de Bootstrap para una apariencia más atractiva y coherente.
- Todo este proceso de diseño estará reflejado en un tablero Kanban, permitiendo una gestión visual del progreso y la asignación eficiente de tareas.

Fase de implementación

Una vez definido el diseño, se procederá a la implementación.

- En el proyecto basado en el stack MERN, se codificarán los controladores y modelos necesarios para gestionar las solicitudes y respuestas de la API REST.
- Se configurará la base de datos relacional y se crearán las tablas y relaciones pertinentes para almacenar los datos de la aplicación.
- En el frontend, se utilizará el lenguaje Dart y el framework Flutter para construir los módulos y la lógica de la interfaz de usuario y se aplicarán los principios del modelo MVC para estructurar el código. Aquí es donde se integrarán los estilos de Bootstrap para lograr una apariencia visual atractiva y responsive.

Fase de pruebas

Se llevarán a cabo las siguientes pruebas:

- En el backend, se llevarán a cabo pruebas para asegurarse de que la API REST funcione correctamente y maneje adecuadamente los datos.
- En el frontend, se llevarán a cabo pruebas de funcionamiento para comprobar que el aplicativo funciones correctamente y cumpla con los requerimientos solicitados.

1.4. MARCO TEÓRICO

1.4.1. APLICACIONES MÓVILES [1]

La sociedad actual está experimentando un cambio significativo impulsado por la telefonía móvil, similar al impacto que Internet tuvo en su momento. Esta revolución está en sus inicios, ya que los modernos dispositivos móviles ofrecen capacidades comparables a las de un ordenador personal, permitiendo a los usuarios acceder al correo electrónico y navegar por Internet. La gran diferencia radica en que un teléfono móvil siempre está al alcance del usuario en su bolsillo, lo que abre un nuevo mundo de aplicaciones cercanas y accesibles. De hecho, muchos expertos concuerdan en que el terminal móvil se perfila como el ordenador personal más relevante del siglo veintiuno.

1.4.1.1. TIPOS DE APLICACIONES MÓVILES

1.4.1.1.1. APLICACIONES NATIVAS

Las aplicaciones nativas son desarrolladas con un lenguaje y entorno de programación específicos, garantizando un rendimiento óptimo en el sistema operativo para el cual fueron diseñadas. Por ejemplo, las aplicaciones de iOS se crean con Objective-C, las de Android con Java y las de Windows con .Net. Cada uno de estos lenguajes y entornos está optimizado para su respectivo sistema operativo, asegurando un funcionamiento eficiente en cada plataforma.

1.4.1.1.2. APLICACIONES WEB

Las aplicaciones web son ampliamente utilizadas para garantizar accesibilidad desde cualquier dispositivo, independientemente del sistema operativo. Su principal ventaja radica en que solo se requiere un navegador web para acceder a ellas, permitiendo a los usuarios utilizar la aplicación desde cualquier lugar con conexión a internet. Esta facilidad de acceso y distribución sin necesidad de instalación específica las convierte en una opción popular. Los lenguajes más comunes empleados en su desarrollo son HTML, CSS y JavaScript.

1.4.1.1.3. APLICACIONES HÍBRIDAS

Las aplicaciones híbridas combinan elementos de aplicaciones nativas y web según sea necesario. Su principal ventaja es la flexibilidad en el desarrollo, ya que no requieren un entorno específico y las herramientas utilizadas son mayormente gratuitas. Estas aplicaciones pueden integrarse con herramientas de desarrollo nativas, brindando a los desarrolladores opciones adicionales. La versatilidad permite aprovechar lo mejor de ambos enfoques y adaptarse eficientemente a diversas necesidades y plataformas. Utilizando lenguajes web como JavaScript, CSS3 y HTML5, estas aplicaciones garantizan un funcionamiento uniforme en varios sistemas.

1.4.2. GIT Y GITHUB [2]

Git es un sistema de control de versiones avanzado y distribuido, similar al "control de cambios" de Microsoft Word. A lo largo del tiempo, realiza "capturas" que registran los cambios en un proyecto, permitiendo visualizar las modificaciones, quién las hizo y por qué. Esto facilita retroceder a versiones anteriores. Git favorece el trabajo colaborativo al permitir que múltiples participantes trabajen de forma paralela, creando y fusionando sus propias "capturas". A diferencia de otros sistemas como SVN o CVS, Git no depende de un servidor central, y aunque existen alternativas como Mercurial o Bazaar, Git destaca como la opción más ampliamente utilizada.

GitHub es una plataforma de alojamiento en línea especializada en repositorios Git, destinada a albergar proyectos y fomentar la colaboración entre usuarios. Un repositorio, que es un directorio que contiene todos los archivos necesarios para el desarrollo de un proyecto, se utiliza para gestionar el progreso de manera remota y compartirlo entre diferentes usuarios. Aunque existen otras opciones similares, como GitLab o Bitbucket, GitHub destaca como la elección más ampliamente utilizada en la actualidad.

Esta plataforma registra de forma remota el progreso de los proyectos, permitiendo compartirlos y ofreciendo funciones de seguridad basadas en la nube, entre otras ventajas. En proyectos colaborativos, la interacción entre Git y GitHub implica que

GitHub sirve como el repositorio central, donde se almacenan y gestionan las distintas versiones del proyecto durante su desarrollo colaborativo.

1.4.3. METODOLOGÍA ÁGIL

En la actualidad, el mundo empresarial se desenvuelve en un entorno global en constante cambio. Por lo tanto, es fundamental responder de manera ágil a las nuevas necesidades y oportunidades del mercado, considerando que el software juega un papel crucial en prácticamente todas las operaciones empresariales. Es necesario desarrollar soluciones informáticas con flexibilidad para ofrecer respuestas de calidad en todo lo que se requiere.

Las metodologías ágiles se caracterizan principalmente por su enfoque en la flexibilidad. Los proyectos en desarrollo se subdividen en tareas más pequeñas, lo que conlleva una comunicación constante con el usuario y un alto grado de colaboración. Estas metodologías son altamente adaptables a los cambios y, de hecho, la capacidad de cambiar los requerimientos por parte del cliente es una característica especial. También se enfocan en realizar entregas periódicas y mantener una revisión y retroalimentación constante para asegurar que el desarrollo se ajuste a las necesidades en evolución [3].

1.4.3.1. METODOLOGÍA KANBAN

Esta metodología se basa en la aplicación de los procesos Just inTime, desarrollados por Toyota, en los que se utilizan tarjetas visuales con la finalidad de identificar la falta de materiales dentro de la cadena de producción.

La metodología Kanban emplea un sistema de control visual que permite realizar un seguimiento del trabajo a medida que avanza a lo largo del flujo de valor. Por lo general, se emplea un tablero o pizarra con notas adhesivas, o bien un panel electrónico de tarjetas, para gestionar el flujo de trabajo y las asignaciones. Las mejores prácticas recomiendan combinar ambos métodos para obtener mayores beneficios [4].

Los principios fundamentales de la metodología Kanban incluyen la priorización de la calidad sobre la velocidad, resaltando la importancia de hacer las cosas correctamente desde el principio para evitar correcciones futuras, así como la minimización de tareas superfluas para enfocarse en lo esencial. Se promueve una mejora continua, ajustando los procesos según los objetivos establecidos para optimizar el rendimiento y la eficiencia. La flexibilidad es clave, permitiendo adaptarse a las necesidades cambiantes y priorizar tareas según el momento, lo que facilita una gestión ágil. Además, se subraya la necesidad de construir y mantener relaciones sólidas a largo plazo con los

proveedores para garantizar una colaboración estable y confiable en la provisión de recursos y servicios [5].

Reglas

La metodología Kanban se fundamenta en un conjunto de seis reglas, las cuales se encuentran detalladas en la Figura 1.1.

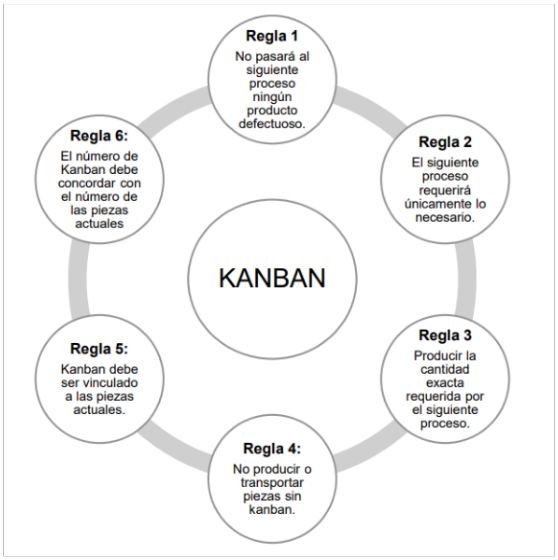

Figura 1.1 Reglas de la metodología Kanban

Estas reglas son fundamentales para el éxito de la metodología Kanban, ya que permiten una gestión ágil y eficiente del trabajo, fomentando la mejora continua y la colaboración entre los miembros del equipo [6].

Tablero Kanban

El uso del tablero Kanban es un aspecto central y esencial en la metodología Kanban, ya que brinda una visión clara y transparente del flujo de trabajo en todo el proceso de desarrollo. Al emplear este tablero, los equipos pueden tener una comprensión detallada de cada etapa del trabajo y de cómo avanzan las tareas en tiempo real.

Adicionalmente, el tablero Kanban promueve la comunicación clara y la colaboración efectiva entre los miembros del equipo. Al estar el estado del trabajo expuesto a todos, se pueden identificar tareas que necesitan atención o asistencia adicional, fomentando la transparencia y la responsabilidad [7]. En la Tabla 1.1 se describe la estructura básica de un tablero Kanban:

Tabla 1.1 Estructura de un tablero Kanban

POR HACER	EN PROGRESO	FINALIZADO
Actividades planeadas	Actividades que están siendo ejecutadas	Actividades que han sido concluidas

1.4.4. TECNOLOGÍAS Y HERRAMIENTAS

1.4.4.1. PATRÓN MODELO VISTA CONTROLADOR [8]

El patrón Modelo-Vista-Controlador es una arquitectura de software que se utiliza para diseñar aplicaciones de forma organizada y eficiente. MVC divide una aplicación en tres componentes principales:

Modelo: Es responsable de representar y gestionar los datos, controlando sus transformaciones y manipulaciones. No tiene conocimiento específico sobre Controladores o Vistas, y el sistema establece enlaces entre el Modelo y las Vistas, notificando cambios.

Vista: Muestra visualmente los datos representados por el Modelo al usuario. Prefiere interactuar con el Controlador, pero también se comunica con el Modelo mediante una referencia.

Controlador: Asigna significado a las acciones del usuario y actúa sobre los datos del Modelo. Es el mediador entre la Vista y el Modelo, tomando decisiones cuando hay cambios en la información. Interactúa directamente con el Modelo .

1.4.4.2. FLUTTER Y DART

Flutter, una innovadora tecnología de desarrollo híbrida respaldada por Google y basada en Dart, permite la rápida creación de aplicaciones nativas para Android e iOS. Su interfaz amigable, basada en Material Design, destaca por la capacidad de visualizar instantáneamente los cambios en el código en el emulador mientras la aplicación sigue en ejecución. Aunque Flutter no utiliza widgets nativos de Android e iOS, esta peculiaridad implica que todo el código relacionado con estos widgets se incorpora directamente en la aplicación, resultando en un tamaño de aplicación considerable. En resumen, Flutter facilita la integración de widgets personalizados y proporciona una variedad de widgets predefinidos para Android, iOS y Material Design.

Funcionalidades de Flutter

- **Calidad nativa**: Mientras que las aplicaciones nativas se desarrollan específicamente para un sistema operativo, Flutter aprovecha todas las ventajas de las aplicaciones nativas para garantizar una alta calidad en el resultado final.
- **Experiencia de usuario**: Flutter incorpora tanto el diseño Material de Google como el diseño Cupertino de Apple, lo que proporciona una experiencia de usuario óptima y hace que las interfaces de usuario sean idénticas a las de las aplicaciones desarrolladas por estas compañías.

- **Tiempo de carga**: Un factor crucial que puede llevar al abandono de una aplicación es el tiempo que tarda en cargar. Con Flutter, se experimentan tiempos de carga inferiores a un segundo tanto en dispositivos iOS como en Android.
- **Desarrollo ágil y rápido**: Gracias a la característica de hot-reload, se puede programar y ver los cambios en tiempo real en su dispositivo o en los simuladores. Esto facilita un desarrollo ágil y rápido, lo que permite una iteración más fluida y eficiente en el proceso de desarrollo de aplicaciones.

Dart es un lenguaje de programación de código abierto desarrollado por Google, presentado en 2011. Diseñado para la programación orientada a objetos y el análisis estático de tipos, se planteó como una alternativa a JavaScript con el potencial de ser el lenguaje principal en navegadores actuales. Aunque aún está en evolución, continúa mejorando y adaptándose.

Entre las características más importantes de Dart, destacan:

- Dart ofrece programación estructurada y flexibilidad, desarrollado por ingenieros de Google para adaptarse a una amplia variedad de proyectos, desde individuales hasta complejos y avanzados.
- Es un lenguaje familiar y fácil de aprender, con tutoriales disponibles en su sitio web que facilitan el proceso de aprendizaje y fomentan la colaboración entre desarrolladores.
- Dart se adapta a cualquier navegador al ser ejecutado mediante máquina virtual o compilador a JavaScript, proporcionando versatilidad en su implementación.
- Dart se basa en clases e interfaces, orientándose hacia la Programación Orientada a Objetos (POO), lo que simplifica la encapsulación y reutilización del código. Esto resulta en un desarrollo más eficiente y estructurado [9].

La estructura básica de un código en Dart se compone de varias partes esenciales.

- **Función principal**: Es el punto de entrada de la aplicación, donde se inicia la ejecución del código.

Código 1.1 Estructura básica de la función main

```
void main() {
  // Código de la función principal
}
```

- **Funciones y clases**: A continuación de la función principal, se puede definir otras funciones y clases que se utilizarán en el programa.

Código 1.2 Estructura básica de una función y una clase

```
void miFuncion() {
  // Código de la función
}

class MiClase {
  // Código de la clase
}
```

- **Widgets**: Son el elemento central del mecanismo de diseño en Flutter. Los elementos visibles, como imágenes, íconos y texto en una aplicación Flutter, son ejemplos de widgets. Sin embargo, también existen widgets que no son visibles para el usuario, como las filas, columnas y cuadrículas que organizan, limitan y alinean los widgets visibles en la interfaz de la aplicación [10].

Código 1.3 Estructura básica de un widget

```
class MiApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Mi Aplicación'),
        ),
        body: Center(
          child: Text('¡Hola, Flutter!'),
        ),
      ),
    );
  }
}
```

1.4.4.3. VISUAL STUDIO CODE

Visual Studio es una herramienta de desarrollo altamente poderosa que proporciona todas las capacidades necesarias para completar todo el ciclo de desarrollo en un solo lugar. Se trata de un entorno de desarrollo integrado (IDE) completo que permite a los desarrolladores escribir, editar, depurar y compilar código, además de facilitar la implementación de sus aplicaciones. Además de sus funciones de edición y depuración de código, Visual Studio incorpora compiladores, herramientas de autocompletado de código, control de código fuente, extensiones y una amplia variedad de características adicionales para mejorar cada etapa del proceso de desarrollo de software [11].

1.4.4.4. POSTMAN [12]

Postman es una herramienta esencial para desarrolladores y equipos de ingeniería de software, diseñada para simplificar el desarrollo, prueba y documentación de APIs. Permite enviar solicitudes HTTP, realizar pruebas automatizadas, organizar solicitudes en colecciones y generar documentación detallada de APIs. Postman facilita la

colaboración entre equipos al permitir compartir colecciones y documentación, convirtiéndose en una herramienta integral en el desarrollo de software moderno.

1.4.4.5. JSON [13]

JSON, abreviatura de JavaScript Object Notation es un estándar para la transmisión de datos entre un servidor y un navegador. Se utiliza tanto en el intercambio entre aplicaciones como dentro de aplicaciones del lado del cliente, destacando por su simplicidad de lectura y facilidad de manejo en diversos contextos. Durante la transferencia de datos entre un servidor y un navegador, estos se limitan a texto plano. JSON simplifica este proceso convirtiendo datos a su formato compuesto por texto plano, facilitando así su envío y recepción. Una característica fascinante de JSON es su versatilidad, ya que puede ser utilizado desde múltiples lenguajes, incluyendo aquellos orientados a objetos, tecnologías del lado del servidor, e incluso lenguajes dedicados a la manipulación de bases de datos.

A continuación, se puede ver un pequeño ejemplo sobre la representación del formato JSON:

Código 1.4 Ejemplo de la representación de un objeto en formato JSON

```
{“nombre”: “Juan”,
“apellido”: ”López”,
“edad”: 23,
“ciudad”: ”Quito”}
```

En este ejemplo, la variable JSON representa la información de una persona llamada Juan, en donde se incluye su nombre, edad, ciudad y edad. Su lectura es muy sencilla, ya que, cada propiedad está asociada a un valor, y los pares clave-valor se encuentran separados por comas.

1.4.4.6. MONGO DB

MongoDB constituye un sistema de gestión de bases de datos no relacionales, donde almacena objetos denominados documentos en lugar de filas. La estructura de almacenamiento de estos objetos sigue un formato similar al de JSON [14].

En la **Tabla 1.2** se muestra la representación JSON de un objeto de una clase llamada *Cliente*:

Tabla 1.2 Representación de una clase en formato JSON

Cliente	"id": 1, "nombre" "Luis", "apellido : "Sanchez", "edad" : 23
- id: int - nombre: String - apellido: String - edad: int	

1.4.4.6.1. ESTRUCTURA DE DATOS EN MONGODB [15]

En MongoDB, los documentos equivalen a los registros presentes en bases de datos relacionales. La distinción radica en que cada documento comprende tanto la descripción de la estructura como las instancias mediante un conjunto de pares de llaves y valores, tal como se ilustra en la **Código 1.5.**

Código 1.5 Ejemplo de documento en MongoDB **[15]**

```
1 {
2   "_id" :
    ↪ ObjectId("58a78d740b0fa18b936f1d9a"),
3   "movieId" : 1,
4   "title" : "Toy Story (1995)",
5   "genres" : "Adventure|Animation|
    ↪ Children|Comedy|Fantasy"
6 }
```

Cada documento posee un identificador denominado "_id", el cual puede ser proporcionado por el programador y puede ser de cualquier tipo. En ausencia de suministro, MongoDB asigna automáticamente un valor de tipo "ObjectId", el cual se muestra en la línea 2 del Código 1.5.

Los documentos se almacenan en una colección que se asemeja a una tabla en bases de datos relacionales. En esta colección, no es necesario que los documentos compartan el mismo conjunto de llaves, lo que significa que pueden tener estructuras diversas. Esta característica brinda a los diseñadores de bases de datos en MongoDB la flexibilidad para crear de manera dinámica estructuras más apropiadas para la aplicación.

1.4.4.7. NODE JS [16]

Node.js es un entorno de ejecución multiplataforma diseñado para el lenguaje de programación JavaScript. Este entorno es de código abierto y está distribuido bajo la licencia MIT, lo que implica que cualquier individuo tiene la libertad de descargarlo e instalarlo en su computadora sin incurrir en costos de licencia.

La sintaxis de las primitivas utilizadas para programar en Node.js se basa en la especificación ECMAScript, la cual es publicada por la organización ECMA International.

Node.js presenta una arquitectura orientada a eventos, y su motor para interpretar y ejecutar código JavaScript es el V8 de Google. Inicialmente concebido para entornos de servidores con elevada concurrencia, Node.js emplea un modelo de evaluación de un solo hilo de ejecución, respaldado por un esquema asíncrono de entrada/salida llamado EventLoop. Este enfoque permite manejar miles de llamadas sin recurrir a cambios de contexto, un proceso que resulta oneroso a nivel del sistema operativo.

1.4.4.8. EXPRESS JS [17]

Express.js, un destacado framework de Node.js, simplifica la construcción de APIs al operar en la capa superior del servidor web Node.js. Su enfoque se centra en la organización mediante middlewares y enrutamiento, ofreciendo una extensibilidad estándar y facilitando la representación de vistas HTML dinámicas. Express.js es ideal para aplicaciones de una sola página, sitios web híbridos o APIs públicas.

Dos aspectos clave de Express.js son su capacidad para simplificar tareas complejas durante el desarrollo, como enviar imágenes con una sola línea de código, mejorando el rendimiento del servidor. Además, permite trabajar con partes y componentes específicos, promoviendo la facilidad de mantenimiento y la modularidad del sistema.

1.4.4.9. REST

REST (Representational State Transfer) es un estilo arquitectónico ampliamente adoptado para el diseño de servicios web que se comunican a través de la web. Propuesto por Roy Fielding en 2000, REST establece principios arquitectónicos sin ser una especificación estándar en sí misma. En su implementación, se basa en estándares como HTTP y XML. Los servicios REST se caracterizan por cuatro operaciones principales: GET para consultar, POST para crear, PUT para editar y DELETE para eliminar recursos. Además, hacen uso de hipermedias que permiten a los clientes navegar entre recursos a través de enlaces HTML. Las respuestas del API REST comúnmente se presentan en formato JSON, independientemente del lenguaje utilizado [18].

1.4.4.10. ANDROID STUDIO

Android Studio es el entorno de desarrollo integrado oficial de Google para crear aplicaciones móviles en el sistema operativo Android. Es una herramienta poderosa y completa que proporciona a los desarrolladores todas las funciones necesarias para diseñar, desarrollar, depurar y probar aplicaciones Android de manera eficiente.

Android Studio es un IDE desarrollado por Google, basado en el popular IDE IntelliJ IDEA de JetBrains y optimizado específicamente para el desarrollo de aplicaciones

Android. Ofrece una amplia variedad de herramientas y características que simplifican el proceso de desarrollo, como un editor de código avanzado con resaltado de sintaxis, autocompletado inteligente y refactorización de código. Además, incluye un emulador de dispositivos Android para probar las aplicaciones en diversas configuraciones y se integra con el sistema de compilación Gradle para facilitar la gestión del proyecto. Es compatible con varios lenguajes de programación, como Java y Kotlin. [19].

2. METODOLOGÍA

En este capítulo se detallará el proceso de elaboración del aplicativo móvil para la gestión de la bolsa de empleos. Se abordará desde la adquisición de requerimientos hasta el diseño e implementación de la aplicación móvil lo cual estará gestionado y estructurado según la metodología Kanban.

2.1. DISEÑO

2.1.1. DEFINICIÓN DEL TABLERO KANBAN

Se ha empleado la herramienta Trello de Atlassian [20] para elaborar el tablero Kanban. Este tablero se utilizará durante las etapas de diseño, implementación y pruebas, por lo tanto, es esencial distinguir las tareas correspondientes a cada fase mediante etiquetas ubicadas en la parte superior de cada tarea, según se describe a continuación:

- Fase de diseño:
- Fase de implementación:
- Fase de pruebas: 

En la **Figura 2.1**. se puede observar el estado del tablero durante la fase de Diseño:

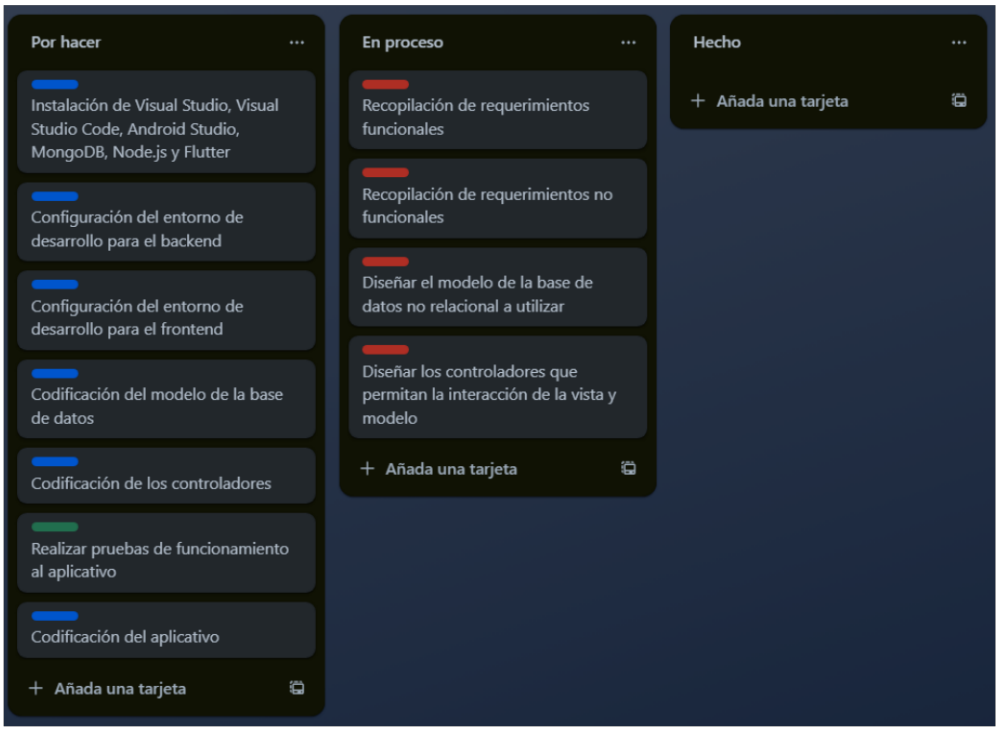

Figura 2.1 Tablero Kanban durante la fase de Diseño

2.1.2. OBTENCIÓN DE REQUERIMIENTOS

La recopilación de requisitos es esencial en el desarrollo, involucrando la identificación y documentación de las necesidades y expectativas de los usuarios, así como los objetivos del negocio. Requiere una comunicación efectiva con los involucrados para comprender claramente los requisitos funcionales y no funcionales. Esta fase sienta las bases para el diseño y desarrollo de la aplicación, asegurando la satisfacción de los usuarios y el alineamiento con los objetivos comerciales. Un análisis exhaustivo en esta etapa previene malentendidos y cambios significativos en fases posteriores, mejorando la eficiencia del desarrollo de la aplicación.

En el proceso de obtención de requisitos de este proyecto, se llevaron a cabo entrevistas con los usuarios finales, obteniendo así un conjunto de requisitos tanto funcionales como no funcionales.

2.1.2.1. REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales son detalles específicos que describen las funciones que una aplicación debe cumplir para satisfacer las necesidades del usuario y los objetivos del negocio. Estos especifican cómo debe comportarse el sistema en términos de operaciones, procesos e interacción con los usuarios. Son esenciales para guiar a los desarrolladores y definir las capacidades operativas clave de la aplicación.

A continuación, se muestran las historias de usuario derivadas de los requisitos funcionales recopilados, donde se pueden identificar los diversos roles de usuario que la aplicación gestionará:

Tabla 2.1 RF01: Registro de Usuarios

Código	RF01
Nombre:	Registro de usuario
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
El sistema permite el registro de nuevos usuarios, los cuales podrán ingresar ya sea sus datos personales o los datos de su empresa según corresponda.	

Tabla 2.2 RF02: Inicio de sesión

Código	RF02
Nombre:	Inicio de sesión
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Un usuario, previamente registrado, podrá iniciar sesión en la aplicación ingresando usuario y contraseña.	

Tabla 2.3 RF03: Perfil de usuario

Código	RF03
Nombre:	Perfil de Usuario
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Los usuarios podrán ver y editar la información mostrada en sus perfiles en función de su rol. Tanto los Administradores como los Clientes podrán acceder a su información personal, académica y datos relacionados con su experiencia laboral. Las empresas podrán visualizar la información relacionada con su entidad. El sistema simplificará la navegación hacia estas funciones para todos los usuarios.	

Tabla 2.4 RF04: Gestión de Empleos

Código	RF04
Nombre:	Gestión de Empleos
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Los usuarios tendrán acceso a distintas funcionalidades de acuerdo con su rol. Los Clientes podrán visualizar la lista de ofertas de empleos a los cuales podrán postular. Las Empresas podrán crear, ver, editar y eliminar sus ofertas de trabajo, así como	

aceptar o negar las postulaciones que estos hayan recibido, en caso de negación, se podrá seleccionar el motivo de rechazo, adicionalmente, se podrá acceder a una vista del perfil del usuario que realizó la postulación. Los Administradores podrán visualizar y eliminar todas las ofertas de trabajo que hayan sido publicadas por Empresas registradas, así como visualizar y eliminar las postulaciones que estas hayan recibido.

Tabla 2.5 RF05: Postulaciones de Usuario

Código	RF05
Nombre:	Postulaciones de Usuario
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Los usuarios podrán visualizar y eliminar la lista de postulaciones realizadas.	

Tabla 2.6 RF06: Exploración de Usuarios

Código	RF06
Nombre:	Exploración de Usuarios
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Los usuarios podrán visualizar los perfiles de los demás usuarios (Clientes y Empresas) que estén registrados en la aplicación.	

Tabla 2.7 RF07: Gestión de Usuarios

Código	RF07
Nombre:	Gestión de Usuarios
Usuarios:	Administrador
Prioridad	Alta
Descripción:	
Los administradores tendrán la capacidad de ver y modificar los perfiles de los usuarios (Clientes y Empresas) registrados en la aplicación. Además, podrán cambiar el estado de los usuarios de activo a inactivo y viceversa según sea necesario.	

Tabla 2.8 RF08: Reportes

Código	RF08
Nombre:	Reportes
Usuarios:	Administrador
Prioridad	Alta
Descripción:	
Los administradores tendrán acceso a una interfaz que les proporcionará un resumen de la cantidad total de usuarios registrados, separado por clientes y empresas. Además, tendrán a su disposición gráficos estadísticos que detallan el número de	

postulaciones recibidas y su estado correspondiente. Por último, podrán visualizar un gráfico que presenta los motivos de rechazo de estas solicitudes.

2.1.2.2. REQUERIMIENTOS NO FUNCIONALES

Los requerimientos no funcionales son criterios que definen la calidad y características operativas de un sistema, pero no se refieren directamente a sus funciones específicas. Estos abordan aspectos como la seguridad, el rendimiento, la usabilidad y otros atributos que influyen en la eficiencia y experiencia del usuario. A diferencia de los requerimientos funcionales, que se centran en lo que el sistema debe hacer, los no funcionales se centran en cómo debe hacerlo, garantizando así que el sistema cumpla con estándares y expectativas de rendimiento.

A continuación, se presentan los requerimientos no funcionales:

Tabla 2.9 RNF01: Interfaz intuitiva

Código	RNF01
Nombre:	Interfaz intuitiva
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
Las interfaces del aplicativo deben ser intuitivas para prevenir confusiones y permitir una navegación fácil entre los distintos módulos.	

Tabla 2.10 RNF02: Aplicación modular

Código	RNF02
Nombre:	Aplicación modular
Usuarios:	Administrador, Cliente y Empresa
Prioridad	Alta
Descripción:	
El aplicativo debe ser implementado de manera que posibilite la integración de nuevos módulos con el fin de mejorar su desempeño.	

2.1.3. DIAGRAMAS DE CASOS DE USO

Un diagrama de casos de uso es una representación gráfica que ilustra la interacción de los usuarios con un sistema para alcanzar objetivos específicos. En el análisis y diseño de sistemas, se emplea para visualizar las funciones clave desde la perspectiva del usuario. Este diagrama identifica actores (usuarios o sistemas externos) y casos de uso (acciones del sistema), facilitando la comunicación, la identificación de requisitos y sirviendo como base para el diseño de sistemas efectivos [21].

En las Figuras que se muestran a continuación, se puede observar los actores, los módulos que se encontrarán a su disponibilidad y las funcionalidades del aplicativo.

Figura 2.2 Diagrama de casos de uso para el rol Cliente

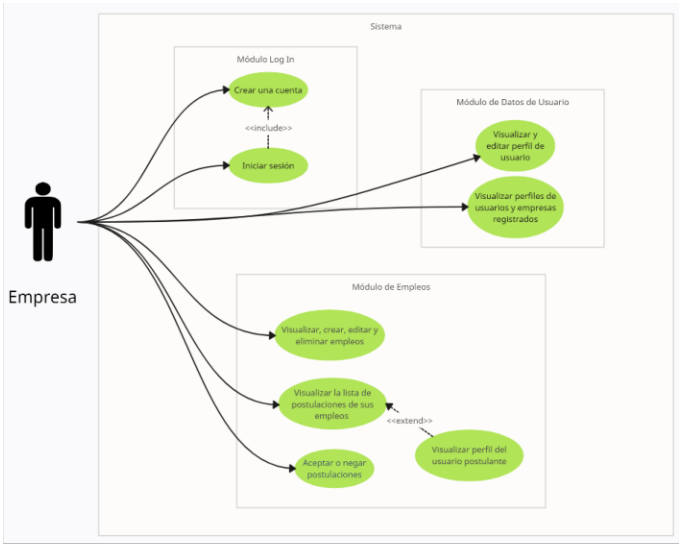

Figura 2.3 Diagrama de casos de uso para el rol Empresa

Figura 2.4 Diagrama de casos de uso para el rol Administrador

2.1.4. DISEÑO DEL MODELO

En el modelo, reside la lógica central de la aplicación y las utilidades para interactuar con la base de datos. En esta sección, se detallará la configuración de la base de datos no relacional utilizada en la aplicación. Al optar por MongoDB, una base de datos no relacional, se pueden establecer relaciones entre documentos de dos maneras: mediante una relación incrustada, que implica almacenar documentos secundarios dentro de un documento principal; o mediante una relación referenciada, empleada en la estructuración de la base del proyecto. También se puede utilizar DBRefs como una convención para representar un documento en lugar de un tipo de referencia específico [22].

Para el diseño de la base de datos, es necesario reconocer las colecciones de datos, así como las relaciones existentes, para ello, empleando la herramienta Moon Modeler proporcionada por DATENSEN [23], se elabora el esquema de relaciones de la base, mismo que puede ser observado en la **Figura 2.5**:

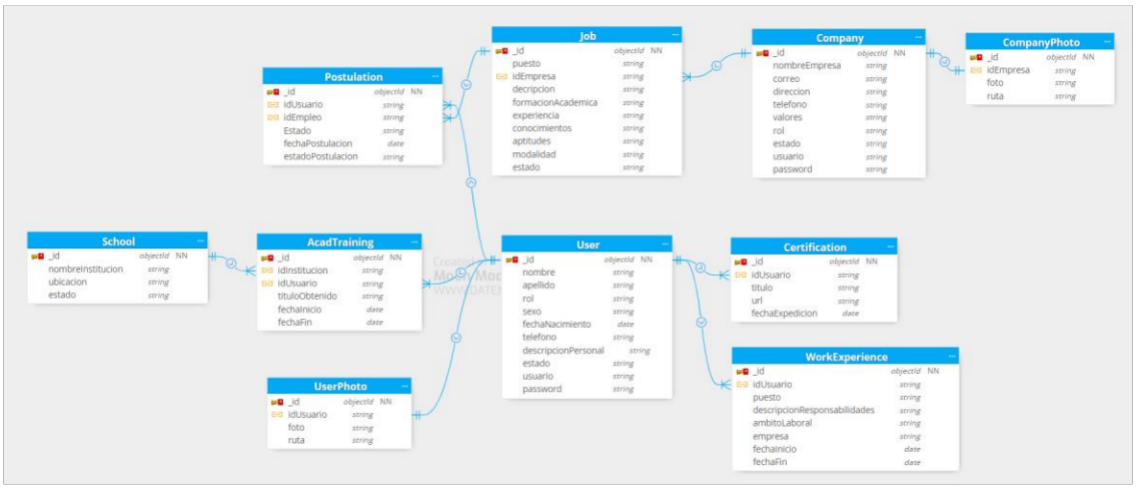

Figura 2.5 Esquema de relaciones de la base de datos

En la **Figura 2.6**, se examina minuciosamente una de las colecciones del esquema de relaciones, mostrando los atributos propios de la colección, así como los que posibilitan establecer la conexión entre colecciones a través de referencias.

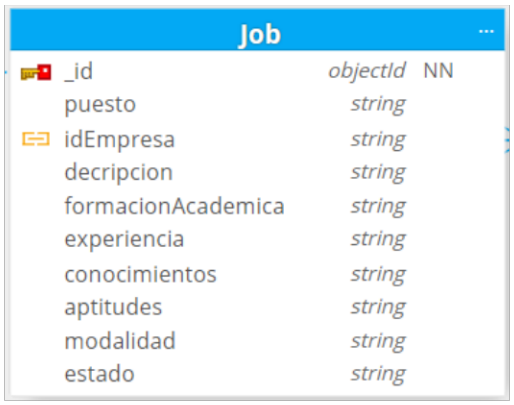

Figura 2.6 Colección "Job" de la base de datos

2.1.5. DISEÑO DEL CONTROLADOR

Se utilizarán diagramas de secuencia para describir cómo funcionan los controladores, ya que son útiles para visualizar la interacción entre la Vista y el Modelo, mediada por el controlador. Estos diagramas mostrarán las acciones que la Vista lleva a cabo sobre los datos almacenados y cómo el Controlador las convierte en solicitudes hacia el Modelo. En las **Figuras 2.7, 2.8 y 2.9**, se presentan los diagramas correspondientes a las operaciones CRUD del sistema. Es relevante destacar que el funcionamiento de los controladores es uniforme para todas las entidades, por lo tanto, se utiliza el término "Entity" para referirse a estas de manera genérica.

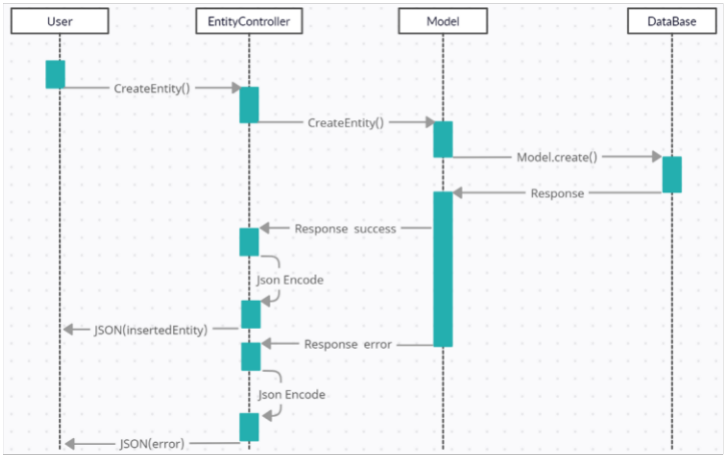

Figura 2.7 Diagrama de secuencia del controlador que permite la creación de una nueva entidad

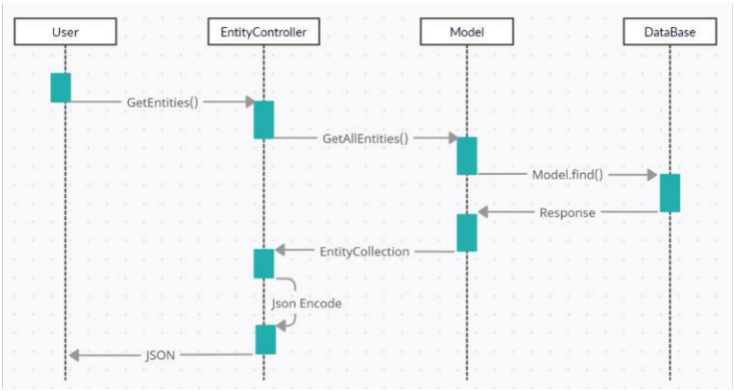

Figura 2.8 Diagrama de secuencia del controlador que permite la obtención de la lista completa de una entidad

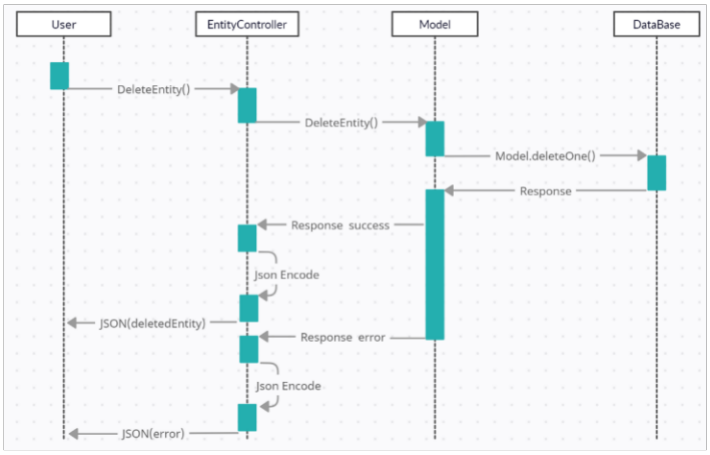

Figura 2.9 Diagrama de secuencia del controlador que permite eliminar una entidad

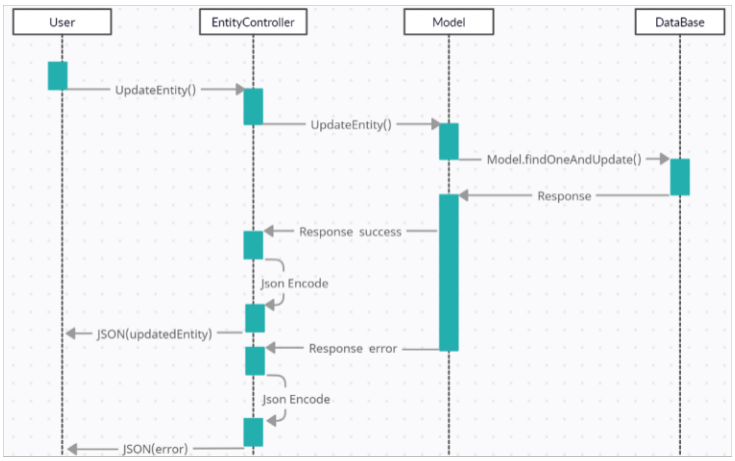

Figura 2.10 Diagrama de secuencia del controlador que permite la actualización de una entidad

2.1.6. DISEÑO DE LA VISTA

La vista es la representación de la interfaz de usuario, siendo esta la forma en que los datos se muestran a los miembros de la Junta Parroquial. Su función primordial es presentar la información de manera comprensible y transmitir las acciones del usuario al controlador para su procesamiento. No tiene la capacidad de manipular los datos directamente ni de llevar a cabo acciones; su tarea es mostrar la información adecuadamente y comunicar las interacciones del usuario al controlador para su gestión correspondiente.

La interfaz de usuario se creará con Flutter y la librería Cupertino para los widgets visuales. Estos se basan en elementos que se pueden manipular para crear interfaces interactivas. Por lo tanto, la estructura de la Vista debe centrarse en la reutilización de widgets para garantizar eficiencia y escalabilidad. Además, el diseño debe abordar el enrutamiento de la aplicación y el acceso a los datos a través de una API.

2.1.6.1. ENRUTAMIENTO

En Flutter, Navigator es una herramienta crucial para la navegación entre diferentes vistas dentro de una aplicación. Funciona como una pila de rutas, donde cada pantalla visitada se añade a la pila y puede ser eliminada o reemplazada según las interacciones del usuario. Por ello, es necesario distinguir los tipos de rutas que se manejaran en la aplicación:

- **Rutas nombradas**: Las Rutas Nombradas son caminos que conducen a pantallas o interfaces que no necesitan un parámetro de entrada específico para operar. La aplicación incorporará varias interfaces de este tipo, que incluyen funciones como Inicio de Sesión, Registro de Nuevo Cliente y Registro de Nueva Empresa.

- **Rutas dadas**: Las Rutas Dadas, a diferencia de las mencionadas anteriormente, son interfaces que necesitarán que la pantalla anterior envíe uno o más parámetros de entrada para llevar a cabo sus funciones basadas en dichos datos. Algunas de estas interfaces están relacionadas con procesos de edición de información de usuarios, empleos y postulaciones.

2.1.6.2. ACCESO A LOS DATOS

La Vista obtendrá acceso a la información a través de una API suministrada por el backend de la aplicación, la cual facilitará la gestión de los datos guardados de los usuarios de la Junta Parroquial. La interacción y manipulación de datos se llevarán a cabo mediante el empleo de la **librería HTTP de Flutter**. Esta herramienta comprende un conjunto de funciones y clases de nivel superior que simplifican la realización de solicitudes HTTP. Vale la pena mencionar que la librería HTTP de Flutter admite diversos tipos de peticiones, lo que permite realizar operaciones como obtener datos (GET), enviar datos para su procesamiento (POST), actualizar información (PUT), y eliminar datos (DELETE), entre otras funcionalidades.

2.1.6.3. BOCETOS DE LAS INTERFACES

Antes de comenzar la fase de Implementación, se empleó la herramienta Figma para esbozar brevemente las interfaces. En la Figura 2.11 se muestra el bosquejo de la interfaz de inicio de sesión de la aplicación; los demás se encuentran detallados en el **Anexo I**.

Figura 2.11 Boceto de la interfaz de inicio de sesión

2.2. IMPLEMENTACIÓN

2.2.1. ACTUALIZACIÓN DEL TABLERO KANBAN

Una vez completada la etapa de Diseño, se realiza la actualización del tablero Kanban, trasladando las tareas de la fase de Diseño a la lista de tareas finalizadas y las tareas

de la fase de Implementación a la lista de tareas en progreso. En la **Figura 2.12** se puede visualizar el tablero actualizado.

Figura 2.12 Tablero Kanban para la fase de Implementación

2.2.2. INSTALACIÓN Y CONFIGURACIÓN DE HERRAMIENTAS

En la **Tabla 2.11**, se encuentra detallado el proceso de instalación y configuración de las herramientas necesarias para la implementación de la aplicación.

Tabla 2.11 Instalación y configuración de herramientas

Herramienta	Instalación
Visual Studio	• Descargar Visual Studio Community desde la **página oficial**. • Finalizada la descarga, ejecutar el instalador. Dentro del instalador, buscamos la sección de **Móviles y de escritorio**, marcar las siguientes opciones de Desarrollo móvil y para el escritorio con C++ y comenzar el proceso de instalación.
Visual Studio Code	• Descargar Visual Studio Code desde su **página oficial** y seguir el proceso de instalación por defecto. • Instalar los pluggins Dart, Flutter y Flutter Widget Snipets lo cuales son necesarios para la generación de proyectos de Flutter.
Android Studio	• Descargar el instalador de Android Studio desde su **página oficial.** • Una vez finalizada la descarga, ejecutar el instalador y seguir la instalación por defecto. Al concluir con la instalación, acceder a Android Studio e ingresar al

	SDK mánager ubicado en la parte superior derecha. • Dentro de la pestaña de **SDK Platfmorms,** seleccionar la versión de Android del dispositivo con el que se trabajará y dar clic en Apply para comenzar con el proceso de descarga e instalación del paquete.
Postman	• Descargar el instalador de la **página oficial** y seguir la instalación por defecto
Node.js	• Descargar el instalador de Node.js de su **página oficial** y seguir la instalación dejando los parámetros por defecto. • Para verificar que la instalación se completó de forma correcta usar el comando **node -v** dentro en una terminal.
MongoDB	• Para la instalación de MongoDB, se siguió el proceso descrito en la **página oficial de MongoDB**.
Flutter	• Descargar el SDK de Flutter para Windows desde la **página oficial**. • Descomprimir el archivo descargado y moverlo a la raíz del disco C. • Agregar una variable de entorno dentro de la variable **Path** que contenga la dirección a la carpeta **bin** ubicada dentro del directorio del paso anterior (**C:\flutter\bin**) • Comprobar que el sistema reconozca la ruta empleando el comando Flutter doctor dentro de la terminal.

2.2.3. IMPLEMENTACIÓN DEL MODELO

La ejecución del modelo se realizará utilizando la biblioteca Mongoose para MongoDB. Esto facilitará el modelado de datos, la aplicación de esquemas, la validación de modelos y la manipulación general de datos. Para comenzar, dentro de Visual Studio Code, ejecutamos los comandos descritos a continuación en la **Tabla 2.12**:

Tabla 2.12 Comandos de configuración del entorno de desarrollo para el modelo

```
mkdir Backend
cd Backend
npm init -y
npm install express mongodb mongoose nodemon
```

En una secuencia ordenada, los comandos cumplen las siguientes funciones: crear el directorio destinado al proyecto, ingresar al directorio recién creado, iniciar el proyecto y generar el archivo llamado package.json que almacena toda la información del proyecto; por último, instalación de las bibliotecas express, mongodb, mongoose y nodemon.

2.2.3.1. ESTRUCTURA DE CARPETAS

Para abordar el proyecto, es importante comprender bien el scaffolding (o estructura de carpetas). Aunque esta disposición puede variar de un proyecto a otro, se utilizará la estructura descrita en la **Tabla 2.13**.

Tabla 2.13 Estructura de archivos del proyecto para el Backend

```
-Backend/                          #Directorio raíz del proyecto
   -  config/                      #Configuración de la base de datos
   -  controllers/                 #Controladores solicitudes http
   -  models/                      #Entidades de la base de datos
   -  routes/                      #Definición de rutas
   -  node_modules/                #Paquetes de Node
   -  server.js                    #Levantamiento servidor
```

2.2.3.2. CONEXIÓN A MONGODB

El **Código 2.1**, corresponde al archivo **mongoose.config.js** ubicado dentro de la carpeta **config**. Este código se encarga de conectar la aplicación Node.js a la base de datos MongoDB llamada 'tic_db', utilizando la biblioteca Mongoose, e imprime mensajes de éxito o error en la consola según el resultado de la conexión.

Código 2.1 Configuración de la conexión con la base de datos

```
const mongoose = require('mongoose');
const db = 'tic_db';

mongoose.connect(`mongodb://127.0.0.1/${db}`, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
})
    .then(() => console.log(`Established a connection to database ${db}`))
    .catch((error => console.log(`Something went wrong when connecting to
the ${db}`, error)))
```

En el **Código 2.2**, se observa el contenido del archivo **server.js** el cual se encarga de establecer la conexión con la base de datos y de iniciar el servidor encargado de manejar las solicitudes HTTP mediante una instancia de Express.

Código 2.2 Levantamiento del servicio web

```
const app = express();
const port = 8000;
require('./config/mongoose.config.js');
app.listen(port, () => console.log("Server is listening at port ", port));
```

2.2.3.3. APLICACIÓN DE NODEMON

Nodemon es una herramienta esencial para el desarrollo en Node.js, ya que reinicia automáticamente la aplicación al detectar cambios en los archivos fuente. Esto acelera

el proceso de desarrollo al eliminar la necesidad de reiniciar manualmente el servidor tras realizar modificaciones en el código. Para su implementación en el proyecto, se ingresa la línea 7, dentro del archivo package.json. Esto puede ser observado en el **Código 2.3**:

Código 2.3 Implementación de Nodemon

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "server": "npx nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "mongodb": "^6.1.0",
    "mongoose": "^7.6.3",
    "nodemon": "^3.0.1"
  }
}
```

2.2.3.4. DEFINICIÓN DE MODELOS

En Mongoose, la definición de modelos se logra mediante Schemas, estructuras que definen la forma de los documentos almacenados en MongoDB. Los Schemas determinan campos, tipos de datos y opciones de validación. Al crear un modelo desde un Schema, se establece una estructura coherente para los datos, simplificando la interacción con la base de datos y proporcionando una capa de abstracción que facilita las operaciones CRUD [24]. Para ilustrar su implementación, se toma como ejemplo el modelo “User”, el resto de los modelos pueden ser observados en el **Anexo II**. En el **Código 2.4**, se puede observar el modelo de la entidad Usuario.

Código 2.4 Modelo de la entidad “User”

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
    nombre:{
        type: String,
        required: [true, 'El nombre es obligatorio'],
    },
    apellido:{
        type: String,
        required: [true, 'El apellido es obligatorio'],
    },
    rol: {
```

```
        type: String,
        required: [true, 'El rol es obligatorio']
    },
    sexo: {
        type: String,
        required: [true, 'El sexo es obligatorio']
    },
    fechaNacimiento:{
        type: Date,
        required: [true, 'La fecha de nacimiento es obligatoria'],
    },
    telefono:{
        type: String,
        required: [true, 'El teléfono es obligatorio'],
    },
    descripcionPersonal:{
        type: String,
        required: [true, 'La descripción personal es obligatoria'],
    },
    estado:{
        type: String,
        required: [true, 'El estado es obligatorio'],
    },
    usuario:{
        type: String,
        required: [true, 'El usuario es obligatorio'],
    },
    password:{
        type: String,
        required: [true, 'La contraseña es obligatorio'],
    },
});

const User = mongoose. Model('User', UserSchema);
module.exports = User;
```

En el **Código 2.4**, se puede observar que, para cada atributo que compone al modelo, se define su tipo y si es obligatorio, junto con un mensaje que se proporcionará en caso de no recibir dicho atributo al enviar una solicitud HTTP.

Mongoose también permite la inclusión de atributos virtuales, los cuales no se almacenan directamente en la base de datos, pero permiten realizar acciones antes de su almacenamiento. En este caso, se empleará un atributo virtual llamado "confirmPassword" para verificar la igualdad entre las contraseñas proporcionadas por el usuario al realizar una solicitud POST durante el registro. Esto puede observarse en el **Código 2.5**.

Código 2.5 Esquema virtual de la entidad "User"

```
UserSchema.virtual('confirmPassword')
.get( () => this.confirmPassword )
.set( value => this.confirmPassword = value );
```

```
UserSchema.pre('new', function(next) {
    console.log(this.password ,this.confirmPassword)
    if (this.password != this.confirmPassword) {
    this.invalidate(¡'confirmPassword', 'Las contraseñas deben
coincidir!');
    }
    next();
    });
```

Finalmente, en el **Código 2.6**, se implementa un middleware en Mongoose para MongoDB en Node.js. Este middleware, utilizando la función pre, intercepta el evento "save" antes de almacenar un documento en la base de datos. Durante este proceso, se aplica la función md5 al campo de contraseña, convirtiéndolo en su hash MD5 correspondiente. Esta práctica mejora la seguridad al almacenar las contraseñas como hashes en lugar de texto plano.

Código 2.6 Implementación del middleware para la encriptación de la contraseña

```
UserSchema.pre('save', function(next) {
    this.password = md5(this.password);
    next();
});
```

2.2.4. IMPLEMENTACIÓN DEL CONTROLADOR

De igual manera que para el modelo, partimos ejecutando, dentro del directorio del proyecto, los comando descritos en la **Tabla 2.14**:

Tabla 2.14 Comandos de configuración del entorno de desarrollo para el controlador

```
cd Backend
npm install cors multer md5
```

El segundo comando se utiliza para instalar dos bibliotecas: "cors", que posibilita la implementación de un middleware para asegurar peticiones HTTP [25], y "multer", una librería que también permite desarrollar middleware en Node.js y resulta especialmente útil para el procesamiento de datos de formularios que contienen archivos binarios, como imágenes [26].

2.2.4.1. CODIFICACIÓN DE LOS CONTROLADORES

En la base de datos, se manejan diversas entidades mediante funciones CRUD estándar para asegurar operaciones básicas. Sin embargo, entidades como Usuario y Empresa requieren un tratamiento más especializado. Estas entidades van más allá de las operaciones CRUD convencionales al incorporar la gestión de fotografías de perfil. Para

abordar esto, se han implementado controladores adicionales que permiten una gestión optimizada y segura de las imágenes asociadas a Usuarios y Empresas en la aplicación.

Debido a esta distinción en los requisitos y operaciones, la gestión de entidades en nuestra aplicación se divide en dos secciones principales.

2.2.4.1.1. CONTROLADORES DE PETICIONES CRUD

Con el propósito de ilustrar la implementación práctica de estos controladores, se tomará la entidad “User” como ejemplo.

En el **Código 2.7**, se muestra la función "**createUser**", encargada de generar nuevos usuarios mediante peticiones **POST**. Al recibir los datos del nuevo usuario a través de una solicitud POST, extrae las propiedades esenciales, como nombre, apellido y contraseña, del cuerpo de la solicitud y crea un nuevo registro dentro de la base de datos.

Código 2.7 Controlador para peticiones HTTP POST

```
module.exports.createUser = (request, response) =>{

    const {nombre, apellido, rol, sexo, fechaNacimiento, telefono,
descripcionPersonal, estado, usuario, password, confirmPassword} =
request.body;
    console.log(request.body)
    User.create({
        nombre, apellido, rol, sexo, fechaNacimiento,
telefono,descripcionPersonal, estado, usuario, password, confirmPassword
    })
        .then(User => response.json({insertedUser: User, msg: 'Succesful
creation'}))
        .catch(err => response.status(400).json(err));
}
```

En el **Código 2.8**, se presenta el controlador para peticiones **GET** implementado mediante la función llamada "**getAllUsers**". La cual permite obtener la información de todos los usuarios en la base.

Código 2.8 Controlador para peticiones HTTP GET

```
module.exports.getAllUsers = (_,response) =>{
    User.find({})
    .then(retrievedUsers => response.json(retrievedUsers))
    .catch(err => response.json(err))
}
```

La funcion “**getUser**”, observada en el **Código 2.9,** permite obtener información de un usuario específico en la base de datos utilizando el ID del usuario como parámetro como parámetro de búsqueda.

Código 2.9 Controlador para peticiones HTTP GET de un usuario especifico

```
module.exports.getUser = (request, response) =>{
    User.findOne({_id: request.params.id})
    .then(User => response.json(User))
    .catch(err => response.json(err))
}
```

En el **Código 2.10**, se presenta la implementación del controlador para peticiones **PUT** mediante la función "**updateUser**", la cual posibilita la actualización de la información de un usuario en la base de datos. Al recibir una solicitud HTTP PUT que incluye el ID del usuario y los nuevos datos, realiza la búsqueda y actualización del usuario correspondiente utilizando el modelo "User". Luego, responde con la información actualizada del usuario o un mensaje de error.

Código 2.10 Controlador para peticiones HTTP PUT

```
module.exports.updateUser = (request, response) =>{
    User.findOneAndUpdate({_id: request.params.id}, request.body, {new:
true})
    .then(updateUser => response.json(updateUser))
    .catch(err => response.json(err))
}
```

Finalmente, la función "**deleteUser**", que se encuentra en el **Código 2.11,** actúa como un controlador para peticiones **DELETE**. Al recibir una solicitud HTTP DELETE con el ID del usuario como parámetro, utiliza el modelo "User" para localizar y eliminar el usuario correspondiente. Luego, responde proporcionando detalles sobre la eliminación del usuario o, en caso necesario, emite un mensaje de error.

Código 2.11 Controlador para peticiones HTTP DELETE

```
module.exports.deleteUser = (request, response) =>{
    User.deleteOne({_id: request.params.id})
    .then(UserDeleted => response.json(UserDeleted))
    .catch(err => response.json(err))
}
```

2.2.4.1.2. CONTROLADORES DE PETICIONES DE FOTOS DE PERFIL

Para comenzar, incorporamos una carpeta adicional denominada "**Imágenes**" dentro de **la estructura de carpetas del proyecto**. Esta carpeta se destina a almacenar los archivos de imágenes que sean enviados por usuarios y empresas.

Es necesario establecer un nuevo modelo para el almacenamiento de las fotos de perfil tanto para Usuarios como para Empresas. En el **Código 2.12**, se puede observar el esquema del modelo de la entidad “UserPhoto”:

Código 2.12 Modelo para la entidad "UserPhoto"

```
const mongoose = require('mongoose');

const UserPhotoSchema = new mongoose.Schema({

    foto:{
        type: String,
        default: ''
    },
    ruta: {
        type: String,
        default: ''
    },
    idUsuario:{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    }
});

const UserPhoto = mongoose.model('UserPhoto', UserPhotoSchema);
module.exports = UserPhoto;
```

En el **Código 2.13**, se puede observar la implementación del controlador para las peticiones POST de imágenes de perfil de usuario.

Código 2.13 Controlador para peticiones POST de imágenes de perfil de usuarios

```
module.exports.addPhoto = (request, response) => {
    if (!request.file) {
        return response.status(400).json({ error: 'No se ha enviado ningún
archivo.' });
    }
    const filePath = 'Imagenes/' + request.file.filename;
    const nuevaFoto = {
        foto: request.file.filename,
        ruta: filePath,
        idUsuario : request.params.id
    };

    UserPhoto.findOne({idUsuario: request.params.id})
        .then(fotoUsuario => {
            if (!fotoUsuario) {
                return User.findById(request.params.id);
            }
            fotoUsuario.foto = nuevaFoto.foto;
            fotoUsuario.ruta = nuevaFoto.ruta;
            return fotoUsuario.save();
        })
        .then(userOrFoto => {
            if (!userOrFoto) {
```

```
                return Promise.reject({ status: 404, message: 'Usuario no
encontrado' });
            }
            if (userOrFoto instanceof User) {
                return UserPhoto.create(nuevaFoto);
            }
            return userOrFoto;
        })
        .then(() => {
            return response.json({ mensaje: 'Foto agregada exitosamente'
});
        })
        .catch(error => {
            console.error('Error al agregar la foto:', error);
            if (error.status) {
                return response.status(error.status).json({ error:
error.message });
            } else {
                return response.status(500).json({ error: 'Error interno
del servidor' });
            }
        });
};
```

Se comienza con la validación de la existencia de un archivo en la solicitud. Posteriormente, se procede a crear la ruta y los datos asociados a la foto, incluyendo el nombre, la ruta y el ID del usuario correspondiente. La base de datos es consultada y actualizada mediante el modelo "UserPhoto". La respuesta a esta operación se presenta como un mensaje JSON que informa sobre el éxito al agregar la foto. Cualquier error que surja durante este proceso se gestiona adecuadamente, proporcionando códigos y mensajes coherentes con el estado HTTP.

Finalmente, en el **Código 2.14**, se presenta el controlador para las peticiones GET, cuyo funcionamiento es similar a los previamente descritos. Vale la pena señalar que este controlador añade la funcionalidad de devolver la ruta hacia el archivo solicitado por el usuario o, en caso de no encontrar uno asociado al usuario, devuelve un objeto nulo.

Código 2.14 Controlador para peticiones GET de fotos de perfil de usuarios

```
module.exports.getUserPhoto = (request, response) => {
    UserPhoto.findOne({idUsuario: request.params.id})
        .then(user => {
            if (!user) {
                return response.status(404).json({ error: 'Usuario no
encontrado.' });
            }
            const imageUrl = user.foto ?
`http://localhost:8000/Imagenes/${user.foto}` : null;
            response.json({ foto: imageUrl });
        })
        .catch(error => {
            console.error('Error al obtener la foto del usuario:', error);
```

```
            response.status(500).json({ error: 'Error interno del servidor'
});
        });
};
```

2.2.4.2. ENRUTAMIENTO

Se ha implementado una estructura organizativa que asigna archivos específicos para cada entidad, y cada uno de estos archivos está asociado a su respectivo controlador. Por ejemplo, en el archivo "user.routes.js", se encuentran todas las rutas relacionadas con la entidad "User". Esta metodología contribuye a una gestión más eficiente del código, mejorando la claridad al agrupar las rutas vinculadas a la entidad de usuario en un archivo exclusivo.

A continuación, en el **Código 2.15**, se puede observar las rutas definidas para la entidad “User”.

Código 2.15 Rutas definidas para la entidad "User"

```
const UserController = require('../controllers/user.controller');
const multer = require('multer');
const path = require('path');

const storage = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, path.join(__dirname, '..', 'Imagenes/'));
    },
    filename: function(req, file, cb) {
        cb(null, file.fieldname + '-' + Date.now() +
path.extname(file.originalname));
    }
});
const upload = multer({ storage: storage });

module.exports = function(app){
    app.post('/api/user/new', UserController.createUser);
    app.get('/api/users', UserController.getAllUsers);
    app.get('/api/user/:id',UserController.getUser);
    app.put('/api/user/:id/',UserController.updateUser);
    app.delete('/api/user/:id', UserController.deleteUser);
    app.put('/api/user/foto/:id', upload.single('foto'),
UserController.addPhoto);
    app.get('/api/user/foto/:id', UserController.getUserPhoto);
}
```

En este código, se configura multer para manejar el almacenamiento de archivos, se especifica la carpeta de destino y el formato del nombre de los archivos. Adicionalmente, se definen las rutas específicas para las peticiones CRUD.

2.2.4.3. PRUEBAS DEL SERVIDOR CON POSTMAN

Para poder realizar pruebas de funcionamiento de la API, es necesario previamente incluir las rutas definidas de cada uno de los archivos mencionados anteriormente, para ello, se programa la configuración del archivo "server.js" se la siguiente manera como se observa en el **Código 2.16**:

Código 2.16 Configuración del archivo server.js

```
const express = require('express');
const path = require('path');
const cors = require('cors');

const app = express();

const port = 8000;

require('./config/mongoose.config.js');

app.use(cors());

app.use(express. Json());

app.use(express.urlencoded( { extended: true } ));

const allRolRoutes = require('./routes/rol.routes');
allRolRoutes(app);

const allUserRoutes = require('./routes/user.routes');
allUserRoutes(app);

const allSchoolRoutes = require('./routes/school.routes');
allSchoolRoutes(app);

const allAcadTrainingRoutes = require('./routes/acadTraining.routes');
allAcadTrainingRoutes(app);

const allCertificationRoutes = require('./routes/certification.routes');
allCertificationRoutes(app);

const allWorkExperienceRoutes = require('./routes/workExperience.routes');
allWorkExperienceRoutes(app);

const allCompanyRoutes = require('./routes/company.routes');
allCompanyRoutes(app);

const allJobRoutes = require('./routes/job.routes');
allJobRoutes(app);

const allPostulationRoutes = require('./routes/postulation.routes');
allPostulationRoutes(app);

const allLogInRoutes = require('./routes/login.routes');
allLogInRoutes(app);

app.use('/Imagenes', express.static(path.join(__dirname, 'Imagenes')));
```

```
app.listen(port, () => console.log("Server is listening at port ", port));
```

Como se puede observar en la **Figura 2.13**, se inicia el servidor mediante el comando ***npm run server***, observando en la consola el puerto para las solicitudes y la conexión establecida con la base de datos:

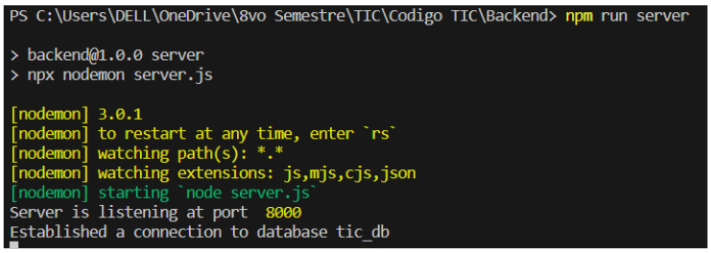

Figura 2.13 Inicialización del servidor

Se procede a realizar una prueba para agregar un nuevo registro a la colección "User". Para ello, se accede a Postman y se crea una nueva solicitud POST dirigida a http://localhost:8000/api/user/new, incluyendo los parámetros en formato JSON como se observar en la **Figura 2.14**.

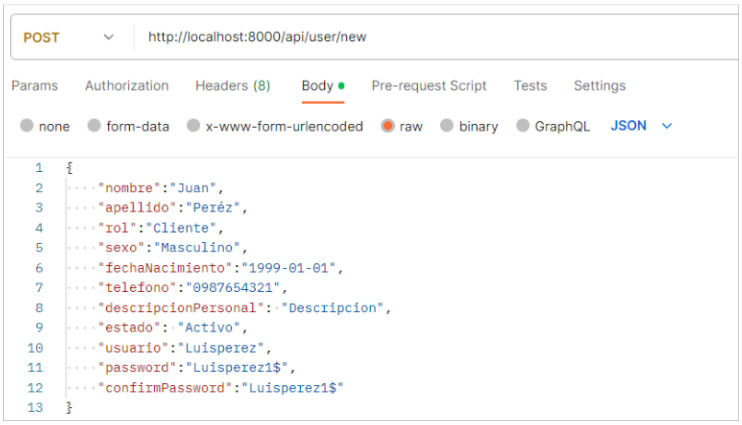

Figura 2.14 Petición para crear un nuevo usuario realizada con Postman

Tras enviar la petición, el servidor responde con lo observado en la **Figura 2.15**.

```
{
    "insertedUser": {
        "nombre": "Juan",
        "apellido": "Peréz",
        "rol": "Cliente",
        "sexo": "Masculino",
        "fechaNacimiento": "1999-01-01T00:00:00.000Z",
        "telefono": "0987654321",
        "descripcionPersonal": "Descripcion",
        "estado": "Activo",
        "usuario": "Luisperez",
        "password": "706691d82382bce0e2451b33116ab758",
        "_id": "65a5ed290682ce9784320fb1",
        "__v": 0
    },
    "msg": "Succesful creation"
}
```


Figura 2.15 Respuesta del servidor

2.2.5. IMPLEMENTACION DE LA VISTA

Como se indicó previamente, la Vista se desarrollará utilizando Flutter. Para iniciar este proceso, en Visual Studio, se utiliza la combinación de teclas **Ctrl+Shift+P** para abrir la Paleta de Comandos. Luego, se introduce Flutter: new Project para iniciar la creación de un nuevo proyecto.

2.2.5.1. ESTRUCTURA DE CARPETAS

Para abordar el proyecto, es importante comprender bien el scaffolding (o estructura de carpetas). Aunque esta disposición puede variar de un proyecto a otro, se utilizará la estructura descrita a continuación en la **Tabla 2.15**.

Tabla 2.15 Estructura de archivos del proyecto para el frontend

```
-Frontend/                          #Directorio raíz del proyecto
   -  lib/
         -  classes/                 #Clases para mapeo JSON>Lista
         -  pages/                   #Interfaces de la app
         -  routes/                  #Definición de rutas nombradas
         -  services/                #Peticiones HTTP
   -  main.dart                      #Archivo principal
```

2.2.5.2. MAPEO BIDIRECCIONAL DE OBJETOS JSON

Con el fin de asegurar una correcta interacción con la API, se generarán clases que faciliten la conversión de datos asociados a las entidades de la base de datos, como se demuestra en el **Código 2.17** con la clase correspondiente a la entidad "School". En dicho código, se destacan las funciones "schoolFromJson" y "schoolToJson", las cuales posibilitan la transformación bidireccional de datos entre su representación en formato JSON y objetos Dart del tipo School.

Código 2.17 Clase para el mapeo de objetos del tipo "School"

```dart
import 'dart:convert';

List<School> schoolFromJson(String str) =>
    List<School>.from(json.decode(str).map((x) => School.fromJson(x)));

String schoolToJson(List<School> data) =>
    json.encode(List<dynamic>.from(data.map((x) => x.toJson())));

class School {
  String id;
  String nombreInstitucion;
  String ubicacion;
  String estado;
  int v;

  School({
    required this.id,
    required this.nombreInstitucion,
    required this.ubicacion,
    required this.estado,
    required this.v,
  });

  factory School.fromJson(Map<String, dynamic> json) => School(
        id: json["_id"],
        nombreInstitucion: json["nombreInstitucion"],
        ubicacion: json["ubicacion"],
        estado: json['estado'],
        v: json["__v"],
      );

  Map<String, dynamic> toJson() => {
        "_id": id,
        "nombreInstitucion": nombreInstitucion,
        "ubicacion": ubicacion,
        "estado": estado,
      };
}
```

2.2.5.3. CONEXIÓN CON LA API

Con el fin de establecer una conexión con la API, se introducirán clases que incorporen métodos HTTP para realizar peticiones a cada ruta definida en la capa Controlador. Estos métodos harán uso de las funciones detalladas anteriormente, facilitando así la serialización y deserialización de datos.

En el **Código 2.18**, se puede observar la función que realiza la petición GET al método **getAllSchools** definido en el controlador de Instituciones Educativas.

Código 2.18 Petición GET para obtener las Instituciones Educativas

```
Future<List<School>> getAllSchools() async => http
        .get(Uri.parse("https://0vmlb023-
8000.use2.devtunnels.ms/api/schools"))
        .then((res) {
      if (res.statusCode == 200) {
        String body = utf8.decode(res.bodyBytes);
        return schoolFromJson(body);
      } else {
        throw Exception("Conexión fallida");
      }
    });
```

En el código, se observa que se hace uso de la función "**schoolFromJson**" para decodificar la respuesta y la convierte en una lista de objetos Dart del tipo "**School**".

De la misma manera, se definieron funciones para realizar peticiones POST, PUT y DELETE hacia los controladores definidos anteriormente. En el **Código 2.19**, se muestra la función para agregar una Institución Educativa.

Código 2.19 Petición POST para crear una Institución Educativa

```
void createSchool(School school) async {
  try {
    var response = await http.post(
        Uri.parse('https://0vmlb023-
8000.use2.devtunnels.ms/api/school/new'),
        body: {
          'nombreInstitucion': school.nombreInstitucion,
          'ubicacion': school.ubicacion,
          'estado': school.estado
        });
  } catch (e) {
    print(e);
  }
}
```

2.2.5.4. DEFINICION DE RUTAS

La rutas de la aplicación están organizadas de acuerdo con las especificaciones detalladas en la sección **Enrutamiento** del Diseño de la Vista.

- **Rutas nombradas**:

El **Código 2.20**, se presenta una clase llamada "Routes" que contiene un mapa de rutas en una aplicación Flutter. Cada ruta se asocia con una función que devuelve un widget de Flutter correspondiente a esa ruta. Estas rutas están vinculadas a diferentes páginas

de la aplicación, como la página inicial y las páginas de inicio de sesión de usuario y empresa.

Código 2.20 Clase Routes para realizar el mapeo de rutas nombradas

```
import 'package:flutter/material.dart';
import 'package:frontend/pages/pages.dart';

class Routes {
  static Map<String, Widget Function(BuildContext)> routes = {
    'start': (_) => const initialPage(),
    'home': (_) => const loginPage(),
    'userSignIn': (_) => const userSignIn(),
    'companySignIn': (_) => const companySignIn(),
  };
}
```

Como se puede observar en el **Código 2.21**, esta clase resulta beneficiosa, ya que posibilita la invocación de interfaces simplemente utilizando el nombre asociado a ellas.

Código 2.21 Ruta nombrada

```
Navigator.pushNamed(context, userSignIn');
```

- **Rutas dadas**: Como se ve en el **Código 2.22**, estas rutas apuntan a interfaces que necesitan de al menos un argumento para poder desplegarse.

Código 2.22 Ruta dada

```
Navigator.of(context)
    .push(MaterialPageRoute<Null>(
        builder:(BuildContext context) {
            return homePageUser(authUser);}));
```

2.2.5.5. CODIFICACIÓN DE INTERFACES

La estructura de la interfaz de usuario se fundamenta en tres interfaces clave: Inicio de sesión, Registro de Usuario o Compañía e Interfaz Inicial. Esta última brinda acceso a los módulos adicionales mencionados en el alcance del proyecto, desde los cuales se puede acceder a otros módulos o regresar a la Interfaz Inicial. Sin embargo, debido a la extensión de los códigos, esta sección se enfocará exclusivamente en la implementación del módulo de Datos de Usuario del Cliente; los demás módulos se abordarán en la sección de **Pruebas de funcionamiento**.

2.2.5.5.1. INTERFAZ INICIAL

La Interfaz Inicial de la aplicación está formada por diversos contenedores que, como se ha mencionado, proporcionan acceso a los diferentes módulos de la aplicación. En

su desarrollo, se emplea el widget denominado GestureDetector, el cual posibilita la detección de gestos del usuario, específicamente el gesto de toque (onTap). Al tocar el contenedor (Container) vinculado al GestureDetector, se activa una acción que conlleva la transición a la pantalla deseada. En el **Código 2.23** se encuentra la implementación del GestureDetector que facilita la transición al módulo de Datos de Usuario.

Código 2.23 Botón para acceder al módulo de Datos de Usuario

```
GestureDetector(
  onTap: () {
    Navigator.of(context).push(MaterialPageRoute<Null>(
      builder: (BuildContext context) {
        return userProfile(widget.loggedUser);
      },
    ));
  },
  child: Container(
    width: MediaQuery.of(context).size.width * 0.45,
    height: MediaQuery.of(context).size.height * 0.28,
    padding: EdgeInsets.only(left: 16.0, right: 16.0, top: 10, bottom: 10),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(8.0),
      border: Border.all(
        color: Color.fromRGBO(1, 167, 211, 1),
        width: 4.0,
      ),
      boxShadow: const [
        BoxShadow(
          color: Colors.black26,
          offset: Offset(8.0, 8.0),
          blurRadius: 15.0,
        ),
      ],
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Container(
          padding: EdgeInsets.all(8),
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            color: Color.fromRGBO(1, 167, 211, 1),
          ),
          width: 100,
          height: 100,
          child: Image(
            image: AssetImage('assets/img/usuario.png'),
            fit: BoxFit.contain,
          ),
        ),
        Container(
          child: Text(
            "Mi perfil",
            style: TextStyle(
              fontWeight: FontWeight.w700,
```

```
              fontSize: 17.0,
            ),
          ),
        ),
      ],
    ),
  ),
),
```

En la **Figura 2.16**, se puede observar su representación dentro de la aplicación:

Figura 2.16 Botón de acceso a la interfaz de Datos de Usuario

2.2.5.5.2. INTERFAZ DE DATOS DE USUARIO

La interfaz del módulo de Datos de Usuario se encuentra dividida en cuatro secciones: Datos personales, Formación Académica, Certificaciones y Experiencia Laboral. A continuación, se detalla la implementación de algunas de estas secciones.

Datos personales: Como se observar en la **Figura 2.17**, la sección Datos personales contiene toda la información referente al usuario, como nombre, fecha de nacimiento, numero de contacto, entre otros. Adicionalmente, se puede observar la foto de perfil asociada al usuario o una imagen por defecto, en caso de que no cuente con una.

Figura 2.17 Sección “Datos personales” del módulo de Datos de Usuario

El botón "Agregar foto" activa una ventana de diálogo que proporciona al usuario la opción de tomar una fotografía o elegir una de la galería. Esto se puede observar en la **Figura 2.18**.

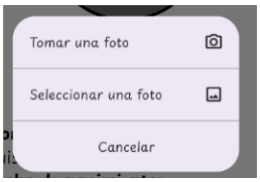

Figura 2.18 Ventana de diálogo para elegir el origen de la foto de perfil

Certificaciones: En este segmento, observado en la **Figura 2.19**, se despliega la relación de certificados adquiridos por el usuario, detallando el título, la fecha de obtención y un enlace que facilita la visualización del certificado.

Figura 2.19 Sección "Certificaciones" del módulo de Datos de Usuario

Las secciones Formación Académica y Experiencia Laboral detallan la lista de logros educativos y trayectoria profesional del usuario, respectivamente. El código para la implementación de este módulo y los demás puede ser observado en el **Anexo III**.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En este capítulo se expondrán los resultados obtenidos tras completar la aplicación, además del análisis de las pruebas efectuadas por los usuarios finales de la Junta Parroquial.

3.1. RESULTADOS

3.1.1. ACTUALIZACIÓN DEL TABLERO KANBAN

Se realiza la actualización del tablero Kanban, trasladando las tareas de la fase de Implementación a la lista de tareas finalizadas y la tarea de la fase de Pruebas a la lista de tareas en progreso. En la **Figura 3.1** se puede visualizar el tablero actualizado.

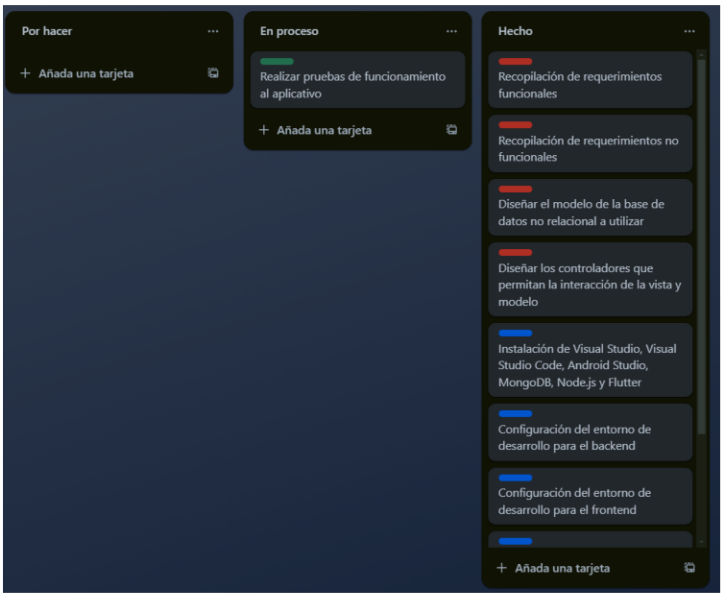

Figura 3.1 Tablero Kanban para la fase de Pruebas

3.1.2. PRUEBAS DE FUNCIONAMIENTO

En esta sección se describirán las pruebas efectuadas en la aplicación móvil, donde se verificarán las funcionalidades y validaciones implementadas en cada una de las interfaces accesibles para el usuario.

3.1.2.1. MÓDULO DE REGISTRO Y AUTENTICACIÓN

El módulo de registro incluye las validaciones necesarias para que los usuarios o empresas ingresen correctamente sus datos, así como la verificación de la contraseña mediante su repetición en un segundo campo de texto, en caso de que las contraseñas no coincidan o no cumplan con el formato requerido se notifica al usuario mediante una alerta. En las **Figura 3.2 y 3.3**, se pueden observar estas validaciones.

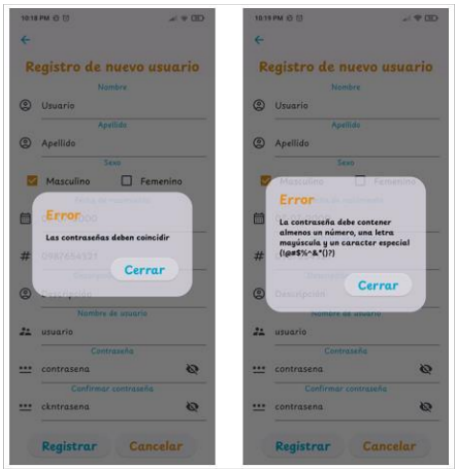

Figura 3.2 Validaciones implementadas en el módulo de registro de Usuario

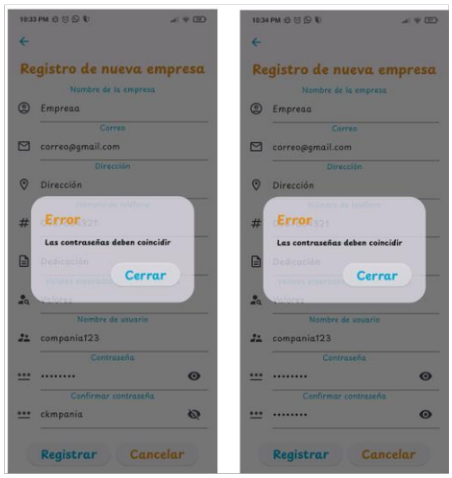

Figura 3.3 Validaciones implementadas en el módulo de registro de Empresa

El módulo de autenticación permite verificar que la información ingresada en los campos coincida con la proporcionada previamente en el módulo de registro para todos los roles de usuario. En caso de no ser así, se emite una alerta notificando al usuario, esto se muestra en la **Figura 3.4** donde se verifica dicha validación.

Figura 3.4 Validaciones implementadas en el módulo de autenticación

3.1.2.2. MÓDULO DE DATOS DE USUARIO

3.1.2.2.1. PERFIL DE USUARIO

El módulo de datos de usuario facilita la visualización y gestión de la información que los usuarios o empresas consideren relevante. Tanto los usuarios con roles de Cliente como los de Administrador comparten las mismas funcionalidades, ya que en sus perfiles pueden visualizar información personal, académica y laboral. En contraste, los usuarios con el rol de Empresa solo pueden acceder a información relacionada con la entidad que representan.

3.1.2.2.1.1. ROLES ADMINISTRADOR Y CLIENTE

El módulo se estructura en cuatro secciones distintas: Datos personales, Formación Académica, Certificaciones y Experiencia Laboral. Cada una de estas secciones ofrece funcionalidades que posibilitan la creación, edición o eliminación de registros dentro de su área correspondiente.

- **Datos personales**

  Esta sección permite al usuario ver y editar su información personal en una interfaz con las mismas validaciones del módulo de registro. Si los campos de contraseña están vacíos, se le pregunta al usuario si desea conservar la contraseña actual. Además, se puede cambiar la foto de perfil seleccionando su origen desde la galería o la cámara mediante una ventana emergente. Esto puede observarse en la **Figura 3.5**.

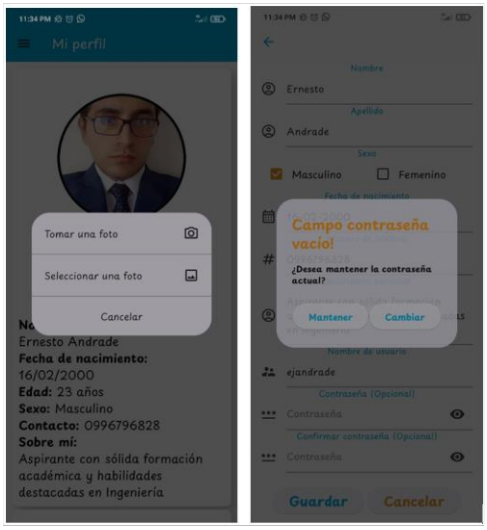

Figura 3.5 Funcionalidades y validaciones implementadas en la sección de Datos personales

- **Formación académica, Certificaciones y Experiencia laboral**

  Las 3 secciones restantes comparten la misma estructura funcional, ya que permiten acceder a dos interfaces en las que se puede editar, eliminar o agregar nuevos registros. Como se puede observar en las **Figuras 3.6 y 3.7**, para cada una de estas posibilidades, se han incluido alertas que notifican si todos los campos han sido ingresados correctamente o realizar una confirmación previa a la eliminación de un registro.

Figura 3.6 Funcionalidades implementadas en la secciones Formación Académica, Certificaciones y Experiencia Laboral

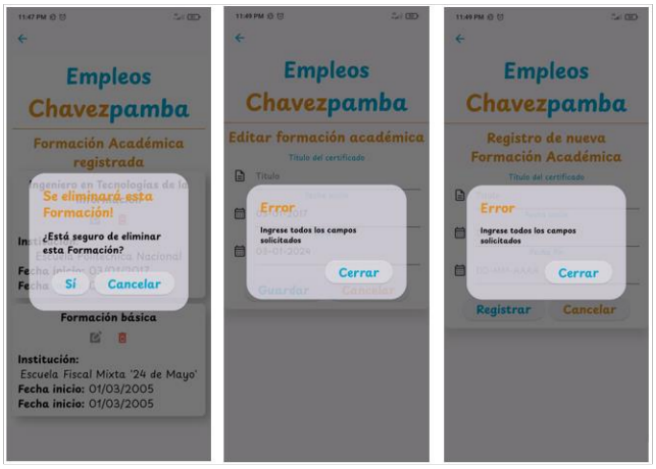

Figura 3.7 Validaciones implementadas

3.1.2.2.1.2. ROL EMPRESA

Como se ha indicado previamente, el módulo dispone de una única sección que presenta la información de la entidad junto con una foto de perfil, la cual puede ser editada siguiendo el mismo proceso descrito en la sección anterior. Del mismo modo que para los roles anteriores, al modificar la información de la empresa, se ofrece la opción de mantener la contraseña actual. Esto se observa en la **Figura 3.8**.

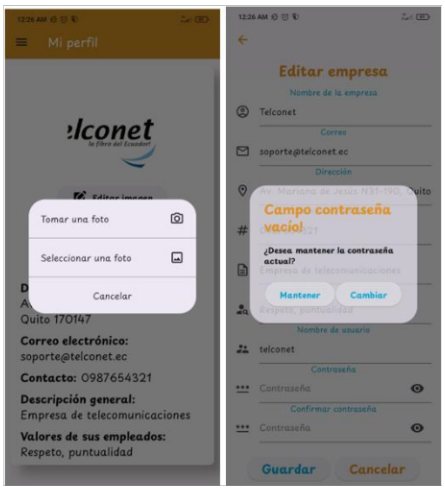

Figura 3.8 Funcionalidades y validaciones implementadas

3.1.2.2.2. EXPLORACIÓN DE USUARIOS

Este módulo permite a todos los usuarios explorar los perfiles de los usuarios registrados en la aplicación. Sin embargo, se establece una distinción entre los roles de Cliente y Empresa, y el de Administrador, ya que este último dispone de funcionalidades adicionales.

3.1.2.2.2.1. ROLES CLIENTE Y EMPRESA

El módulo se encuentra dividido en dos secciones que distinguen a los usuarios entre Clientes y Empresas. En la **Figura 3.9,** se observa que cada usuario registrado se representa por medio de su foto, nombre y un botón que permite acceder a su perfil.

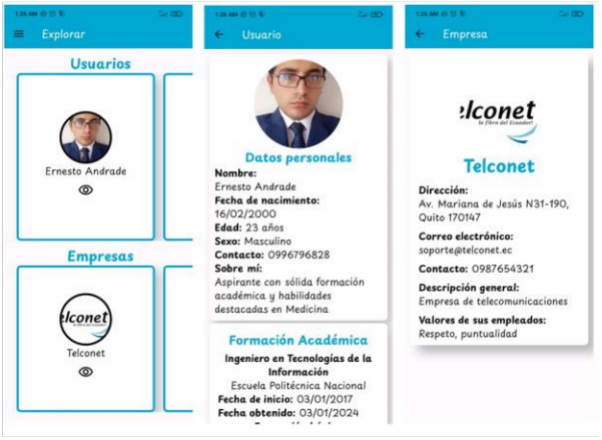

Figura 3.9 Módulo de exploración de usuarios para Clientes y Empresas

3.1.2.2.2.2. ROL ADMINISTRADOR

En la **Figura 3.10**, se observa que el módulo cuenta con la misma distribución mencionada en la sección anterior, sin embargo, al tratarse del rol Administrador, se cuenta con dos botones adicionales que permiten: editar la información del Cliente o Empresa y habilitar o inhabilitar a los usuarios. La edición de los perfiles de usuarios sigue la misma lógica descrita en las secciones anteriores.

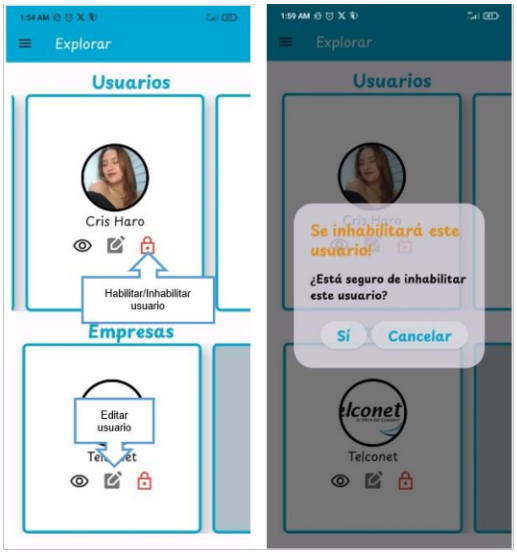

Figura 3.10 Módulo de exploración de usuarios para Administradores

Es relevante señalar que cuando un usuario se inhabilita, su perfil se muestra con un tono ligeramente más opaco en comparación con los demás, tal como se puede apreciar en la **Figura 3.11**.

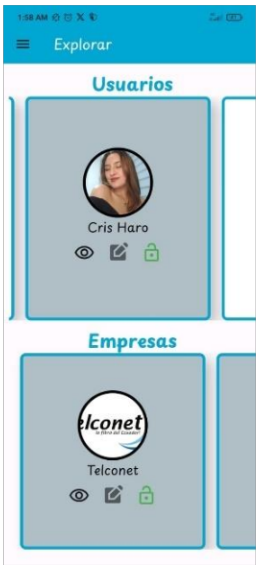

Figura 3.11 Vista de usuario inhabilitado

3.1.2.3. MÓDULO DE EMPLEOS

El módulo de Empleos ofrece a los usuarios la capacidad de realizar todas las funcionalidades relacionadas con un empleo. Se diferencian las funcionalidades según el rol de usuario en las siguientes secciones.

3.1.2.3.1. ROL CLIENTE

El módulo permite a los Clientes visualizar una lista de todos los empleos publicados por una Empresa registrada, además de poder acceder a los detalles específicos de cada empleo utilizando el botón "*Más detalles*". Esto puede ser verificado en la **Figura 3.12**.

Figura 3.12 Módulo de Empleos para Clientes – Lista de Empleos

Además, un Cliente tiene la opción de enviar una postulación a los empleos que le resulten atractivos utilizando el botón "Me interesa". Posteriormente, como se observa

en la **Figura 3.13,** tendrá acceso a una interfaz donde se muestra la lista de postulaciones realizadas, permitiéndole verificar el estado actual de cada una y, en caso de haber sido rechazada, visualizar el motivo de la negativa.

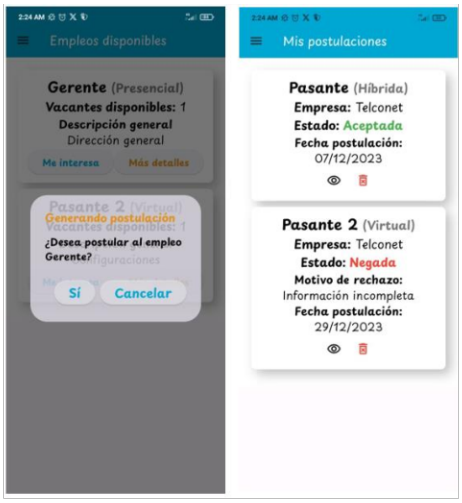

Figura 3.13 Módulo de Empleos para Clientes – Postulaciones

Finalmente, las postulaciones pueden ser eliminadas en cualquier momento que el usuario lo considere; sin embargo, no se eliminan definitivamente hasta que el administrador las apruebe. En este caso, pasan a un estado inactivo, y el usuario ya no podrá verlas en su lista de postulaciones como se observa en la **Figura 3.14**.

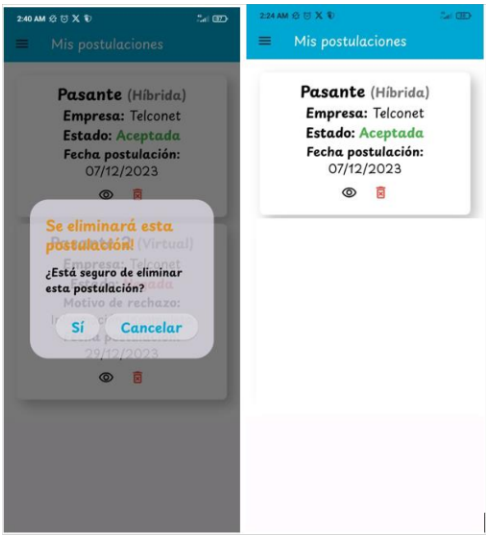

Figura 3.14 Módulo de Empleos para Clientes – Eliminación de Postulaciones

3.1.2.3.2. ROL EMPRESA

El módulo permite a una Empresa visualizar la lista de los empleos que haya publicado, así como acceder a la lista de postulaciones que cada empleo haya recibido y acceder al perfil del usuario que haya realizado la postulación. Esto se puede visualizar en la **Figura 3.15**.

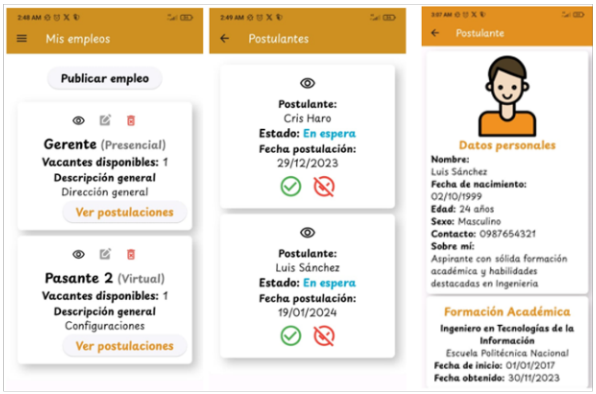

Figura 3.15 Módulo de Empleos para Empresas – Lista de empleos, Postulaciones y Postulante

Además, la empresa ofertante tiene la capacidad de aceptar o rechazar las postulaciones según su criterio. En caso de rechazo, como se observa en la **Figura 3.16**, se requerirá que proporcione un motivo para la negación.

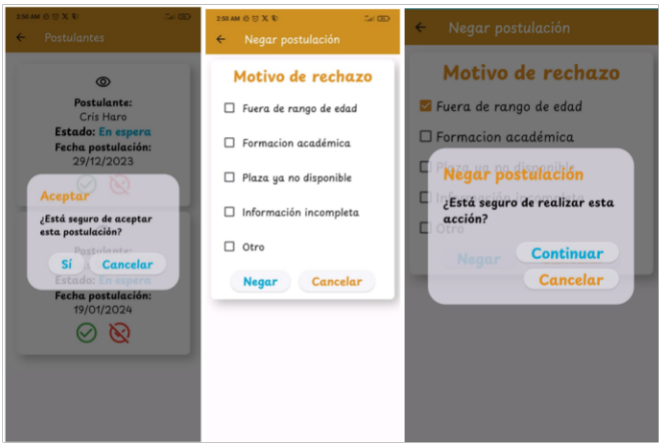

Figura 3.16 Módulo de Empleos para Empresas – Aceptar/Negar Postulaciones

Finalmente, como se observa en la **Figura 3.17**, la empresa ofertante puede modificar los empleos publicados, agregar nuevos empleos utilizando el botón "Publicar empleo" y, al eliminarlos, pasan al estado inactivo y deja de ser visible parala Empresa al igual que sus postulaciones.

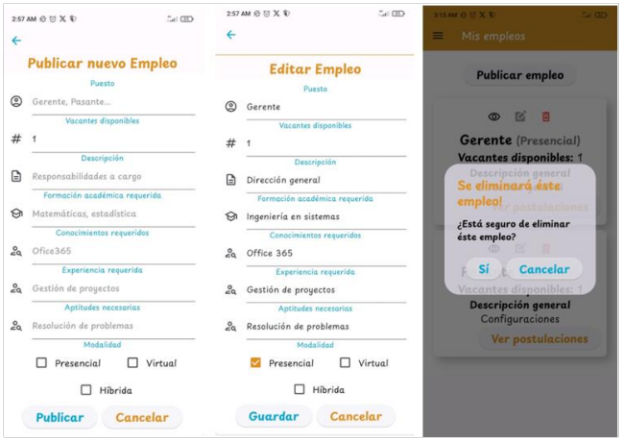

Figura 3.17 Módulo de Empleos para Empresas – Publicar/Editar/Eliminar Empleos

3.1.2.3.3. ROL ADMINISTRADOR

El módulo de Empleos, como se observa en la **Figura 3.18**, para el rol de Administrador es muy similar al que se observa para el rol de empresa. Permite al usuario visualizar una lista de los empleos publicados por todas las empresas registradas y, al igual que las empresas, acceder a la lista de postulaciones recibidas. Sin embargo, al igual que en el **módulo de exploración de usuarios para el administrador**, los empleos que hayan sido "eliminados" por las empresas se muestran con un tono opaco que los identifica, lo mismo ocurre con sus postulaciones "eliminadas".

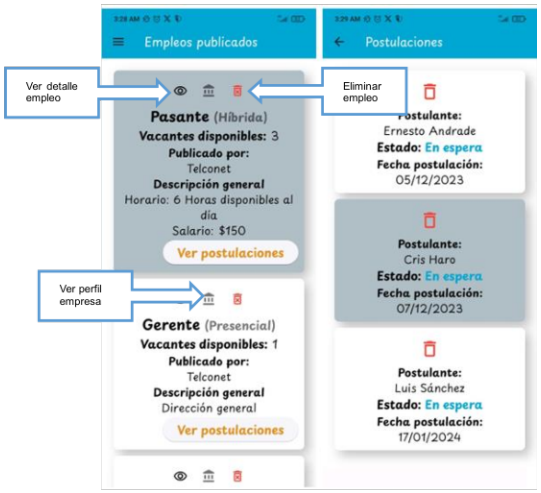

Figura 3.18 Módulo de Empleos para Administradores – Lista de Empleos y Postulaciones

Adicionalmente, al igual que se observó en la sección anterior, en la **Figura 3.19**, se puede verificar que se puede acceder al detalle del empleo publicado, así como al detalle de la empresa ofertante.

Figura 3.19 Módulo de Empleos para Administradores – Detalle Empleo y Empresa

Finalmente, se observa en la **Figura 3.20** que el Administrador puede eliminar definitivamente los empleos y postulaciones cuando lo considere apropiado.

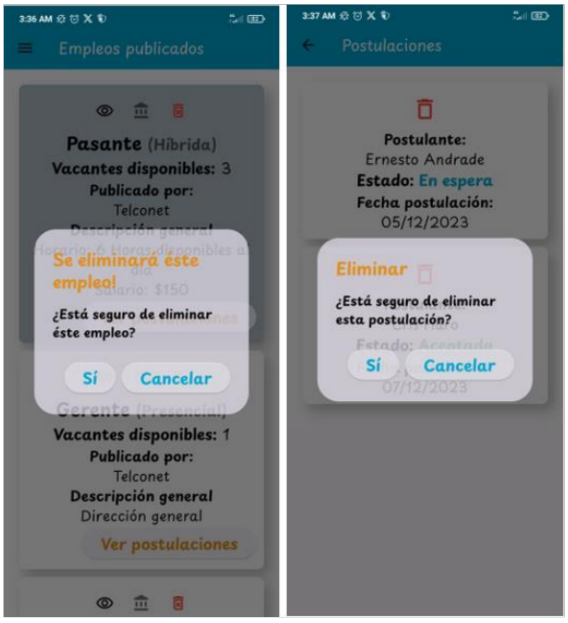

Figura 3.20 Módulo de Empleos para Administradores – Eliminación de Empleos y Postulaciones

3.1.2.4. MÓDULO DE REPORTES

Como se mencionó anteriormente, el módulo de reportes está disponible exclusivamente para usuarios con el rol de Administrador. Este módulo ofrece información sobre el número de usuarios registrados, diferenciándolos entre Clientes y Empresas. Además, presenta un gráfico estadístico que muestra el estado de las postulaciones registradas y otro que detalla los motivos de rechazo más comunes. Esto se puede observar en la **Figura 3.21**.

Figura 3.21 Módulo de reportes

3.1.3. VALIDACIÓN DE REQUERIMIENTOS

3.1.3.1. REQUERIMIENTOS FUNCIONALES

La validación de los requerimientos funcionales se realizó mediante un enfoque basado en encuestas de satisfacción dirigidas a usuarios finales, tanto de la Junta Parroquial como ajenos a esta, tras haber probado la aplicación durante un periodo apropiado.

La encuesta se compone de varias preguntas que están enfocadas en la validación de los requerimientos funcionales. En la **Tabla 3.1**, se puede observar el resumen de las respuestas obtenidas en la encuesta.

Tabla 3.1 Resumen del cumplimiento de los requerimientos funcionales

RF	Pregunta	Respuestas		Validación
		Si%	No%	
RF01	¿Pudo crear una cuenta en la aplicación sin problemas?	100	0	**Cumple**

RF02	¿Pudo acceder a la aplicación ingresando su usuario y contraseña sin problemas?	100	0	**Cumple**
RF03	¿Pudo acceder a su perfil de usuario y editar la información presentada?	100	0	**Cumple**
RF04	¿Pudo visualizar y postular a la lista de empleos disponibles?	100	0	**Cumple**
	Como Empresa, ¿pudo crear, ver, editar y eliminar sus ofertas de trabajo?	100	0	**Cumple**
	Como Empresa, ¿pudo visualizar el perfil de los postulantes a un empleo?	100	0	**Cumple**
	Como Empresa, ¿pudo aceptar o negar las postulaciones a un empleo? Al negar, ¿pudo seleccionar el motivo de rechazo?	100	0	**Cumple**
	Como Administrador, ¿pudo visualizar la lista de empleos y sus postulaciones, así como eliminarlas?	100	0	**Cumple**
RF05	Como Cliente, ¿pudo visualizar sus postulaciones y eliminarlas?	100	0	**Cumple**
RF06	¿Pudo visualizar el perfil de los demás usuarios y empresas?	100	0	**Cumple**
RF07	Como Administrador, ¿pudo ver, editar y habilitar o inhabilitar los Usuarios y Empresas registradas?	100	0	**Cumple**
RF08	Como Administrador, ¿pudo visualizar el módulo de reportes?	100	0	**Cumple**
	En el módulo de reportes, ¿pudo visualizar el número de usuarios y empresas registrados?	100	0	**Cumple**
	En el módulo de reportes, ¿pudo visualizar el resumen de postulaciones clasificadas según el estado y de motivos de rechazo de las postulaciones de usuarios?	100	0	**Cumple**

Es relevante resaltar que todas las preguntas han obtenido un índice del 100% en la opción "Sí", lo que indica que el funcionamiento del aplicativo cumple con los requerimientos funcionales recopilados durante la fase de Diseño.

Los resultados obtenidos de las encuestas pueden ser observados en el **Anexo IV**.

3.1.3.2. REQUERIMIENTOS NO FUNCIONALES

Para la validación del requerimiento no funcional RNF01: Interfaz Intuitiva, se añadieron preguntas que permitieran validar la experiencia del usuario en cuanto a la usabilidad del prototipo. El resumen de las respuestas obtenidas en la encuesta se muestra en la **Tabla 3.2**.

Tabla 3.2 Resumen del cumplimiento del requerimiento funcional RNF01

RNF	Pregunta	Respuestas		Validación
		Si%	No%	
RNF01	¿Cree que las opciones de navegación del aplicativo son fáciles de entender para personas de todas las edades?	100	0	**Cumple**
	¿Encuentra usted que los iconos y botones del aplicativo son claros y fácilmente reconocibles?	100	0	**Cumple**
	¿Ha experimentado dificultades al intentar realizar tareas básicas dentro del aplicativo, como buscar información o completar formularios?	0	100	**Cumple**
	¿Encuentra usted que el diseño general del aplicativo es simple y directo?	100	0	**Cumple**

Las respuestas positivas indican que el aplicativo móvil ha logrado su objetivo de ofrecer interfaces intuitivas y una navegación fluida entre los distintos módulos. Esto asegura una experiencia de usuario satisfactoria y coherente con las expectativas establecidas en la fase de diseño. Los resultados obtenidos de las encuestas pueden ser observados en el **Anexo IV**.

En cuanto al requerimiento no funcional RNF02, que especifica la modularidad de la aplicación, su evaluación no se lleva a cabo mediante pruebas directas, dado que la modularidad se refiere a aspectos internos relacionados con la estructura y el diseño del software. Dado que esta característica no es perceptible para el usuario, la verificación de este requerimiento se logra mediante el análisis de la flexibilidad y facilidad que presenta el aplicativo para la integración de nuevos módulos sin afectar el funcionamiento de los existentes. Esto asegura que el sistema pueda adaptarse y expandirse según las futuras necesidades de la Junta Parroquial.

3.1.4. ACTUALIZACIÓN FINAL DEL TABLERO KANBAN

En este punto, se han concluido todas las pruebas realizadas por los usuarios de la Junta Parroquial, lo que ha permitido validar los requerimientos funcionales de manera satisfactoria. Todas las tareas se encuentran en la lista de tareas finalizadas, lo que indica que el desarrollo del aplicativo ha concluido; esta finalización se refleja en la **Figura 3.22**, donde se observa el tablero Kanban completado.

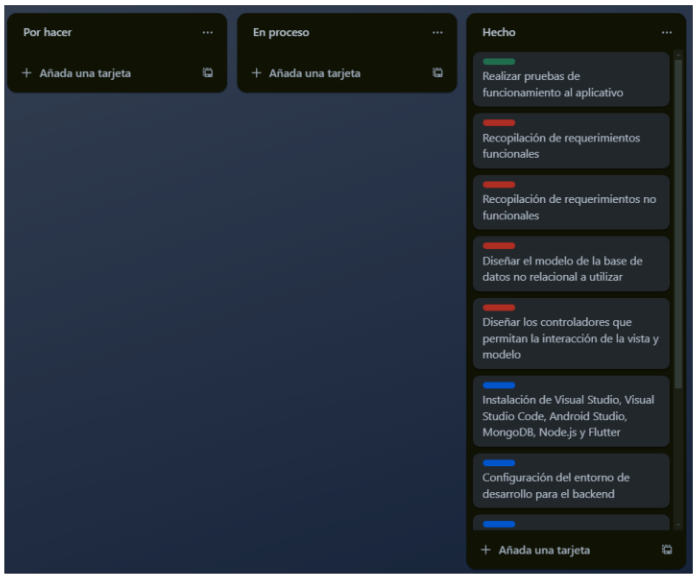

Figura 3.22 Finalización del tablero Kanban

3.2. CONCLUSIONES

La realización de este Trabajo de Integración Curricular ha posibilitado la creación de un prototipo de aplicación móvil diseñado para dispositivos Android. Esta aplicación facilita la recolección y conservación de datos esenciales para la gestión de una bolsa de empleo en la Junta Parroquial.

Las validaciones incorporadas en las interfaces del prototipo durante la recolección de información por parte de los usuarios evitan que se almacenen datos incorrectos en la base de datos.

Optar por herramientas como MongoDB y Flutter ha sido una decisión acertada, lo que ha permitido lograr resultados exitosos al implementar una arquitectura integrada con el patrón MVC. Estas herramientas se han demostrado altamente útiles en la implementación de la base de datos, la lógica de funcionamiento y la presentación del prototipo de manera eficiente.

Los resultados recolectados a través de la encuesta de satisfacción corroboran que el prototipo de la aplicación móvil ha sido implementado de manera correcta, cumpliendo de manera satisfactoria con los requisitos establecidos por los usuarios para simplificar y mejorar la gestión de la bolsa de empleos de la Junta Parroquial.

3.3. RECOMENDACIONES

Debido a que el aplicativo actualmente es un prototipo, solo permite el registro a través de la introducción manual de los datos de los usuarios. Para futuras mejoras, se sugiere

incorporar la opción de registro mediante otros proveedores, como por ejemplo Gmail, Facebook u otros similares.

Para implementar en futuros desarrollos, se sugiere considerar la inclusión de acceso mediante huella digital y la capacidad de conservar contraseñas de manera segura. Estas adiciones podrían mejorar la experiencia del usuario al proporcionar métodos de autenticación más convenientes y seguros.

Se sugiere que el prototipo sea instalado y utilizado en dispositivos Android que tengan una versión igual o superior a la 10. Esto asegurará un rendimiento adecuado del aplicativo y una mejor compatibilidad con el sistema operativo.

Se recomienda como una futura línea de desarrollo considerar la expansión del prototipo a otros sistemas operativos, como iOS. Esta expansión permitirá ampliar el alcance del aplicativo, facilitando así su acceso y utilización por parte de una audiencia más amplia de usuarios, lo que enriquecería significativamente el impacto y la utilidad del prototipo.

4. REFERENCIAS BIBLIOGRÁFICAS

[1] G. M. M. GUADALUPE, «USOS Y TIPOS DE APLICACIONES MÓVILES,» Salina Cruz, 2015.

[2] J. Astigarraga y V. Cruz-Alonso, «¡Se puede entender cómo funcionan Git y GitHub!,» ECOS, vol. 31, nº 1, p. 2332, 2022.

[3] B. M. Montero, H. V. Cevallos y J. D. Cuesta, «Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software,» Espirales. Revista multidisciplinaria de investigación, vol. 2, nº 17, p. 3, 2018.

[4] E. G. Maida y J. Pacienzia, «Metodologías de desarrollo de software,» 2015. [En línea]. Available: https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf.

[5] M. D. A. Serna, L. F. C. Zapata y J. A. Z. Cortes, «Mejoramiento de procesos de manufactura utilizando Kanban,» Ingenierías Universidad de Medellín,, vol. 14, nº 27, pp. 221-234, 2015.

[6] E. D. Y. Llerena y K. F. A. Guillen, «APLICACIÓN DE LA METODOLOGÍA KANBAN EN EL DESARROLLO DEL SOFTWARE PARA GENERACIÓN, VALIDACIÓN Y ACTUALIZACIÓN DE REACTIVOS, INTEGRADO AL SISTEMA INFORMÁTICO DE CONTROL ACADÉMICO UNACH.,» 2020. [En línea]. Available: https://goo.su/RflondY.

[7] P. J. Izquierdo, «Trabajo Fin de Grado, Integración de tableros Kanban en una herramienta que apoya la gestión ágil del trabajo,» 2019. [En línea]. Available: https://goo.su/0zWpl.

[8] Y. D. González y Y. F. Romero, «Patrón Modelo-Vista-Controlador.,» Revista Telem@tica, vol. 11, nº 1, pp. 47-57, 2012.

[9] L. R. Quisaguano Collaguazo , M. S. Pallasco Venegas , A. A. Andaluz Guerrero , M. N. Martines Freire y S. H. Corrales Beltrán, «Desarrollo híbrido con flutter,» Ciencia Latina Revista Científica Multidisciplinar, vol. 6, nº 4, pp. 4594-4609, 2022.

[10] «Dart documentation,» [En línea]. Available: https://dart.dev/guides.

[11] Microsoft, «Visual Code Documentation,» [En línea]. Available: https://code.visualstudio.com/docs.

[12] Postman, «What is Postman?,» [En línea]. Available: https://www.postman.com/product/what-is-postman/.

[13] S. Aguirre, JSON - Vol.1: Primeros pasos - Sintaxis - Tipos de datos, Ciudad Autónoma de Buenos Aires: Plandos, 2020.

[14] M. P. Usaola, MongoDB: gestión, administración y desarrollo de aplicaciones, La Mancha.

[15] K. Calvo, J. Durán, E. Quirós y E. Malinowski., MongoDB: alternativas de implementar y consultar documentos, San José.

[16] L. Puciarelli, Node JS - Vol. 1: Instalación - Arquitectura - node y npm, Ciudad Autonoma de Buenos Aires: Plandos, 2020.

[17] S. I. AGUAYO CÁCERES y C. V. FREIRE OROZCO, DESARROLLO DE UNA PLATAFORMA WEB DE COMERCIO ELECTRÓNICO B2C PARA LAS PYMES DE LA CIUDAD DE MACAS UTILIZANDO EL FRAMEWORK EXPRESS.JS, Riobamba, 2021.

[18] L. M. A. Hernández, V. A. P. Romero, S. A. S. González y J. A. V. Rodríguez, «Arquitectura REST para el desarrollo de aplicaciones web empresariales,» Revista Electrónica Sobre Tecnología, Educación Y Sociedad, vol. 8, nº 15, 2021.

[19] G. Developers, «Introducción a Android Studio,» 2023. [En línea]. Available: https://developer.android.com/studio/intro?hl=es-419.

[20] Atlassian. [En línea]. Available: https://trello.com/home.

[21] G. Tejero, «Casos de uso y diagramas de casos de uso,» 6 Septiempre 2021. [En línea]. Available: http://hdl.handle.net/10251/167637.

[22] MongoDB, «Database References,» [En línea]. Available: https://www.mongodb.com/docs/manual/reference/database-references/#database-references.

[23] DATENSEN, «MOON MODELER,» [En línea]. Available: https://www.datensen.com/data-modeling/moon-modeler-for-databases.html.

[24] Mongoose, «Schemas,» [En línea]. Available: https://mongoosejs.com/docs/guide.html.

[25] FastAPI, «CORS (Cross-Origin Resource Sharing),» [En línea]. Available: https://fastapi.tiangolo.com/tutorial/cors/.

[26] Espress, «Multer,» [En línea]. Available: https://expressjs.com/en/resources/middleware/multer.html.

[27] L. R. Quisaguano Collaguazo, M. S. Pallasco Venegas, A. A. Andaluz Guerrero, M. N. Martines Freire y S. H. Corrales Beltrán, «Desarrollo Hibrido con Flutter,» Ciencia Latina Revista Científica Multidisciplinar, vol. 6, nº 4, pp. 4594-4609, 2022.

5. ANEXOS

El presente documento incluye varios anexos que complementan y respaldan la información proporcionada en el cuerpo principal del texto. Los anexos proporcionan detalles adicionales, ejemplos prácticos y recursos complementarios relacionados con el tema tratado. A continuación, se detallan los anexos incluidos:

ANEXO I. BOCETOS DE LAS INTERFACES.

ANEXO II. CÓDIGO DEL PROYECTO PARA EL BACKEND.

ANEXO III. CÓDIGO DEL PROYECTO PARA EL FRONTEND.

ANEXO IV. RESULTADOS DE LA ENCUESTA.

ANEXO I. BOCETOS DE LAS INTERFACES

Enlace a los bocetos de las interfaces.

ANEXO II.CÓDIGO DEL PROYECTO PARA EL BACKEND

Enlace al repositorio del proyecto para el backend.

ANEXO III. CÓDIGO DEL PROYECTO PARA EL FRONTEND

Enlace al repositorio del proyecto para el frontend.

ANEXO IV. RESULTADOS DE LA ENCUESTA

Enlace a los resultados de las encuestas.