Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Desarrollo Ágil de Software para la Web

Trabajo Fin de Máster

Desarrollo de interfaz basado en frameworks para aplicación EVAH

Autor: Gregory Smith Rodríguez Helena
Tutor/es: José María Gutiérrez Martínez
Sergio de la Mata Moratilla

20__

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Máster Universitario en Desarrollo Ágil de Software para la Web

Trabajo Fin de Máster
Desarrollo de interfaz basado en frameworks para aplicación EVAH

Autor: Gregory Smith Rodríguez Helena
Tutor: José María Gutiérrez Martínez
Sergio de la Mata Moratilla

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA: 22 de septiembre de 2023

Agradecimientos

Quiero expresar mi profundo agradecimiento a todas las personas que contribuyeron de manera significativa a la realización de este Trabajo Fin de Máster. Sus apoyos, conocimientos y orientaciones fueron fundamentales para el éxito de este proyecto.

En primer lugar, quiero agradecer a mis asesores académico, José María Gutiérrez Martínez y Sergio de la Mata Moratilla, por su guía experta y su valioso tiempo dedicado a revisar y mejorar este trabajo. Sus comentarios críticos y sugerencias constructivas fueron esenciales para dar forma a esta investigación.

Además, quiero agradecer a mis compañeros de clase y amigos que me brindaron apoyo emocional y compartieron sus ideas y experiencias a lo largo de este proceso. Su colaboración fue inestimable.

También deseo expresar mi profunda gratitud hacia mi familia y la familia de mi esposa por su constante apoyo y comprensión durante los momentos de dedicación intensa a este proyecto.

Finalmente, agradezco a la Universidad de Alcalá por brindarme la oportunidad de llevar a cabo este trabajo y por su compromiso con la educación de calidad.

Sin la ayuda y el apoyo de todas estas personas, este trabajo no habría sido posible. Muchas gracias por su contribución a este logro.

Índice

Índice de imágenes

Índice de tablas

Resumen

Este documento presenta el estudio realizado sobre la modernización de la interfaz de usuario de la aplicación Evah. Con esto en mente, se ha desarrollado este proyecto con la idea de mejorar la eficiencia y la experiencia del usuario mediante la implementación de una interfaz basada en frameworks avanzados. Para lograr este propósito, se ha seguido una metodología ágil y desarrollado dos microservicios, ‘envahClients’ y ‘envahCollection’, cada uno con sus respectivas dependencias y roles claramente definidos. Así como la creación de una interfaz de usuario altamente intuitiva, eficiente y escalable basada en frameworks.

Palabras Clave

Frameworks, Interfaz avanzada, Aplicación web, Modernización de interfaz de usuario, Ingeniería.

Abstract

This paper presents the study carried out on the modernization of the user interface of the Evah application. With this in mind, this project has been developed with the idea of improving efficiency and user experience by implementing an interface based on advanced frameworks. To achieve this purpose, we followed an agile methodology and developed two microservices, 'envahClients' and 'envahCollection', each with their respective dependencies and roles clearly defined. As well as the creation of a highly intuitive, efficient, and scalable user interface based on frameworks.

Key words

Frameworks, Advanced interface, Web application, User interface modernization, Engineering.

1. Introducción

En un contexto donde se otorga una alta prioridad a la sostenibilidad y la eficiencia urbana, la gestión de residuos se convierte en un desafío de importancia crucial. Esto ha impulsado la creación de proyectos urbanos innovadores, como la implementación de sistemas de recolección neumática selectiva de desechos.

En Europa, fundamentalmente en España e Italia, prolifera progresivamente la implantación de sistemas que evitan el impacto visual de los residuos en el entorno urbano mediante soterramiento de contenedores con buzón exterior de vertido. Sucesivamente van dotándose de perfeccionamientos técnicos dirigidos a mejorar las condiciones del almacenamiento temporal previo a su recogida -estanqueidad, supresión de olores, operaciones de mantenimiento de equipos, etc…-. Al margen de los beneficios que pudieran representar, no atajan el problema de la retirada de los residuos de las zonas urbanas, sino que incluso encarecen y ralentizan el proceso de evacuación [1].

Con el paso del tiempo, la empresa Envac, pionera en el ámbito de la Recogida Neumática de residuos, ha experimentado una transformación significativa, abandonando la dependencia del papel y adoptando una presencia en línea mediante la creación de su sitio web. Permitiendo gestionar y supervisar sus operaciones de recolección de residuos. Pese a estos grandes cambios, en el pasado, las tecnologías utilizadas para el desarrollo web eran muy básicas, como HTML, CSS y JavaScript, las cuales aún se utilizan en la actualidad. Sin embargo, han surgido nuevas tecnologías mucho más eficientes y con funcionalidades avanzadas que ofrecen ventajas significativas en términos de mantenimiento y gestión.

La gestión y mantenimiento de una empresa se vuelven cada vez más desafiantes si no se dispone de herramientas que permitan aprovechar la información para tomar decisiones agiles y mantener el ritmo de la evolución tecnológica. Como es el caso del uso de Frameworks y Componentes que han demostrado ser eficaz en la mejora de la productividad y calidad del software, sobre todo si se combina con enfoques de Líneas de Productos [2]. A pesar de que muchas empresas aún no han explorado plenamente las amplias ventajas que estos marcos de trabajo pueden ofrecer cuando se implementan en sus proyectos, y algunas de las que están al tanto pueden mostrar cierta resistencia a la transición, quizás sea necesario proporcionales una demostración para destacar los notables beneficios que conlleva su utilización, con el fin de persuadirlas. Aunque los frameworks son uno de los artefactos de software que alcanzan mayores tasas de reutilización, cuyo desarrollo ha sido ampliamente estudiado [3].

Dada estas condiciones, el objetivo de este proyecto es desarrollar una aplicación web, adoptando una interfaz de usuario moderna y altamente interactiva mediante el uso de tecnologías de frameworks avanzados. Estas tecnologías permitirán gestionar de manera más eficiente los datos relacionados con el negocio, mejorando la experiencia de los usuarios finales y permitiendo un acceso intuitivo a la información crítica. Lo que llevaría a obtener mejores resultados en general.

Adicionalmente se llevará a cabo la creación de un backend sólido, utilizando una estructura escalable basada en microservicios. Implicando la separación de los distintos componentes de la aplicación en pequeños servicios web autónomos e independientes.

2. Objetivo

Desarrollar una aplicación web con la temática de la empresa Envac, adoptando una interfaz de usuario moderna y altamente interactiva mediante el uso de tecnologías de frameworks avanzados, mostrando las posibilidades de interfaz con distintas combinaciones de apariencia de los listados sobre los datos relacionados con el negocio.

2.1 Objetivos específicos

Para lograr el objetivo principal del proyecto, se presentan acciones específicas que definan cumplir las actividades.

- Identificar y analizar las necesidades y expectativas de los usuarios finales de la aplicación web.
- Evaluar y seleccionar las tecnologías y frameworks avanzados más adecuados para el desarrollo de la aplicación web, teniendo en cuenta la modernidad y la interactividad.
- Implementar las funcionalidades especificas requeridas para mostrar datos relacionados con el negocio, como la gestión de listados de datos.

3. Estado del arte

Desde los años 50, cuando se centraba en la reducción del tamaño del hardware, el software ha experimentado un crecimiento constante. La ingeniería de software ha evolucionado a un ritmo impresionante y se ha vuelto extremamente sofisticado en poco tiempo. Actualmente, satisfacer las demandas de las corporaciones representa un desafío para mejorar las arquitecturas tradicionales (como las monolíticas), donde se implementa la solución completa en una base única de código.

Figura 1: Arquitectura de software monolítica. Despliegue y ejecución sencillos, fácil desarrollo y bajo costes.

Esta arquitectura plantea desafíos en la implementación de nuevas ideas, ya que, en las arquitecturas monolíticas, un solo servicio se encuentra estrechamente vinculado a todos los procesos, lo que incrementa el riesgo de la aplicación. Fundamentada en seguir el patrón de diseño de software MVC (Modelo-Vista-Controlador) comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización [3]. Hoy día, es uno de los patrones utilizados para aplicaciones modernas permitiendo que estas sean escalables, mantenibles y fáciles de expandir. Este enfoque contribuye significativamente a la separación de los componentes de código del frontend y el backend.

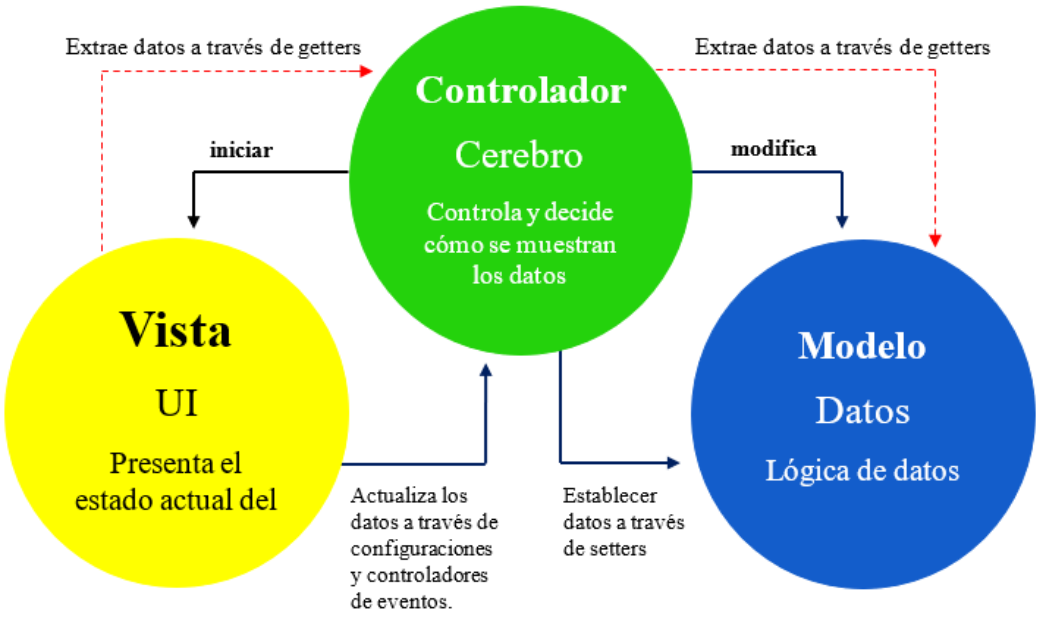


Figura 2: Gráfica de patrones de arquitecturas MVC

La disponibilidad de tecnologías de virtualización a partir de los años 2000 permitió la creación de espacios de almacenamiento a bajo costo y dio lugar al surgimiento de enfoques alternativos en la arquitectura de software, como la Arquitectura Orientada a Servicios (SAO). La principal finalidad de la SOA es construir sistemas informáticos empresariales de manera integrada, utilizando conjunto de estándares informáticos. En resumen, busca descomponer una aplicación en múltiples componentes basados en servicios independientes, lo que permite la reutilización de estos servicios con un bajo acoplamiento y el uso de interfaces estándar.

Por último, hasta este punto, es esencial resaltar la relevancia obtenida de una interfaz que se emplea para conectar varios sistemas basados en el protocolo HTTP, conocida como REST (REpresentional State Transfert). REST surge como una solución para abordar la complejidad que implicaba el Protocolo Simple de Acceso a Objetos (SOAP), ofreciendo un formato más liviano y legible, el JSON (JavaScript Object Notation). Hoy día, el más utilizado.

Esta evolución constante del software y la creciente necesidad de desarrollar aplicaciones con mayor agilidad, escalabilidad, flexibilidad, autonomía y la capacidad de implementar tecnologías y código reutilizable ha llevado a la adopción de la arquitectura de microservicios.

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de estos servicios es independiente y debe implementar una

funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio [4].

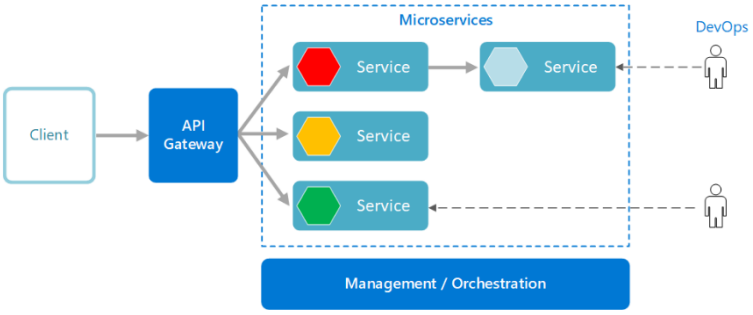


Figura 3: Gráfica de estilo de arquitectura de microservicios.

Los microservicios son un enfoque para la arquitectura de desarrollo de software. El prefijo "micro" en microservicios hace referencia al código que se entrega en partes o componentes pequeños y administrables, y cada "servicio" o función principal se crea e implementa independientemente de los demás servicios. Los componentes independientes funcionan juntos y se comunican a través de documentos de API prescritas denominados contratos [4].

Adoptar una arquitectura de microservicios, al igual que cualquier cambio, requiere comprender y validar ciertas características que no todas las empresas están preparadas para abordar. A pesar de que una de las ventajas fundamentales de los microservicios es simplificar, agilizar y escalabilizar las operaciones, la gestión y el mantenimiento de esta estructura en grandes empresas pueden convertirse en un desafío si no se manejan adecuadamente desde el principio.

Con todo este escenario tecnológico, la modernización y desarrollo de aplicaciones web se ha vuelto menos complicado. En este entorno, han surgido diversas tecnologías y frameworks que facilitan considerablemente el trabajo de los desarrolladores, permitiendo alcanzar estos objetivos con mayor eficiencia.

Un framework no es más que un esquema o marco de trabajo que ofrece una estructura o conjunto de conceptos base para elaborar un proyecto con objetivos específicos. Como es el caso de Sprint, el marco de trabajo más emblemático para desarrollar aplicaciones Java empresarial utilizando POJO. El cual cuenta con una serie de 23 proyectos que lo componen como Sprint Framework, Sprint Boot, Sprint Data, Sprint Cloud, entre otros…

Figura 4: Imagen de los proyectos que componen Sprint

Otro gran término utilizado al momento de desarrollar una aplicación web, es el de librería. El concepto de "librería" es más antiguo que el de framework. Las librerías consisten en una colección de clases o métodos que proveen comportamiento a otra aplicación. También se diferencian de los frameworks en que no se especializan en un control de flujo de datos interno, uso de herencia y patrones de diseño [5].

4. Desarrollo

4.1 Contextualización

En la sección anterior, hemos repasado la evolución que ha ocurrido desde los primeros conceptos de ingeniería de software, abarcando desde las arquitecturas hasta el patrón de diseño más ampliamente utilizado en la actualidad, el MVC. Además, hemos explorado como la virtualización ha planteado nuevos desafíos y necesidades, conduciendo finalmente a la adopción de una arquitectura basada en microservicios. Esta arquitectura ha revolucionado el desarrollo de aplicaciones web y ha dado lugar a la aparición de nuevos frameworks y bibliotecas que permiten abordar estas necesidades de manera más ágil.

Cumpliendo el objetivo principal de este Trabajo Fin de Máster (TFM) sobre proporcionar una interfaz de usuario moderna y altamente interactiva mediante el uso de tecnologías de frameworks avanzados, se abordan aspectos cruciales de desarrollo de software web, desde la arquitectura hasta la implementación, con un enfoque particular en la eficiencia y la escalabilidad.

En la parte de backend, se ha empleado el lenguaje de programación Java junto con el poderoso framework Spring, utilizando el patrón Modelo Vista Controlador (MVC). Esta elección permite la creación de una sólida base de datos MySQL que sustenta la estructura de microservicios, esencial para la gestión eficiente de datos relacionados con clientes, contenedores, órdenes, solicitudes y otros aspectos fundamentales para la recogida de residuos.

La parte frontal de la aplicación se ha desarrollado utilizando React.js en combinación con TypeScript, React Router, Axios, Formik y Material UI, entre otras tecnologías. Esta elección proporciona las posibilidades de interfaz con distintas combinaciones de apariencia de los listados sobre los datos relacionados con el negocio.

En las siguientes secciones, exploraremos en detalle cada aspecto de este proyecto, desde su diseño y desarrollo hasta sus conclusiones y perspectivas futuras.

4.2 Metodología

Para cubrir el avance de una manera ágil y eficaz, asegurando pequeñas entregas graduales que contemplen pequeñas partes funcionales del producto y obtener retroalimentación más rápido, se ha contemplado el uso de las metodologías agiles.

Minimizar la complejidad de la aparición de posibles incidencias y el intento por agilizar la distribución del código bajo un mismo contexto, se aplica el uso de la herramienta adaptable y flexible para la planificación y el seguimiento del trabajo en GitHub llamada “GitHub Projects”. Que según su página oficial menciona, los proyectos se crean a partir de las incidencias y las solicitudes de incorporación de cambios que agregas, lo que crea referencias directas entre el proyecto y el trabajo. La información se sincroniza automáticamente con el proyecto a medida que realizas cambios, y se actualizan las vistas y los gráficos [6].

Aquí, al igual que en otras plataformas, puede definirse las historias de usuarios, asignar tareas, hacer seguimiento del trabajo y gestionar incidencias, lo que te permite tener un control total sobre el ciclo de vida del proyecto.

Para asegurar una gestión eficaz y un desarrollo efectivo en este proyecto, se optó por dividirlo en cuatro (4) sprints de dos semanas claramente definidos. Cada uno de estos sprints permitió centralizar las tareas específicas y avanzar gradualmente hacia los objetivos. A continuación, proporcionamos las fechas de inicio y finalización de cada sprint.

- Sprint 01 | 24 de Julio al 04 de Agosto, 2023.
- Sprint 02 | 07 de Agosto al 18 de Agosto, 2023.
- Sprint 03 | 21 de Agosto al 01 de Septiembre, 2023.
- Sprint 04 | 04 de Septiembre al 15 de Septiembre, 2023.

Se llevaron a cabo dos (2) ceremonias de aproximadamente una hora. La reunión de planificación del sprint con el objetivo de planificar que se deberá finalizar para el siguiente sprint y la reunión de revisión del sprint para recibir los comentarios de las principales partes interesadas del proyecto.

Se implementaron diversas y definidas historias de usuario específicas. Cada historia de usuario incluye una descripción exhaustiva que proporcionaba contexto sobre su propósito y requisitos precisos. De esta manera, se lograba una visión clara para cualquier persona encargada de llevar a cabo las tareas relacionadas con estas historias.

Estas descripciones detalladas de las historias de usuario y las tareas también garantizaban una comprensión precisa y consistente de los requisitos en todo momento.

Figura 5: GitHub Project, backlog del proyecto

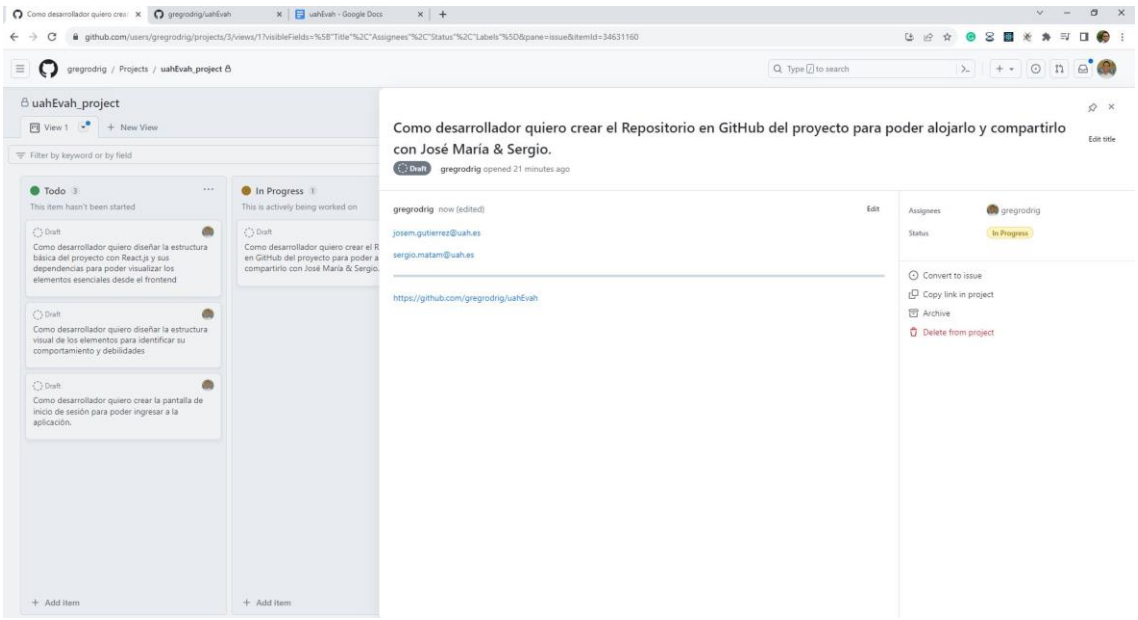


Figura 6: Creación de requerimientos del proyecto en la herramienta GitHub Project.

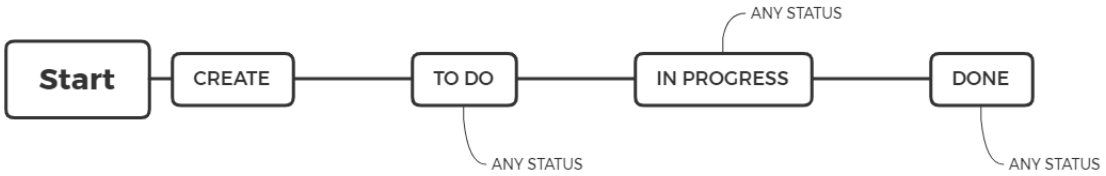


Figura 7: GitHub Project - Workflow de tareas.

Para validar el flujo de trabajo del proyecto, fueron asumidos los estados de un flujo básico de estados que pudieran mantener la viabilidad el progreso.

TO DO: En esta columna se encuentran las tareas que se han registrado o identificado, pero que todavía no han sido asignadas ni iniciadas por ningún miembro del equipo. Aquí se listan todas las tareas pendientes.

IN PROGRESS: Esta columna proporciona detalles sobre las responsabilidades asignadas a un miembro del equipo y si estado de avance. Aquí se registraba el progreso de las actividades en curso.

DONE: Las tareas eran trasladadas a esta columna una vez que habían superado todas las pruebas con éxito y se consideraba que cumplían con los criterios de aceptación. En esta columna, se registraban todas las tareas que habían sido finalizadas y estaban listas para ser entregadas o completadas, dependiendo de la situación.

4.3 Diseño de la Arquitectura de la Aplicación

En el desarrollo de la aplicación EVAH, la elección de una arquitectura de software adecuada es fundamental para garantizar la eficiencia. Dicha arquitectura está diseñada para proporcionar una experiencia de usuario eficiente y coherente, cumpliendo con objetivos solidos para seguir las mejores prácticas de garantizar la escalabilidad, mantenibilidad, desarrollo ágil, resiliencia, facilidad de adopción de nuevas tecnologías, reutilización, facilidad de colaboración e implementación de DevOps, intentando contemplar una mejora sobre la seguridad y la escalabilidad de los datos.

Se adhiere la elección de implementar una arquitectura de microservicios en el proyecto. Con esta estructura, el sistema se divide en componentes independientes y autónomos que se pueden desarrollar, implementar y escalar de forma independiente. Facilitando la adaptación a cambios y permite el despliegue continuo.

Para desarrollar una aplicación de manera eficiente, aprovechamos tecnologías de backend, como el lenguaje de programación Java en combinación con Spring Boot. Spring Boot es una parte esencial del ecosistema Spring, diseñada específicamente para simplificar la gestión de dependencias y la creación de proyectos. Esto nos permite enfocarnos en lo más importante: el desarrollo de la aplicación en sí.

Para comprender mejor el contexto, Spring Boot Starter es la herramienta intuitiva que utilizamos para seleccionar el tipo de aplicación que deseamos crear y las dependencias asociadas a ella. De esta manera, podemos comenzar con una base sólida y construir sobre ella.

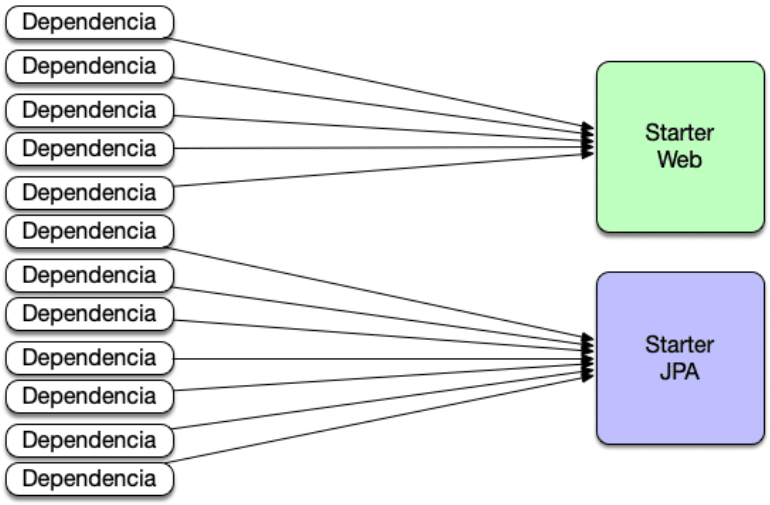


Figura 8: Diagrama de dependencias con Spring Boot Starter.

Además de las tecnólogas backend, se incorpora el sistema de base de datos relacional MySQL. Por la parte frontal, confiamos en la librería Reac.js junto con TypeScript, un superconjunto de JavaScript que agrega tipado estático al lenguaje, lo que proporciona una aplicación más robusta y resistente a errores.

Como los navegadores no entienden TypeScript, es necesario compilarlo (la palabra correcta es 'transpilarlo') a JavaScript antes de usarlo en un navegador. TypeScript se puede transpirlar a cualquier variante de JavaScript/ECMAScript que deseemos. Por ejemplo, si sabemos que nuestro proyecto va a trabajar en navegadores antiguos podemos forzar que se compile a ECMAScript 5 (el JavaScript clásico, soportado por todos los navegadores, incluso Internet Explorer). Sin embargo, si tenemos claro que los usuarios van a emplear navegadores relativamente recientes, podemos generar un código final más sencillo apoyado en ECMAScript 6 o posterior [7].

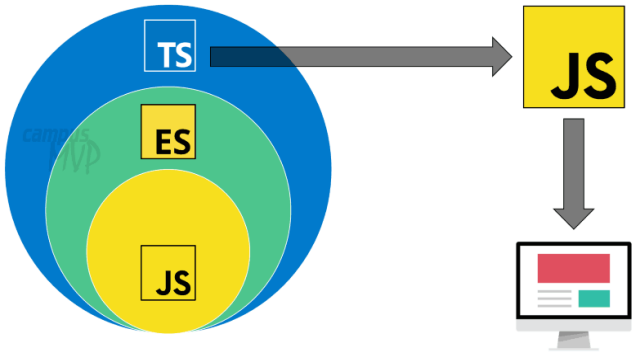


Figura 9: Esquema ilustrativo de TypeScript

TypeScript se convierte en un sólido validador que facilita la identificación de lo que se espera y lo que no espera de cada componente creado en el proyecto. En este caso, desempeñará un papel crucial al verificar que los datos consumidos desde nuestros microservicios y bases de datos cumplan con los estándares establecidos.

Por último, para mejorar la experiencia de usuario, hacemos uso de Material UI (MUI) que tiene un hermoso diseño y presenta un conjunto de opciones de personalización que facilitan la implementación de su propio sistema de diseño personalizado sobre sus componentes [8].

Para añadir una mayor experiencia y rendimiento significativo en la parte de los formularios, se hace uso de formik, una pequeña librería para cubrir 3 partes importantes. Obtener valores dentro y fuera del estado, mensajes de validación y errores, y el manejo y envíos de los formularios.

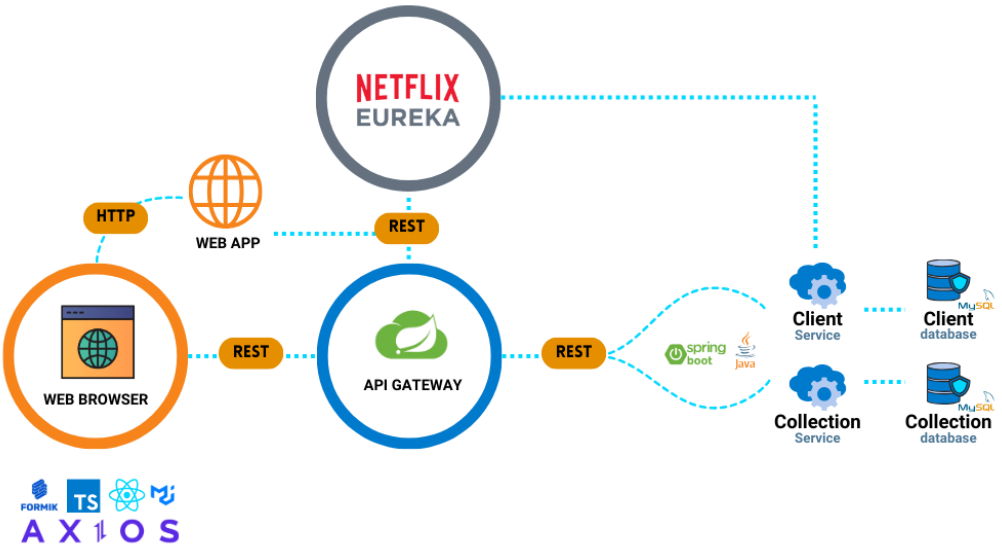


Figura 10: Diseño de arquitectura del software.

4.3.1 Arquitectura y Diseño de la Base de Datos

En la búsqueda de cumplir los objetivos principales sobre la flexibilidad y escalabilidad de la aplicación, optar por MySQL como sistema de gestión de base de datos para resguardar grandes cantidades de datos del proyecto es de vital importancia. Ya que, al ampliar el desarrollo de nuevas funcionalidades, esta nos permite escalar fácilmente.

Para cubrir una estructura distribuida basada en microservicios, se ha creado una base de datos con sus respectivas tablas para cada uno de los servicios.

Figura 11: Base de Datos con MySQL, diseño de tablas para el servicio "Client".

Figura 12: Base de Datos con MySQL, diseño de tablas para el servicio "Collection".

4.3.1.1 Servicio Client

Nombrada envahdb_clients_containers_orders_requests. Contine cuatros (4) tablas definidas, las cuales almacenan los datos sobre los clientes, requerimientos, órdenes y contenedores.

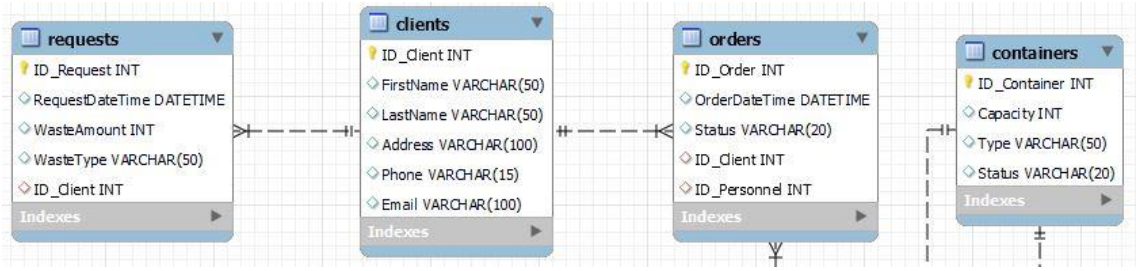


Figura 13: Base de Datos con MySQL envahdb_clients_containers_orders_requests.

4.3.1.2 Servicio Collection

Nombrada envahdb_collection_routes. Contine seis (6) tablas definidas, las cuales almacenan los datos sobre el personal, las rutas, incidencias, colecciones, reportes y mantenimiento de los contenedores.

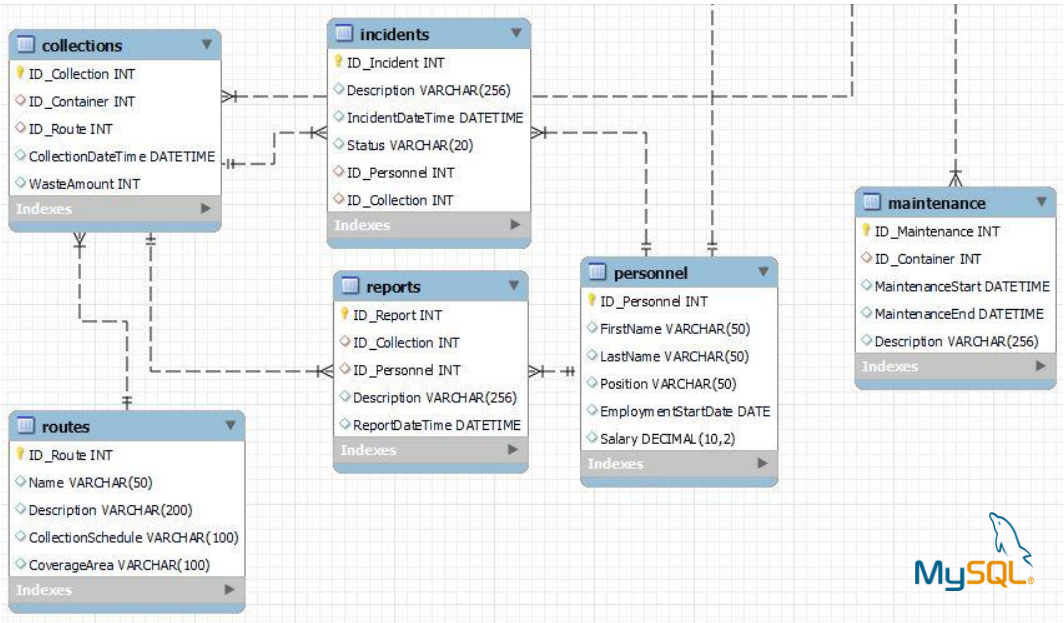


Figura 14: Base de Datos con MySQL envahdb_collection_routes.

En la siguiente imagen contemplaremos una vista general sobre la base de datos creada.

4.3.1.3 Vista General

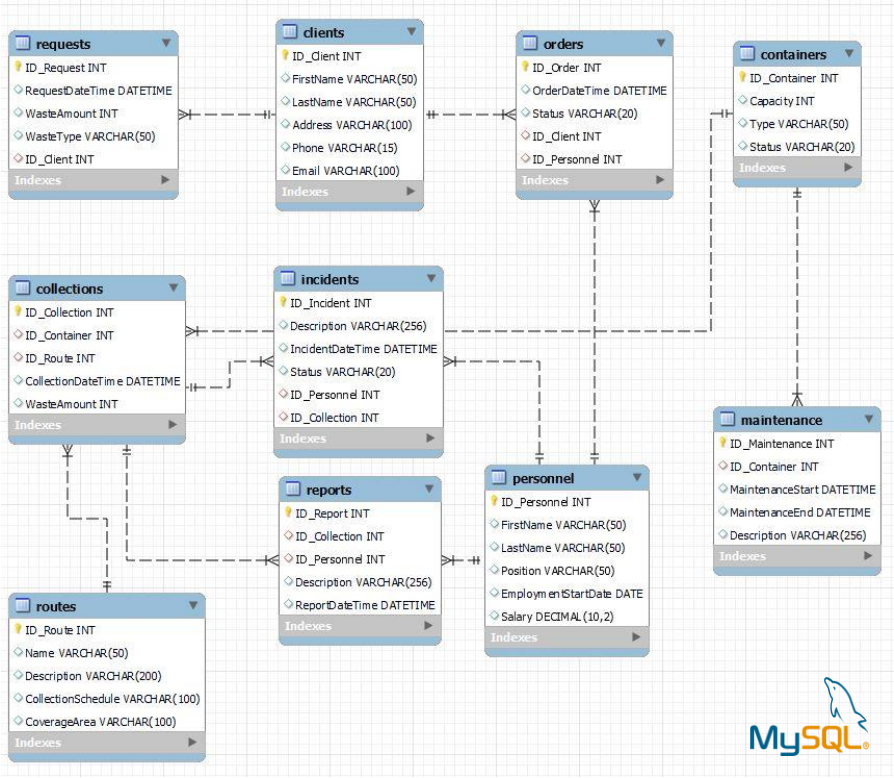
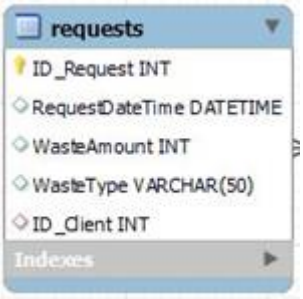
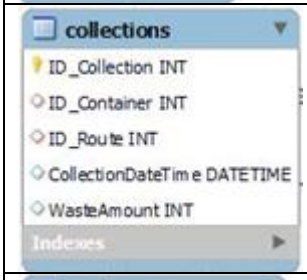
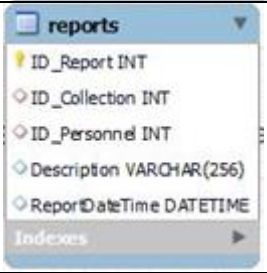


Figura 15: Diseño de arquitectura de la Base de Datos.

4.3.1.4 Vistas especificas

A continuación, se presentan las distintas tablas con las que cuenta el prototipo desacollarado.

Estructura Tabla	Descripción tabla
	**Requests**: Almacena los requerimientos, especificaciones y/o condiciones necesarias que deben cumplirse para que el sistema funcione efectivamente, realizados por parte de los clientes. Contiene el ID del requerimiento, la fecha, desechos o residuos, el tipo de residuo y el ID del cliente.

	**Clients**: Almacena los clientes (personas, empresas u organizaciones) del negocio. Contiene el ID del cliente, nombres y apellidos, dirección, teléfono y correo electrónico.
	**Orders**: Almacena las órdenes realizadas por parte de los clientes y el personal. Contiene el ID de la orden, fecha, estado, el ID del cliente y el ID del personal.
	**Containers**: Almacena los contenedores pertenecientes al negocio. Contiene el ID del contenedor, capacidad, el tipo, y el estado en que se encuentra.
	**Collections**: Almacenan las colecciones programadas para recoger los residuos. Contiene el ID de la colección, el ID del contenedor, el ID de la ruta, la fecha de la colección y la cantidad de residuo.
	**Incidents**: Almacena las incidencias, situaciones o problemas que afectan el funcionamiento normal y eficiente del sistema. Contiene el ID de la incidencia, la descripción, fecha de la incidencia, estado, el ID del personal, y el ID de la colección.
	**Reports**: Almacena los reportes, registros o documentación que se utilizan para rastrear y gestionar la información relaciona con la recogida de residuos. Contiene el ID del reporte, el ID de la colección, ID del personal, descripción del reporte y la fecha de creación.

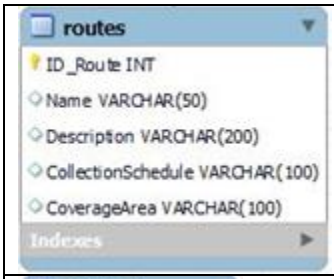
	**Routes**: Almacena las rutas, itinerarios o trayectos predefinidos que siguen los vehículos de recolección de basura para recoger los desechos. Contiene el ID de la ruta, nombre de la ruta, descripción, la fecha o calendario de recogida y el área de cobertura.
	**Personnel**: Almacena el personal o personas que trabajan en las diversas funciones del negocio. Contiene el ID del personal, nombres y apellidos, posición, fecha de entrada en la empresa y el salario que devenga.
	**Maintenance**: Almacena las actividades planificadas y acciones realizadas para garantizar que los equipos, vehículos, contenedores y otros componentes del sistema funcionen de manera eficiente y segura durante su vida útil. Contiene el ID del mantenimiento, ID del contenedor, fecha de inicio y fin del mantenimiento y una descripción.

Tabla 1: Representación del contenido de las tablas de la base de datos.

4.3.2 Arquitectura de Microservicios

Dentro de los objetivos de este proyecto, se ha optado por la implementación de Java Spring Boot, una herramienta de código abierto recomendada por Microsoft en su página oficial. Spring Boot facilita el desarrollo de microservicios y aplicaciones web basados en Java y proporciona una ruta de acceso más fácil y rápida para configurar, establecer y ejecutar aplicaciones. Eliminando el trabajo pesado de configuración necesario para configurar la mayoría de las aplicaciones basadas en Spring [9].

Spring Boot contiene preconfiguradas por defecto las bibliotecas más utilizadas para iniciar un proyecto funcional con comodidades para que los desarrolladores pueden ejercer rápidamente un despliegue.

Se han creado dos microservicios, nombrados “envahClients” y “envahCollection”. Incorporando importantes dependencias en cada uno de ellos, como:

- Spring Boot DevTools, permite reiniciar de manera automática nuestra aplicación en cada cambio ejercido en el código,
- Spring Web, permite crear aplicaciones bajo el patrón MVC y servicios web tipo RESTfull,
- Spring Data JPA, persiste los datos de la aplicación en una BBDD usando JPA,
- MySQL Driver, permite utilizar el controlador JDBC para gestionar la BBDD,
- Spring Cloud Netflix Eureka, servidor encardado de registrar los microservicios para facilitar su descubrimiento y utilización.

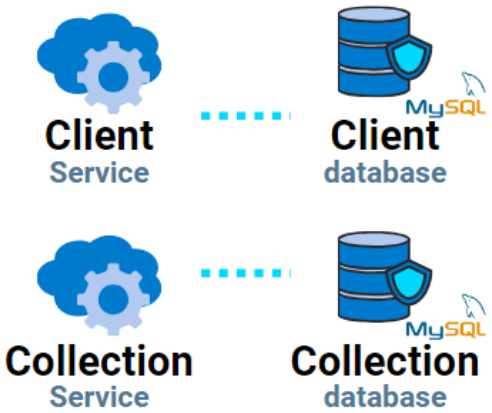


Figura 16: Microservicios creados en el proyecto junto a base de datos

4.3.2.1 Creación de Microservicios

4.3.2.1.1 Spring Initializr

Con Spring Initializr, una herramienta perteneciente al marco Spring, utilizada para la generación de proyectos donde nos otorga un fichero .rar con las dependencias añadidas de elección.

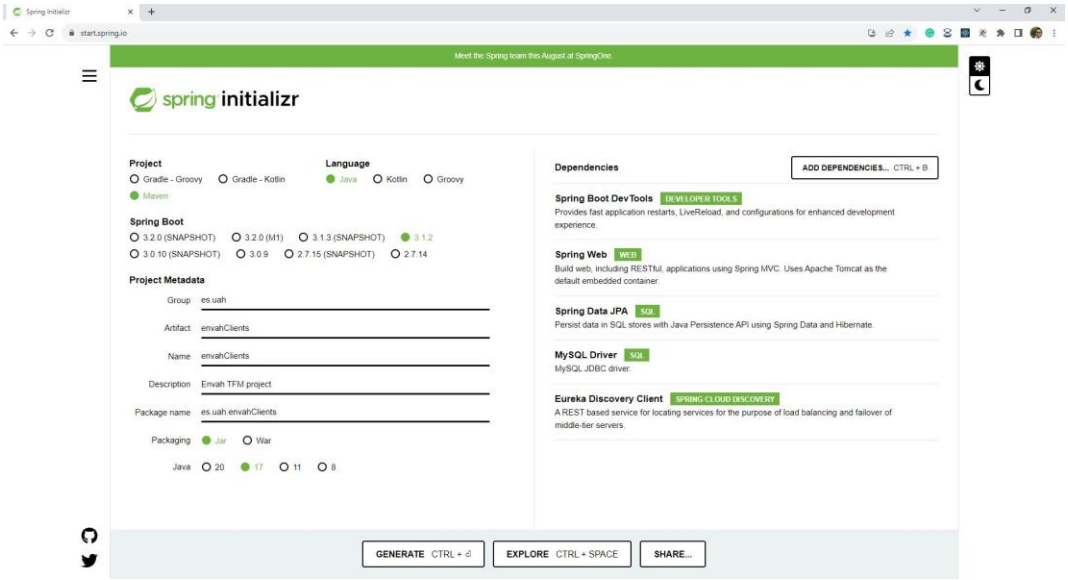


Figura 17: Inicialización de la creación del proyecto con Spring Initializr y dependencias

Una vez obtenido el fichero .rar con Spring Initializr, es cuando puede abrirse el proyecto desde el IDE por elección. Donde el archivo aplication.properties de ambos microservicios deben quedar con la siguiente forma, validando que cada uno persistan los puertos y accesos adecuados:

```
# DATASOURCE (MYSQL 8.0)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/${MYSQ
L_DATABASE_NAME:envahdb_clients_containers_orders_requests}?useSSL=fal
se&serverTimezone=Europe/Madrid&allowPublicKeyRetrieval=true
spring.datasource.username=${MYSQL_USER:root}
spring.datasource.password=${MYSQL_PASSWORD:Password}
#JPA
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDia
lect
spring.jpa.show-sql=true
```

```
# Table names physically
spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandar
dImpl
#Name and Port
server.port=8000
spring.application.name=client-service
eureka.client.service-
url.defaultZone=${EUREKA_HOST:http://localhost:8761}/eureka
#DevTools
#spring.devtools.add-properties=false
```

4.3.2.1.2 Patrón de arquitectura MVC

Al aplicar de manera efectiva el patrón de arquitectura MVC (Modelo-Vista-Controlador) se obtiene una organización más clara y eficiente del código.

4.3.2.1.2.1 Modelo (Model)

En esta parte de la arquitectura, se crea una estructura de carpetas dedicada al "Modelo". Aquí es donde se almacenan los objetos y clases de entidad de la aplicación. Utilizando clases tipo POJO en Java, que contienen atributos, constructores y métodos get y set para representar los datos. Además, se aplica el patrón "DAO" (Data Access Object) para encapsular toda la lógica de acceso a datos en la aplicación. Esto garantiza un acceso consistente y seguro a la capa de datos.

```
package es.uah.envahClients.model;
import jakarta.persistence.*;
import java.util.Objects;

@Entity
@Table(name = "clients")
public class Client {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "ID_Client")
    private Integer idClient;
    @Basic
    @Column(name = "FirstName")
    private String firstName;

    public Integer getIdClient() {
        return idClient;
    }
    public void setIdClient(Integer idClient) {
        this.idClient = idClient;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
```

```
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Client)) return false;
//      if (o == null || getClass() != o.getClass()) return false;
        Client client = (Client) o;

        if (!Objects.equals(idClient, client.idClient)) return false;
        if (!Objects.equals(firstName, client.firstName)) return
false;
        return true;
    }
    @Override
    public int hashCode() {
        int result = idClient != null ? idClient.hashCode() : 0;
        result = 31 * result + (firstName != null ?
firstName.hashCode() : 0);
        return result;
    }
}
```

4.3.2.1.2.2 Controladores (Controller):

En esta etapa, se crea una carpeta específica para implementar la lógica de negocio y servir como un enlace entre la capa de datos (DAO) y la capa de presentación. Aquí es donde implementar los controladores, que son responsables de gestionar las operaciones RESTful necesarias para interactuar con las clases previamente creadas en el servicio. Los controladores actúan como intermediarios entre el modelo y la vista, garantizando una separación clara de las responsabilidades.

```
package es.uah.envahClients.controller;
import es.uah.envahClients.model.Client;
import es.uah.envahClients.service.IClientsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("clients")
@CrossOrigin
public class ClientsController {

    @Autowired
    IClientsService clientsService;

    @GetMapping("")
    public List<Client> findAll() {
        return clientsService.allClients();
    }
    @GetMapping("/{idClient}")
    public Client findById(@PathVariable("idClient") Integer idClient)
```

```
{
        return clientsService.findById(idClient);
    }
    @GetMapping("/name/{name}")
    public List<Client> findByFirstName(@PathVariable("name") String
name) {
        return clientsService.findByFirstName(name);
    }
    @PostMapping(value = "", produces = MediaType.TEXT_PLAIN_VALUE)
    public String saveClient(@RequestBody Client client) {
        return String.valueOf(clientsService.saveClient(client));
    }
    @PutMapping("")
    public void updateClient(@RequestBody Client client) {
        clientsService.updateClient(client);
    }
    @DeleteMapping(value = "/{idClient}", produces =
MediaType.TEXT_PLAIN_VALUE)
    public String deleteClient(@PathVariable("idClient") Integer
idClient) {
        return String.valueOf(clientsService.deleteClient(idClient));
    }
}
```

Con esta organización, consta el logro de una aplicación más estructurada y modular, lo que facilita la mantenibilidad y escalabilidad del proyecto. El patrón MVC nos permite separar la lógica de presentación (Vista), la lógica de negocio (Modelo) y la lógica de control (Controlador), lo que resulta en un código más limpio y fácil de entender y mantener.

En la siguiente imagen, identificaremos como ha quedado estructurado el patrón MVC en el proyecto:

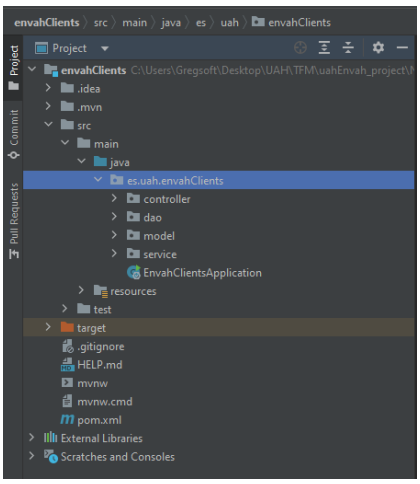


Figura 18: Estructura del proyecto aplicando el patrón de diseño MVC

4.3.2.1.3 Eureka Server

Sirviendo como gestor para mantener los registros y localización de los microservicios creados, y a la vez, brindar la posibilidad de consultar el estado y metadatos de configuración. Es un servidor administrador de direcciones de microservicios, de los cuales deben configurarse como un cliente de Euraka, solicitando una petición al momento de arrancar.

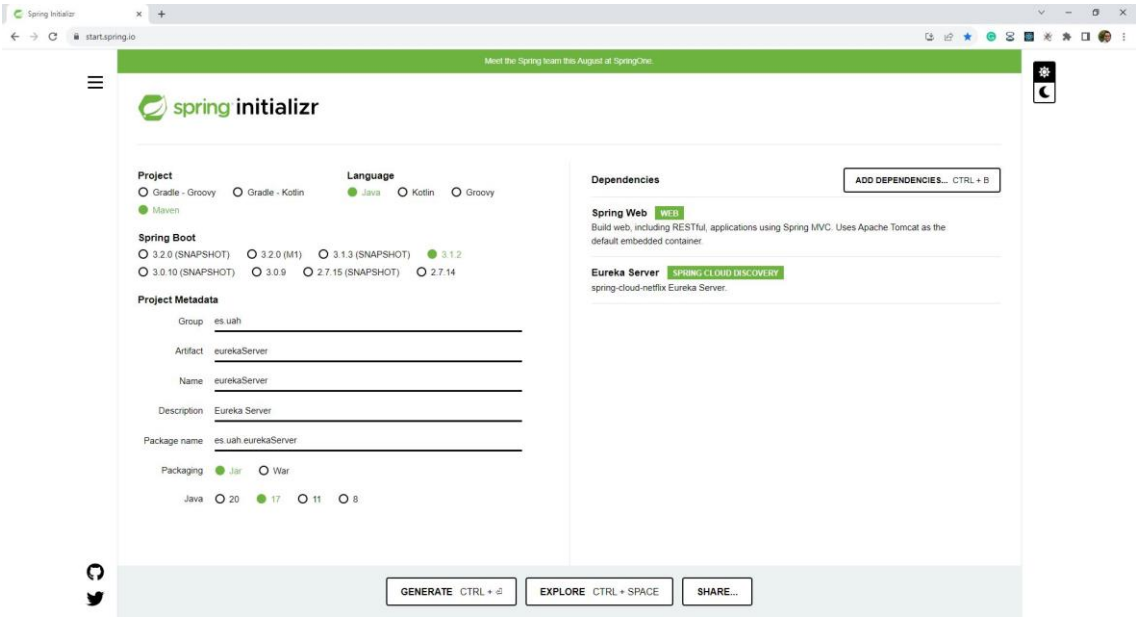


Figura 19: Inicialización de la creación del proyecto Eureka con Spring Initializr y dependencias

Quedando el archivo aplication.properties de esta forma:

```
spring.application.name=eureka-server
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.service-
url.defaultZone=${EUREKA_URI:http://localhost:8761/eureka}

spring.servlet.filter-name=CorsFilter
spring.servlet.filter.order=0
spring.servlet.filter.enabled=true
spring.servlet.filter.registration=CorsFilter
spring.servlet.filter.params.allowed-origins=*
spring.servlet.filter.params.allowed-methods=GET,POST,PUT,DELETE
spring.servlet.filter.params.allowed-headers=Content-Type,X-Requested-
With,accept,Origin,Access-Control-Request-Method,Access-Control-
Request-Headers
spring.servlet.filter.params.exposed-headers=Access-Control-Allow-
Origin,Access-Control-Allow-Credentials
spring.servlet.filter.params.supports-credentials=true
spring.servlet.filter.params.preflight-max-age=1800
```

4.3.2.1.4 API Gateway

Bajo el marco de Spring se contempla Spring Cloud Gateway, encargado de enrutar cualquier petición HTTP entrante hacia el servicio creado. Actuando como único punto de entrada a la aplicación.

Ejecutándose en el puerto 8090, el servidor API Gateway gestionará las peticiones que contengan el path /api/client, enrutándola hacia el servicio 'Client'. De igual manera, en el caso de /api/collection hacia el servicio 'Collection'.

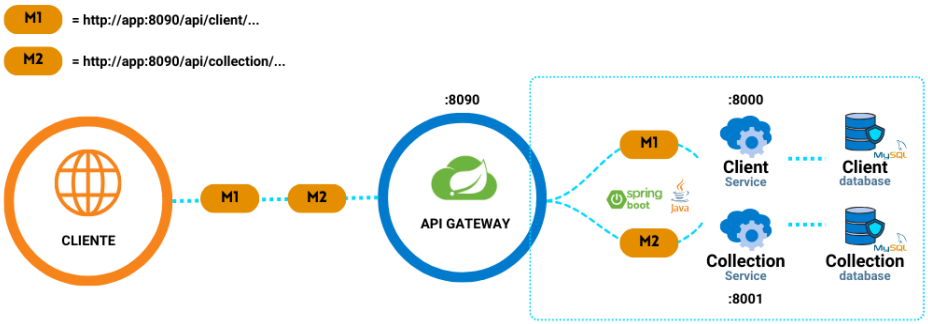


Figura 20: Diagrama de comunicación API Gateway

4.3.2.1.4.1 Ejecución

Una vez creados los microservicios, es de suma importancia constatar la ejecución de primera instancia de Eureka Server, para luego continuar con la API Gateway, finalizando con ambos microservicios 'envahClients' y 'envahCollection'.

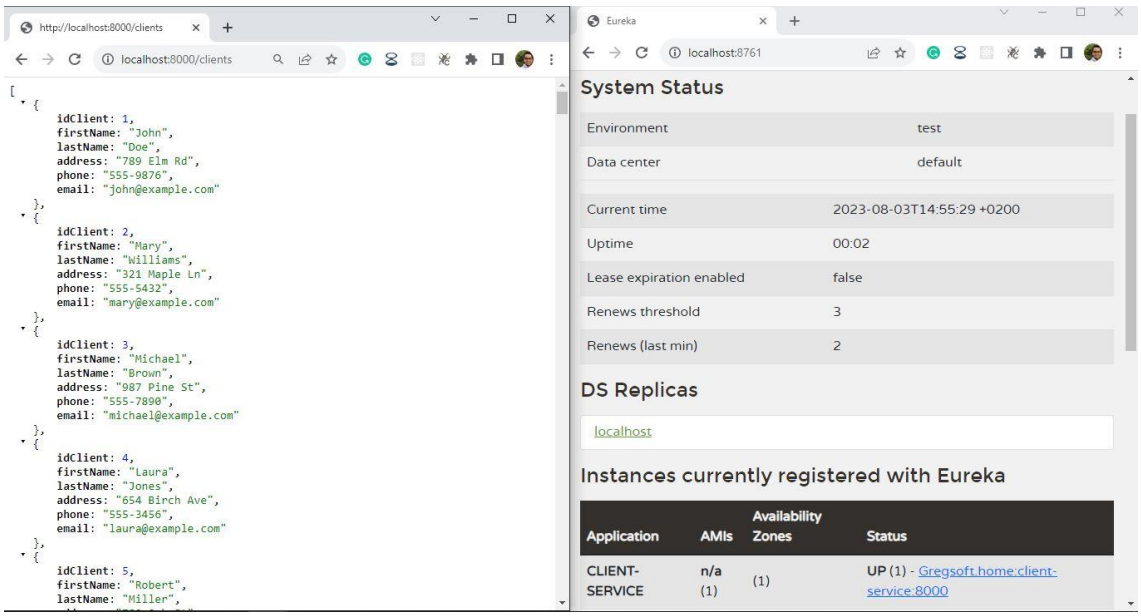


Figura 21: Vista sobre la ejecución de los microservicios, Eureka Server y Api Gateway

4.3.2.1.4.2 Pruebas API

Con el objetivo de identificar el correcto funcionamiento sobre las operaciones básicas de la aplicación y validar la gestión de la base de datos, hacemos uso de la herramienta Postman. Es una plataforma API para crear y utilizar API. Postman simplifica cada paso del ciclo de vida de la API y agiliza la colaboración [10]. Estas operaciones son a las que se les conoce como CRUD.

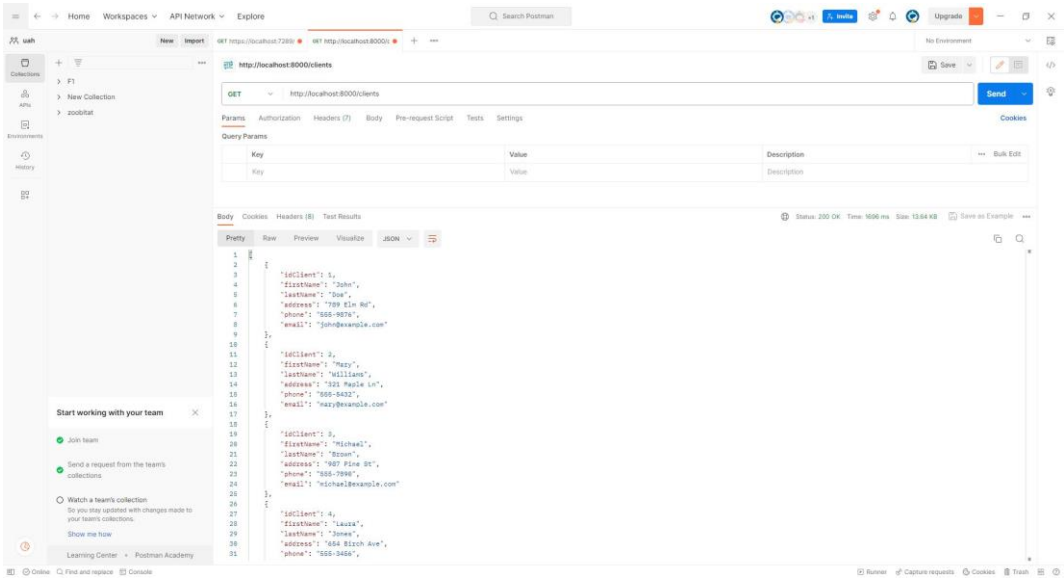


Figura 22: Utilizando Postman - Pruebas API.

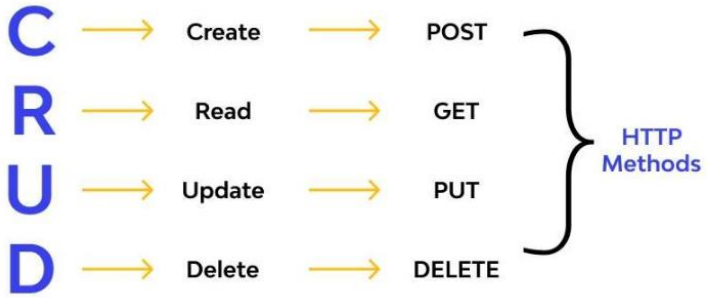


Figura 23: Imagen sobre Operaciones CRUD.

4.3.2.1.4.2.1 Read (Leer)

Utilizando la lógica de control (Controlador) desde el proyecto ‘envahClients’, se realiza la consulta bajo el método GET haciendo uso del path /app:8000/clients/lastName/wi para obtener la lista de clientes por apellidos que contengan las letras ‘wi’.

De este modo, la operación de lectura implica recuperar información de la base de datos para mostrarla, lo cual haciendo uso de postman se logra visualizar en la sección de Body (Cuerpo).

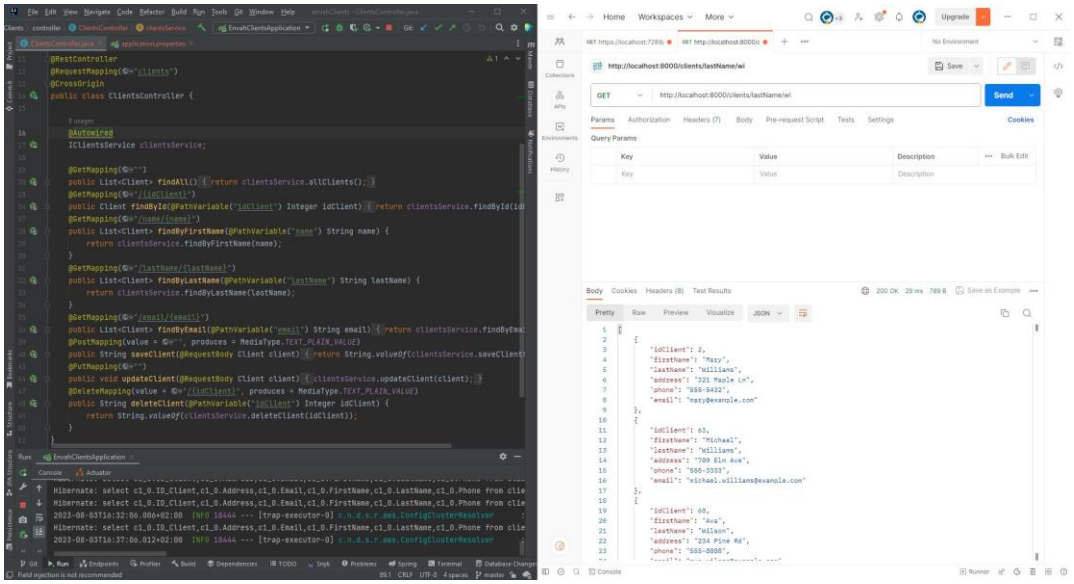


Figura 24: Operación de lectura (GET) consultando los clientes por apellidos, haciendo uso de la herramienta postman mediante el controlador 'clients'.

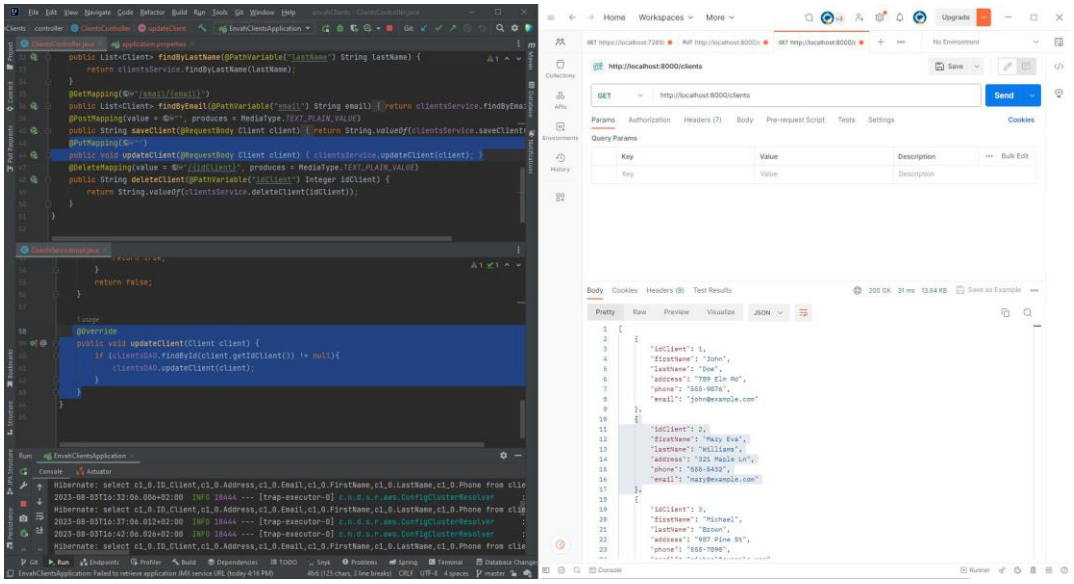


Figura 25: Operación de lectura (GET) consultando la lista de todos los clientes.

4.3.2.1.4.2.2 Create (Crear)

La importancia de crear o insertar nuevos registros en la aplicación, es de suma importancia en la arquitectura de cualquier proyecto web o sistema informático. Con la operación de Crear bajo el método (POST) se logra ingresar un nuevo cliente mediante el path /app:8000/clients. Si la operación es realizada correctamente, obtendríamos como respuesta el valor 'true', de lo contrario un espacio en blanco en la sección de Body (Cuerpo).

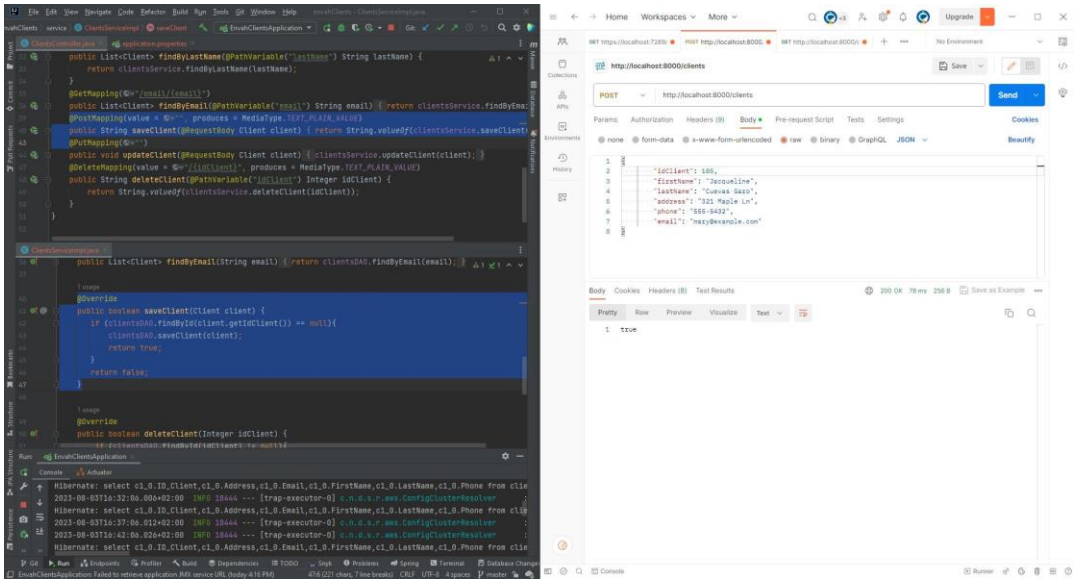


Figura 26: Operación de escritura (POST) ingresando un nuevo cliente, haciendo uso de la herramienta postman mediante el controlador 'clients'.

4.3.2.1.4.2.3 Update (Actualizar)

Una vez lograda la oportunidad de insertar datos a la aplicación, se abre la ventana de poder actualizarlos. Con la operación Actualizar bajo el método (PUT) es permitido modificar registros o entradas existentes en la base de datos. En este caso, mediante el path /app:8000/clients. Si la operación es realizada correctamente, obtendríamos como respuesta un espacio en blanco, de lo contrario un valor 'null' en la sección de Body (Cuerpo).

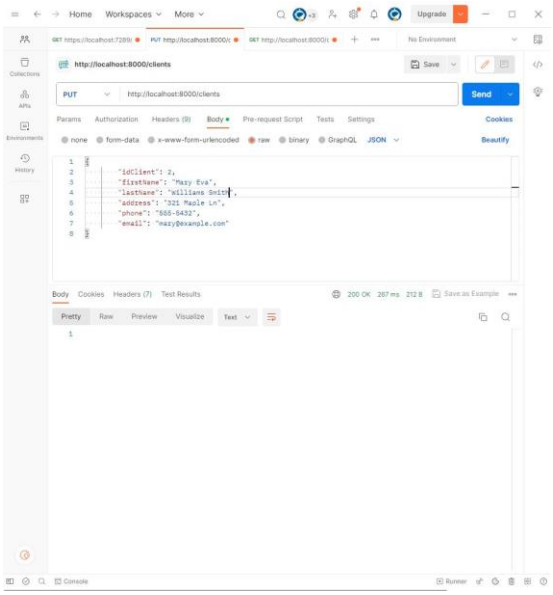


Figura 27: Operación (PUT) para modificar clientes existentes, haciendo uso de la herramienta postman mediante el controlador 'clients'.

4.3.2.1.4.2.4 Delete (Eliminar)

Con la operación eliminar bajo el método (DELETE) implica borrar el registro de la aplicación. Mediante el path /app:8000/clients/IdClient. Si la operación es realizada correctamente, obtendríamos como respuesta el valor ‘true’, de lo contrario un espacio en blanco en la sección de Body (Cuerpo).

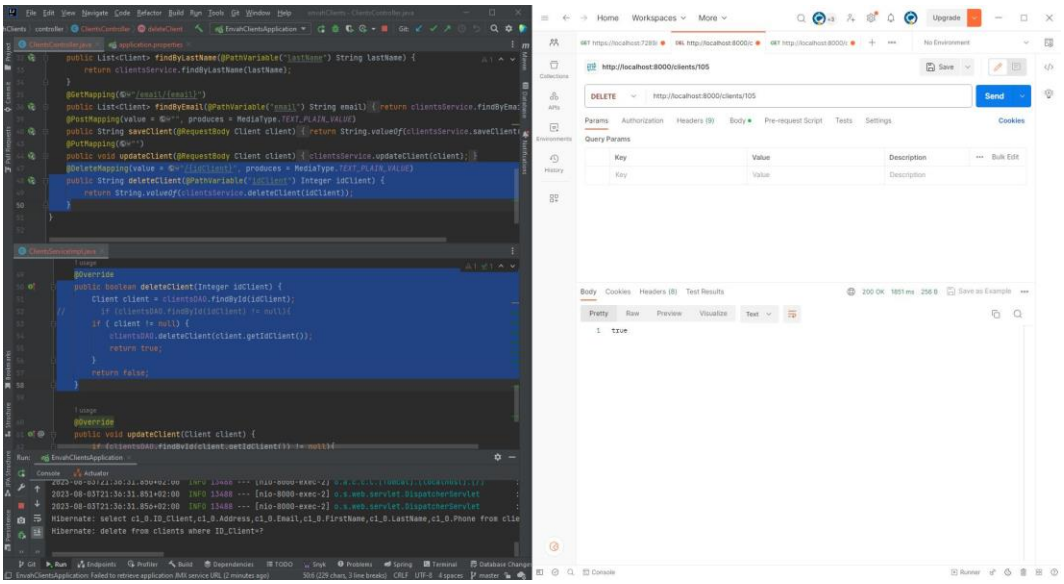


Figura 28: Operación (DELETE) borramos un cliente por su ID, haciendo uso de la herramienta postman mediante el controlador ‘clients’.

Estos métodos GET, UPDATE, POST y DELETE conformados en el acrónimo CRUD, han sido utilizados para todos los path construidos en ambos microservicios del proyecto. Estas operaciones fundamentales en cualquier aplicación web, permiten que los usuarios logren realizar las acciones básicas de gestión de datos.

Adoptando una interfaz de usuario moderna y altamente interactiva mediante el uso de tecnologías de frameworks avanzados. Se crearán a continuación, sendos formularios y funcionalidades específicas de filtrados desde la parte frontal (View), brindando una experiencia completa y funcional.

4.4 Desarrollo del Frontal (View)

A continuación, validaremos en detalle la construcción de la interfaz de usuario de la aplicación web. El desarrollo frontal, también conocido como desarrollo web.

4.4.1 ReactJS

En el desarrollo web, existe una potente e importante tecnología nombrada React.js. Una biblioteca que permite agrupar componentes, pero no prescribe cómo hacer el enrutamiento y la obtención de datos. Facilita construir interfaces de usuario a partir de piezas individuales llamadas componentes que son funciones de JavaScript. Los componentes de React.js reciben datos y devuelven lo que debe aparecer en la pantalla renderizando componentes interactivos. A su vez, permite construir tanto aplicaciones web como nativas utilizando las mismas habilidades [11].

Hoy día, cientos de proyectos han implementado esta librería de React.js, pudiendo mencionar, Airbnb, Discord, WhatsApp, Netflix, Instagram, entre otros más.

La implementación de esta herramienta en el proyecto ofrece la oportunidad de alcanzar los objetivos establecidos. Aprovechando al máximo el potencial de React.js, podemos lograr una interfaz de usuario dinámica, gestionar eficazmente la reactividad de los datos y mejorar el rendimiento de la aplicación, lo que se traduce en una experiencia más rápida y eficiente. Además, Reeact.js se basa en una arquitectura de componentes que fomenta la modularidad y facilita la reutilización de código.

4.4.1.1 Creando proyecto con ReactJS

Para crear un proyecto con React.js es importante dirigirse a su página oficial https://create-react-app.dev/ aunque actualmente ya no está siendo soportada y se sugiere visitar https://es.react.dev/ .

Luego de crear el proyecto, en el archivo principal ‘index.tsx’ se hace uso de ‘BrowserRouter’, un componente raíz de una aplicación React.js que cuenta la biblioteca de enrutamiento de React Router, permitiendo la navegación basada en la URL de la aplicación.

```
ReactDOM.render(
  <HelmetProvider>
    <SidebarProvider>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </SidebarProvider>
  </HelmetProvider>,
  document.getElementById('root')
);
```

```
serviceWorker.unregister();
```

4.4.1.1.1 Componentes

Partiendo de las necesidades del proyecto, y haciendo uso de las grandes ventajas de React.js, se han creado una serie de componentes en la aplicación donde será hará uso de ellos en distintos apartados.

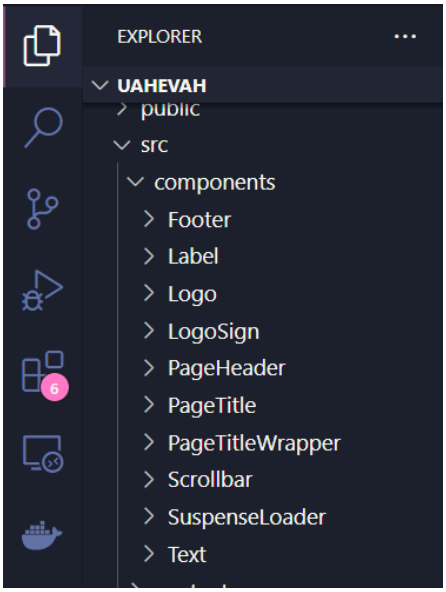


Figura 29: Presentación de componentes creados en la aplicación frontal con React.js

Aquí se muestra un ejemplo de código sobre la creación del componente 'PageHeader', el cual se muestra en las distintas pantallas de la aplicación.

```
function PageHeader({ componentName }: { componentName: string }) {
  const name = componentName.charAt(0).toLowerCase() +
componentName.slice(1);
  const route = name.slice(0, -1);
  const user = {
    name: 'Greys Rodrig',
    avatar: '/static/images/avatars/1.jpg'
  };
  return (
    <Grid container justifyContent="space-between" alignItems="center">
      <Grid item>
        <Typography variant="h3" component="h3" gutterBottom>
          {componentName}
        </Typography>
        <Typography variant="subtitle2">
          {user.name}, these are your recent {name}
```

```
        </Typography>
      </Grid>
      <Grid item>
        <Link to={`/dashboards/${name}/new${route}`}>
          <Button
            sx={{ mt: { xs: 2, md: 0 } }}
            variant="contained"
            startIcon={<AddTwoToneIcon fontSize="small" />}
          >
            Create {componentName}
          </Button>
        </Link>
      </Grid>
    </Grid>
  );
}
export default PageHeader;
```

4.4.1.2 TypeScript

TypeScript puede considerarse como una especie de contenedor de JavaScript o una extensión de este. Hoy es uno de los lenguajes más populares para crear aplicaciones web. Se compila en JavaScript, todos los bits de TypeScript se eliminan, dejando simplemente el antiguo JavaScript.

Los tipos de datos, por ejemplo, son puramente una construcción en tiempo de desarrollo. Una vez que se produce la compilación, desaparecen. Pero, debido a que estuvieron allí durante el desarrollo, el compilador señalará los errores relacionados con el tipo en ese momento, en lugar de descubrirlos en tiempo de ejecución, como es el caso de JavaScript en general [12].

Para consolidar un contexto sobre la utilización de TypeScript en el proyecto, a continuación, visualizaremos un pequeño ejemplo de código, el cual proporciona una interfaz llamada ‘Client’ que describe la estructura o forma que deben tener los objectos que representan los clientes en la aplicación.

```
export interface Client {
  idClient: number;
  firstName: string;
  lastName: string;
  address: string;
  phone: string;
  email: string;
}
```

Entendiendo mejor el contexto, con esta estructura deben recibirse los objectos provenientes desde la base de datos de la aplicación con esta misma estructura, para que estos puedan ser correctamente aplicables para mostrar en el frontal.

4.4.1.3 Material UI (MUI)

MUI es una biblioteca de componentes React.js de código abierto que implementa Material Design de Google. Incluye una colección completa de componentes prediseñados que están listos para su uso en producción. Tiene un hermoso diseño y presenta un conjunto de opciones de personalización que facilitan la implementación de su propio sistema de diseño personalizado sobre nuestros componentes [13].

El uso de material ha brindado un aire de minimalismo y claridad al proyecto, aportando una interfaz de usuario mucho más limpia y clara para brindar mayor conformidad a aquellos que la utilizan.

Un ejemplo claro de cómo implementar MUI en un proyecto React.js es el siguiente. Donde se hace uso de sus componentes ‘Container’ y ‘Grid’ los cuales permiten crear un contenedor que centra su contenido horizontalmente y una cuadricula responsiva que se adapta al tamaño y la orientación de la pantalla, garantizando la coherencia entre los diseños.

```
import { Helmet } from 'react-helmet-async';
import PageHeader from '../../../../components/PageHeader/PageHeader';
import PageTitleWrapper from 'src/components/PageTitleWrapper';
import { Grid, Container } from '@mui/material';
import Footer from 'src/components/Footer';
import CreateEdit from './CreateEdit';
function Form() {
  return (
    <>
      <Helmet>
        <title>New Client</title>
      </Helmet>
      <PageTitleWrapper>
        <PageHeader componentName="Clients" />
      </PageTitleWrapper>
      <Container maxWidth="lg">
        <Grid
          container
          direction="row"
          justifyContent="center"
          alignItems="stretch"
          spacing={3}
        >
          <Grid item xs={12}>
            <CreateEdit />
```

```
          </Grid>
        </Grid>
      </Container>
      <Footer />
    </>
  );
}
export default Form;
```

4.4.1.4 Formik

Formik es una librería declarativa que se encarga de manejar todos los posibles escenarios al interactuar con un input, de cualquier tipo que este sea. Realizando una declaración compuesta del estado que serán los campos de información que se quiere capturar [14].

Para identificar una breve implementación, confirmaremos a continuación como formik junto a los componentes de MUI, ayudan a gestionar los datos en el formulario de creación o edición de nuevos 'Clients'.

```
const CreateEdit: React.FC<{}> = () => {
  const { success, error } = useNotification();
  const { idClient } = useParams();
  const navigate = useNavigate();
  const URL = '/clients';
  const getClient = async () => {
    const respond = await apiUrls.API_URL_MS1.get(`${URL}/${idClient}`);
    formik.setValues(respond.data);
  };
  useEffect(() => {
    if (idClient) {
      getClient();
    }
  }, [idClient]);
  const initialValues: Client = {
    idClient: 0,
    firstName: '',
    lastName: '',
    address: '',
    phone: '',
    email: ''
  };
  const validationSchema = Yup.object({
    firstName: Yup.string().required('First name is required'),
    lastName: Yup.string().required('Last name is required')
  });
  const formik = useFormik({
```

```
  initialValues,
  validationSchema,
  onSubmit: async (values) => {
    try {
      if (!idClient) {
        await apiUrls.API_URL_MS1.post(`${URL}`, values);
      } else {
        await apiUrls.API_URL_MS1.put(`${URL}`, values);
      }
      success();
      setTimeout(() => {
        navigate('/dashboards/clients');
      }, 3000);
    } catch (err) {
      console.log(err);
      error();
    }
  }
});
return (
  <Card>
    {!idClient ? (
      <CardHeader title="New Client" />
    ) : (
      <CardHeader title="Edit Client" />
    )}
    <Divider />
    <CardContent>
      <Box
        sx={{
          '& .MuiTextField-root': { m: 1, width: '25ch' }
        }}
      >
        <div>
          <form onSubmit={formik.handleSubmit}>
            <Grid container spacing={2}>
              <Grid item xs={12} lg={6}>
                <TextField
                  style={{ width: '98%' }}
                  required
                  label="First Name"
                  type="text"
                  name="firstName"
                  value={formik.values.firstName}
                  onChange={formik.handleChange}
                />
```

```
                  {formik.touched.firstName && formik.errors.firstName ?
(
                    <div>{formik.errors.firstName}</div>
                  ) : null}
                </Grid>
                <Grid item xs={12} lg={6}>
                  <Button
                    style={{ width: '98%', margin: '9px' }}
                    type="submit"
                    size="large"
                    sx={{ mt: { xs: 2, md: 0 } }}
                    variant="contained"
                  >
                    Save
                  </Button>
                </Grid>
                <Grid item xs={12} lg={6}>
                  <Button
                    style={{ width: '98%', margin: '9px' }}
                    onClick={() => navigate(`/dashboards/clients`)}
                    size="large"
                    sx={{ mt: { xs: 2, md: 0 } }}
                    variant="outlined"
                    color="error"
                  >
                    Cancel
                  </Button>
                </Grid>
              </Grid>
            </form>
          </div>
        </Box>
      </CardContent>
    </Card>
  );
};
export default CreateEdit;
```

4.4.2 Vistas de la aplicación

Una vez identificada gran parte de la implicación de las tecnologías para poder lograr el desarrollo de la aplicación, a continuación, vistas generadas para la aplicación y algunos ejemplos de componentes creados para ser reutilizados en las diferentes vistas.

4.4.2.1 Ingreso a la aplicación

Se ha creado una pantalla principal básica, que al pinchar en el botón ‘Sign In’ redirecciona a la sección del Dashboard principal. Por medio de los objetivos del proyecto, no fue requerido crear una gestión para el ingreso y registros de usuarios a la aplicación. Añadir este apartado de ‘LogIn’ y ‘LogOut’, mediante esta pantalla o vista principal, sería de mayor facilidad lograr la incorporación.

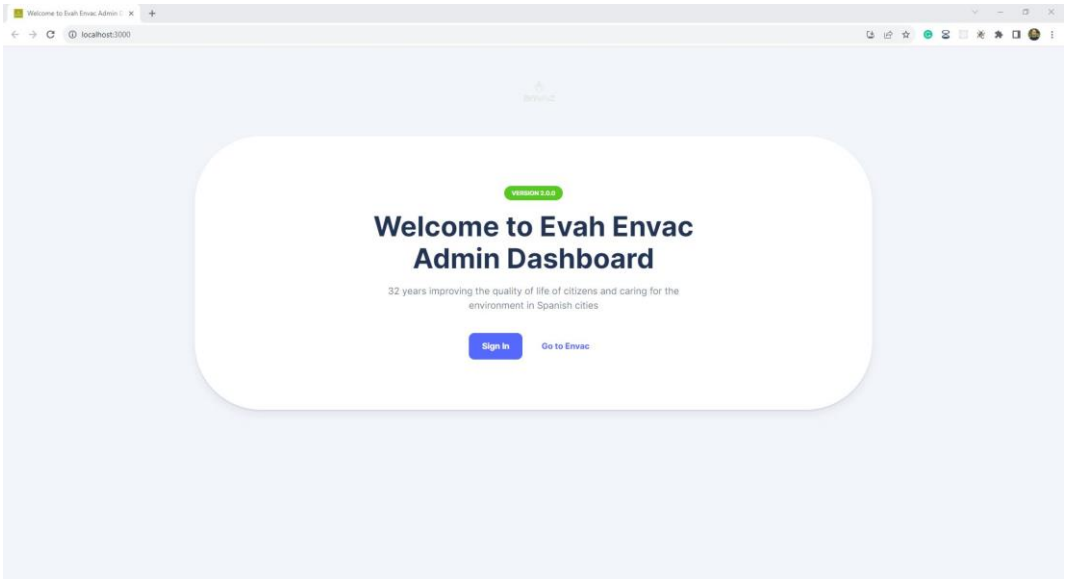


Figura 30: Vista web - Pantalla principal de acceso a la aplicación envah.

4.4.2.2 Dashboard (panel)

Mediante esta pantalla de dashboard, se puede acceder a las distintas vistas de la aplicación. Logrando ingresar a las listas y formularios CRUD, como Clients, Collections, Incidents, Personnels, Routes, entre otras…

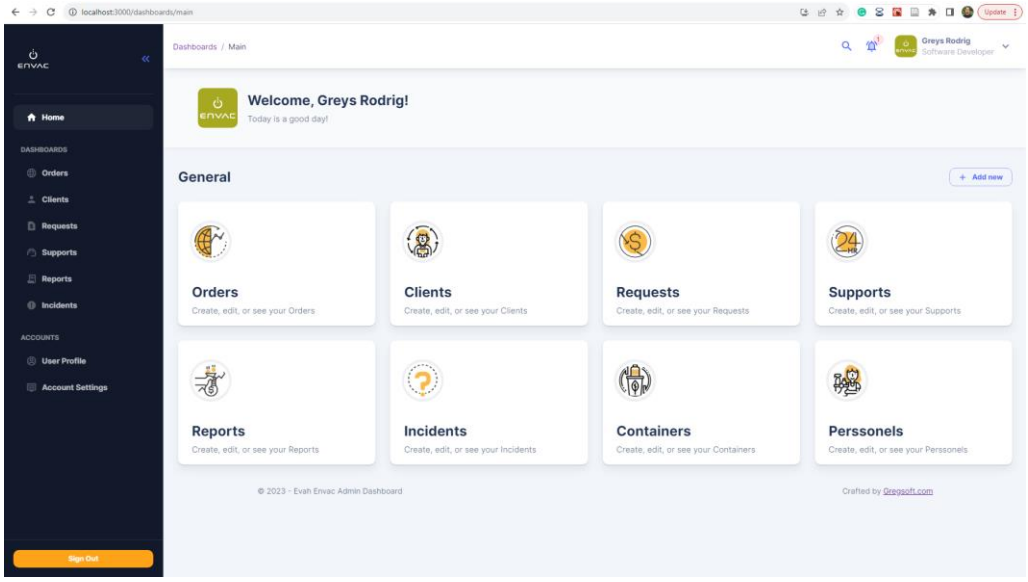


Figura 31: Vista web - Pantalla principal Dashboard (panel) con acceso a las distintas vistas de la aplicación.

4.4.2.2.1 Menú lateral (Sidebar)

En el diseño de la interfaz, en la parte izquierda, se incorpora un componente de Menú Lateral que puede retraerse, expandiendo el contenido principal, y en el que se presenta una lista de accesos a las diversas vistas, todas ellas interactivas.

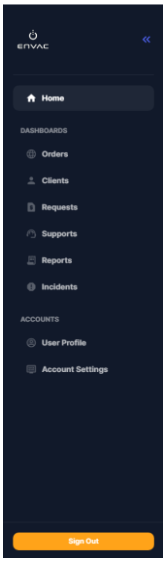


Figura 32: Vista web – Sidebar expandido

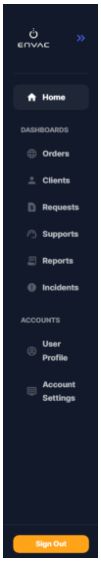


Figura 33: Vista web – Sidebar retraído

```
return (
    <>
      <MenuWrapper>
        <List component="div">
          <SubMenuWrapper>
```

```
            <List component="div">
              <ListItem component="div">
                <Button
                  disableRipple
                  component={RouterLink}
                  onClick={closeSidebar}
                  to="/dashboards/main"
                  startIcon={<HomeIcon />}
                >
                  Home
                </Button>
              </ListItem>
            </List>
          </SubMenuWrapper>
        </List>
      </MenuWrapper>
    </>
  );
export default SidebarMenu;
```

4.4.2.2.2 Header (Encabezado)

En el centro, se destaca la inclusión del componente Header (Encabezado) de página, que se visualiza de forma constante en toda la aplicación. Justo debajo de este encabezado, el contenido se adapta dinámicamente según la sección en el que te encuentres. Esto proporciona una interfaz sencilla y dinámica que permite a los usuarios entender fácilmente el comportamiento de la aplicación y mantiene una consistencia en todos los elementos visuales.

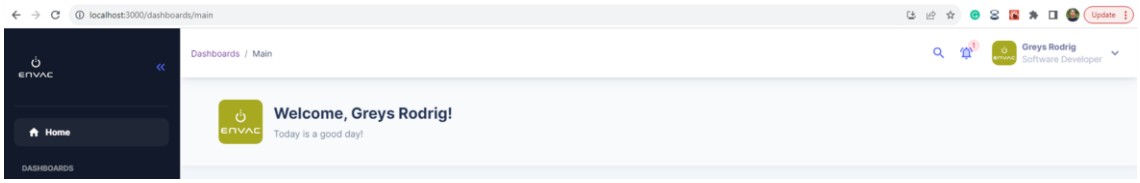


Figura 34: Vista web – Pantalla Dashboard - Encabezado de la aplicación.

```
return (
    <Box display="flex" alignItems="center" sx={headerStyle}>
      <Stack
        direction="row"
        divider={<Divider orientation="vertical" flexItem />}
        alignItems="center"
        spacing={2}
      >
```

```
        <HeaderMenu />
      </Stack>
      <Box display="flex" alignItems="center">
        <HeaderButtons />
        <HeaderUserbox />
        <Box
          component="span"
          sx={{
            ml: 2,
            display: { lg: 'none', xs: 'inline-block' }
          }}
        >
          <Tooltip arrow title="Toggle Menu">
            <IconButton color="primary" onClick={toggleSidebar}>
              {!isSidebarRetracted ? (
                <MenuTwoToneIcon fontSize="small" />
              ) : (
                <CloseTwoToneIcon fontSize="small" />
              )}
            </IconButton>
          </Tooltip>
        </Box>
      </Box>
    </Box>
  );
```

4.4.2.3 Tipos de Listas (Views)

4.4.2.3.1 Tabla

Presentando los datos tabulares en una vista de lista dentro de una tabla de manera visual en forma de filas y columnas. Se muestran dos tipos de tablas, con el objetivo de identificar las posibilidades que nos aporta React.js junto a Material UI (MUI) y la implementación de react-data-table-component. Una biblioteca de tablas simple pero flexible que surgió por necesidad mientras se desarrollaba una aplicación web para una startup en crecimiento que trae consigo la clasificación, ordenación, búsqueda, selección de filas, exportación de datos y paginación integradas [15].

4.4.2.3.2 Tabla MUI

Contando con un diseño coherente y siguiendo las pautas de diseño de Material Design, el componente Tabla puede integrarse fácilmente y permite una interacción con los datos muy efectiva, contando con funciones incorpora ordenación, paginación y filtros. En esta ocasión, se

logra filtrar las Órdenes por estados (status), partiendo de seleccionar las opciones de ‘Completed’, ‘Failed’ o ‘Pending’.

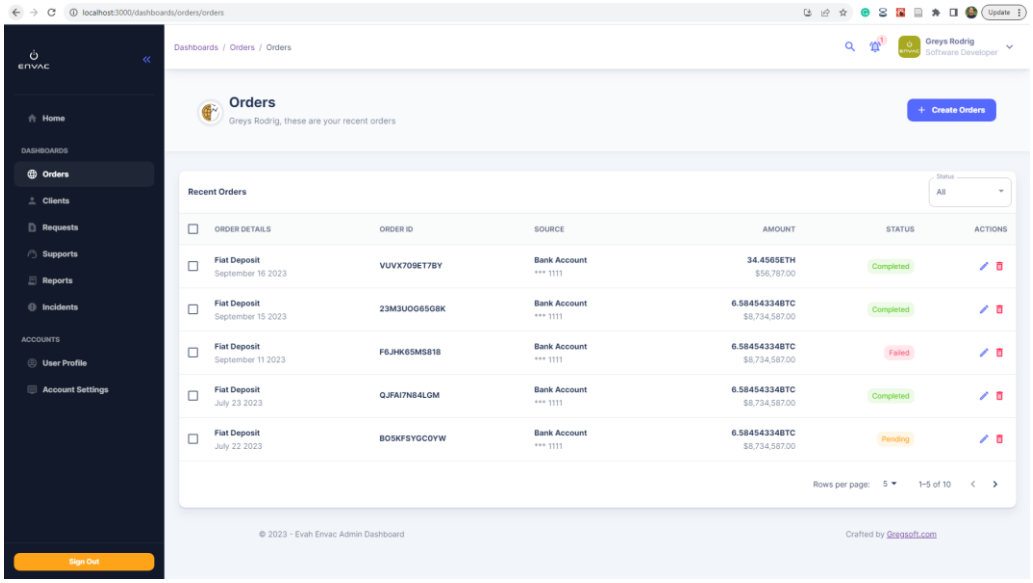


Figura 35: Vista web – Pantalla sobre lista de Órdenes.

```
<CardHeader
          action={
            <Box width={150}>
              <FormControl fullWidth variant="outlined">
                <InputLabel>Status</InputLabel>
                <Select
                  value={filters.status || 'all'}
                  onChange={handleStatusChange}
                  label="Status"
                  autoWidth
                >
                  {statusOptions.map((statusOption) => (
                    <MenuItem key={statusOption.id}
value={statusOption.id}>
                      {statusOption.name}
                    </MenuItem>
                  ))}
                </Select>
              </FormControl>
            </Box>
          }
          title="Recent Orders"
        />
```

4.4.2.3.3 React Data Table Component

Añadir esta librería ofrece una personalización muy flexible y sin la necesidad de escribir código complicado. Está diseñada para un rendimiento optimo con grandes conjuntos de datos y utiliza técnicas como el 'Virtual DOM' para minimizar las actualizaciones en la interfaz de usuario.

```
import DataTable, { TableColumn } from 'react-data-table-component';
const columns: TableColumn<Client>[] = [
    {
      name: '#',
      selector: (row) => row.idClient
    }
 ];
return (
    <>
      <DataTable
        title="List of Clients"
        columns={columns}
        data={filteredItems}
        progressPending={pending}
        pagination
        subHeader
        subHeaderComponent={subHeaderComponentMemo}
        customStyles={defaultTableStyles}
        dense
        responsive
        highlightOnHover
        pointerOnHover
        fixedHeader
        fixedHeaderScrollHeight="300px"
      />
    </>
  );
```

Basados en los objetivos del proyecto, por omisión, se muestran diez (10) elementos, logrando elegir visualizar hasta un máximo de treinta (30) elementos en total. Al seleccionar mostrar más de diez elementos, la tabla conserva su tamaño, permitiendo desplazarse hacia los elementos adicionales utilizando la barra de desplazamiento horizontal que se encuentra en el lado derecho de la tabla. La paginación de los elementos siguientes se realiza a través de los íconos '>' o '>|'.

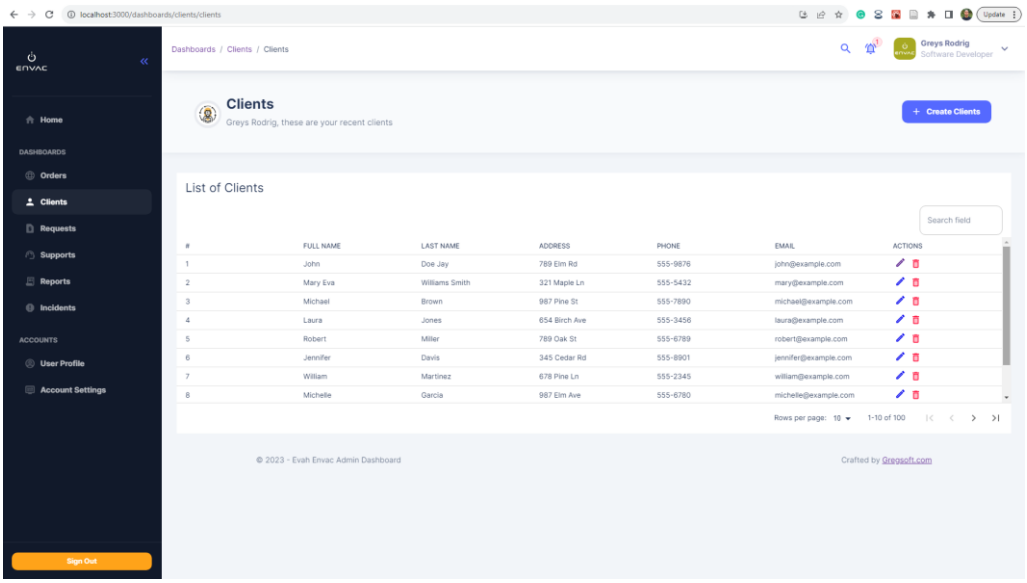


Figura 36: Vista web – Pantalla sobre lista de Clientes – React-Data-Table-Component.

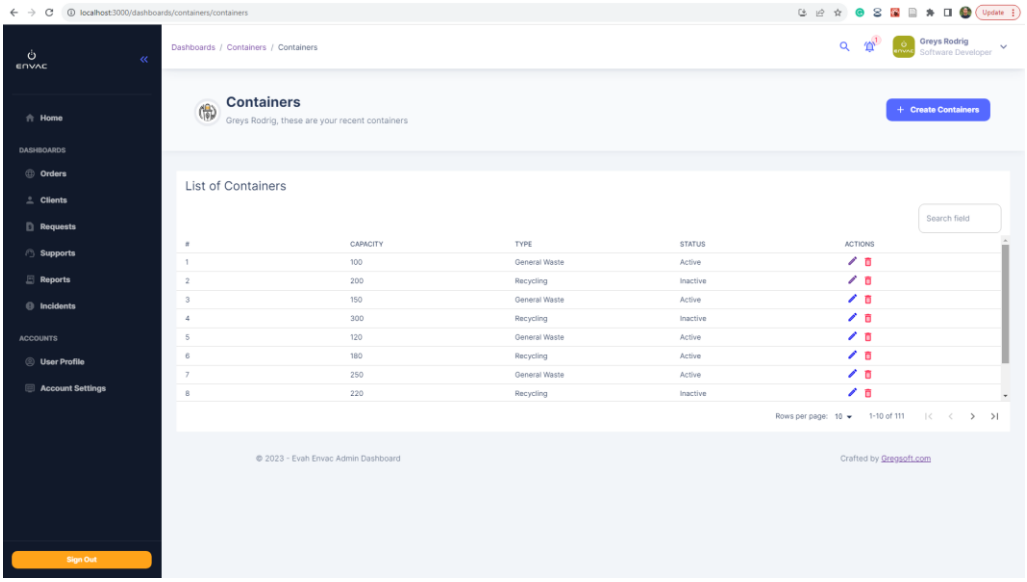


Figura 37: Vista web – Pantalla sobre lista de Contenedores – React-Data-Table-Component.

4.4.2.4 Rejillas (Grid)

Una de las características principales sobre la realización de este proyecto, es mostrar las posibilidades de interfaz con distintas combinaciones de apariencia de los listados sobre los datos relacionados con el negocio. En este caso, se incorporan las Rejillas (Grid) de MUI, un componente que permite cuadriculas flexibles y responsivas que se adaptan a diversos dispositivos y tamaños de pantallas por su base de división de 12 columnas. Dando lugar a poder utilizar el siguiente componente.

```
<Grid item xs={12} lg={6} />
```

4.4.2.5 Tarjetas (Card)

Las tarjetas son otros de los componentes de MUI, que poseen una apariencia coherente y moderna donde prácticamente permiten cualquier contenido dentro, como textos, imágenes, botones, listas o incluso componentes complejos. Su fácil utilidad y belleza logran revestir el proyecto en sus distintas listas a continuación, dándole una apariencia simple, limpia y con mucha frescura ante la vista por parte de los usuarios.

Utilizando estas tarjetas, se han creado listas de los componentes de ‘Incidents’, ‘Requests’, ‘Collections’, ‘Personnels’, ‘Routes’, entre otras… Por esta fácil versatilidad de poder incluir cualquier contenido dentro de estas tarjetas, su implementación hace presencia en los formularios de la aplicación.

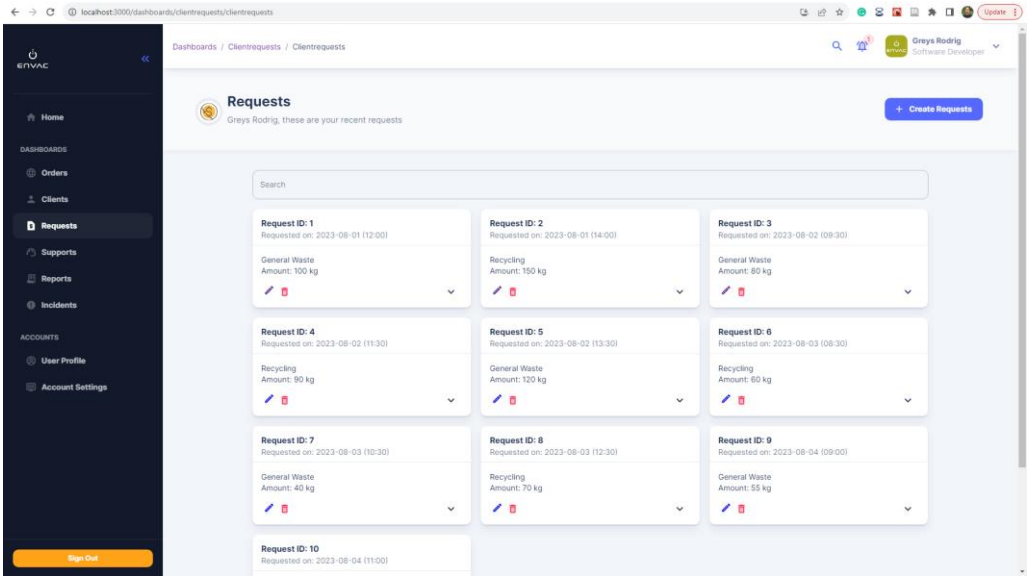


Figura 38: Vista web – Pantalla sobre lista de Requerimientos haciendo uso de Grillas de MUI.

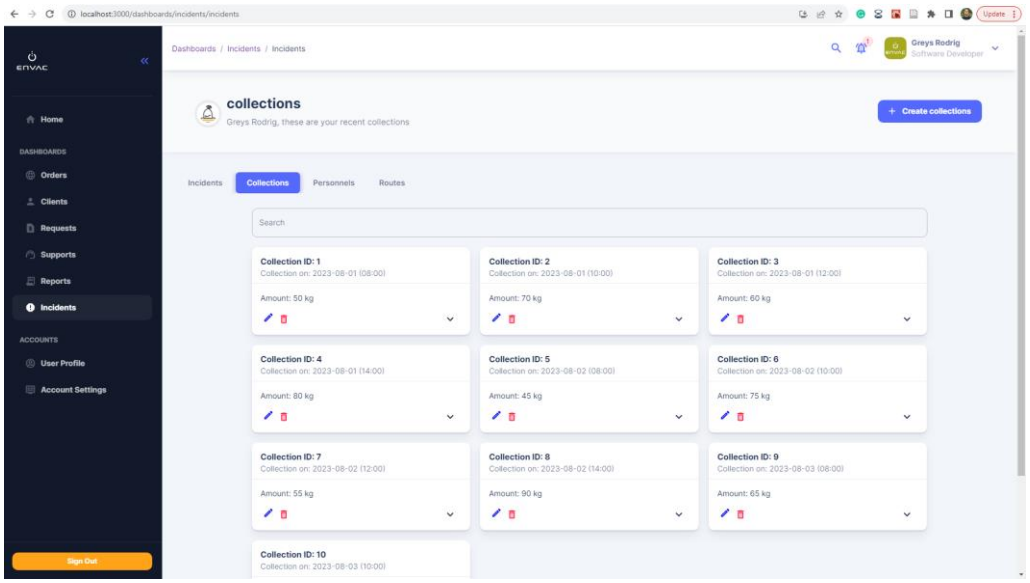


Figura 39: Vista web – Pantalla sobre lista de Colecciones haciendo uso de Grillas de MUI.

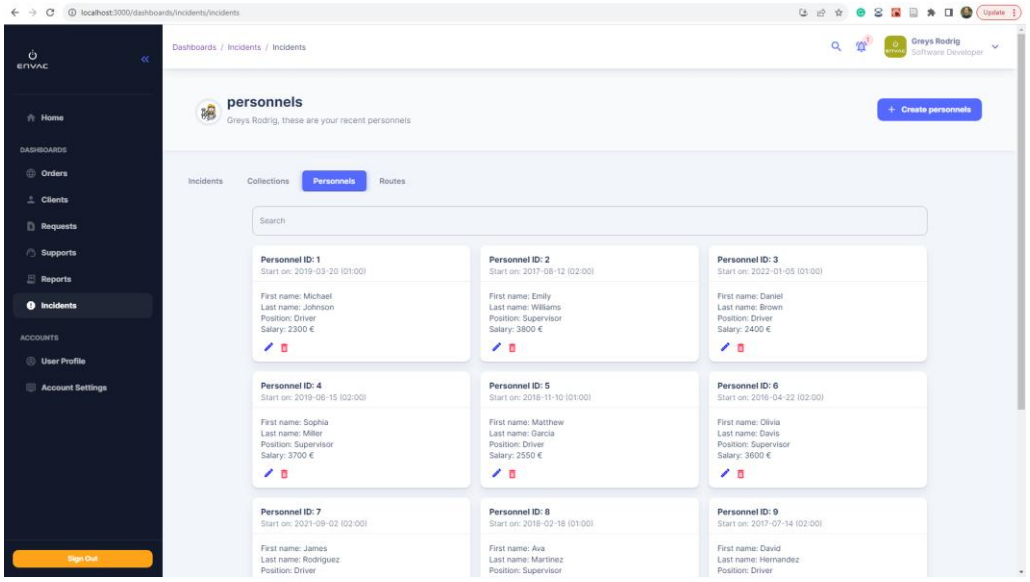


Figura 40: Vista web – Pantalla sobre lista de Personal haciendo uso de Grillas de MUI.

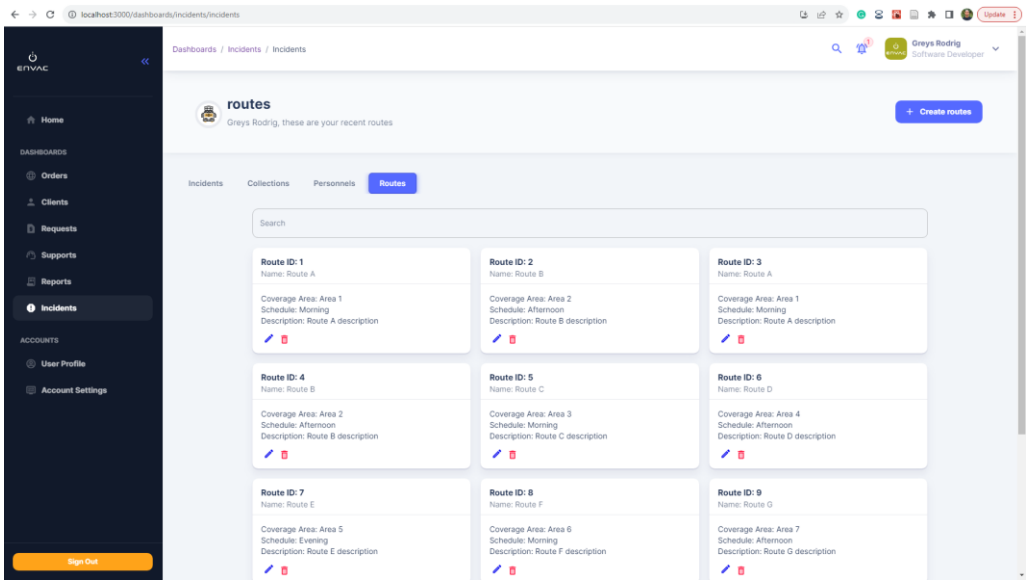


Figura 41: Vista web – Pantalla sobre lista de Rutas haciendo uso de Grillas de MUI.

4.4.2.6 Etiquetas (Tag)

Identificadas las vistas previas, resulta sencillo navegar entre pantallas en la parte superior de las listas. Por ejemplo, encontrarse en la pestaña ‘Incidents’, se puede acceder fácilmente a ‘Collections’, ‘Personnel’ o ‘Routs’ sin tener que abandonar la pantalla, lo que proporciona una experiencia fluida con tan solo unos pocos clics.

Las etiquetas o tag es un componente de MUI que representa distintos elementos HTML como botones, encabezados, cuadros de textos y otros elementos de interfaz de usuario.

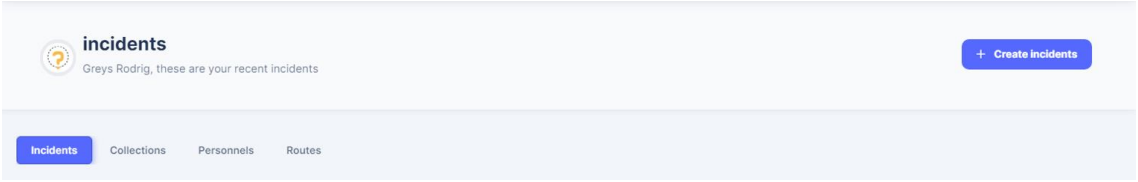


Figura 42: Vista web – Etiquetas (tag) sobre las listas ‘Incidents’, ‘Collections’, ‘Personnel’ y ‘Routs’.

```
const tabs = [
    { value: 'incidents', label: 'Incidents' },
    { value: 'collections', label: 'Collections' },
    { value: 'personnels', label: 'Personnels' },
    { value: 'routes', label: 'Routes' }
  ];
<Grid item xs={12}>
            <TabsWrapper
              onChange={handleTabsChange}
              value={currentTab}
```

```
              variant="scrollable"
              scrollButtons="auto"
              textColor="primary"
              indicatorColor="primary"
            >
              {tabs.map((tab) => (
                <Tab key={tab.value} label={tab.label} value={tab.value}
/>

              ))}
            </TabsWrapper>
          </Grid>
```

4.4.2.7 Campos de Textos (Text Field)

Se añade una función de búsqueda que se encuentra justo debajo de las etiquetas mencionadas anteriormente. Esta función permite filtrar los elementos según los términos contenidos en las tarjetas. Para aclarar, ubicados en la vista de 'Routes' se escríbeme 'Morning' en el campo de búsqueda (Search), se obtendrá todas las colecciones que contengan este término. Esto se aplica de manera similar en todas las distintas vistas.

Los campos de textos no son más que componente Material-UI (MUI) para la entrada y visualización de texto en los formularios y otras áreas de la interfaz de usuario.

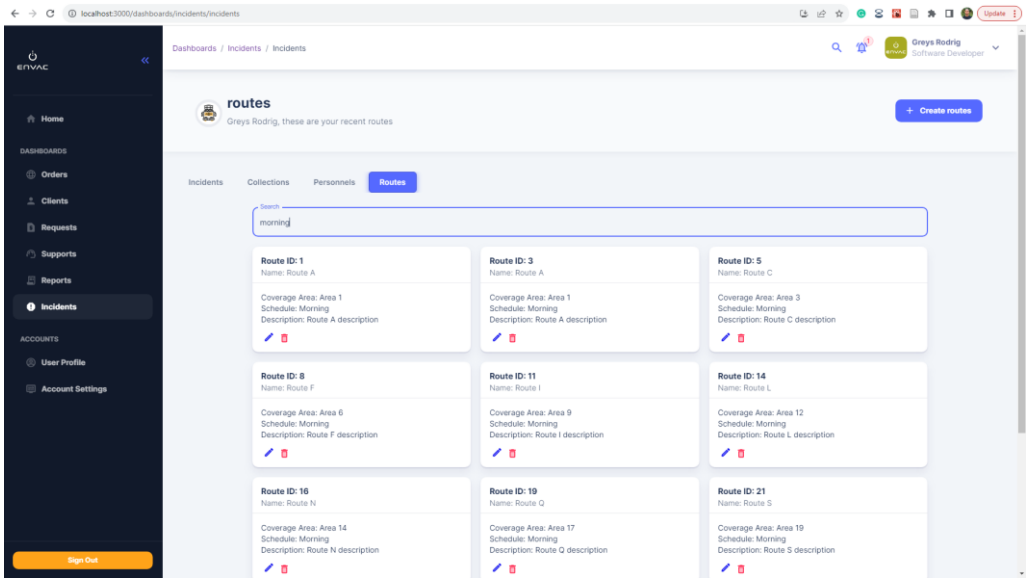


Figura 43: Vista web – Pantalla sobre filtro realizado mediante campo de texto 'Search'.

```
const filterRoutes = (text, collectionRoutes) => {
    const searchText = text.toLowerCase();
    return collectionRoutes.filter((item) => {
```

```
      const { name, description, collectionSchedule, coverageArea } =
item;
      return (
        name.toLowerCase().includes(searchText) ||
        description.toLowerCase().includes(searchText) ||
        collectionSchedule.toLowerCase().includes(searchText) ||
        coverageArea.toLowerCase().includes(searchText)
      );
    });
  };
```

4.4.2.8 Formularios (Forms)

Para logar la introducción de datos en la base de datos y llevar a cabo la mayoría de las operaciones CRUD mencionadas previamente en la sección sobre Pruebas API, es esencial incorporar formularios en una aplicación web. Estos formularios son componentes diseñados específicamente para gestionar y recopilar datos a través de la interfaz de usuario, y desempeñan un papel fundamental en el funcionamiento fluido de la aplicación.

En este proyecto, se integran formularios que incluyen los componentes y funcionalidades estudiadas, lo que simplifica tanto su creación como su proceso de validación.

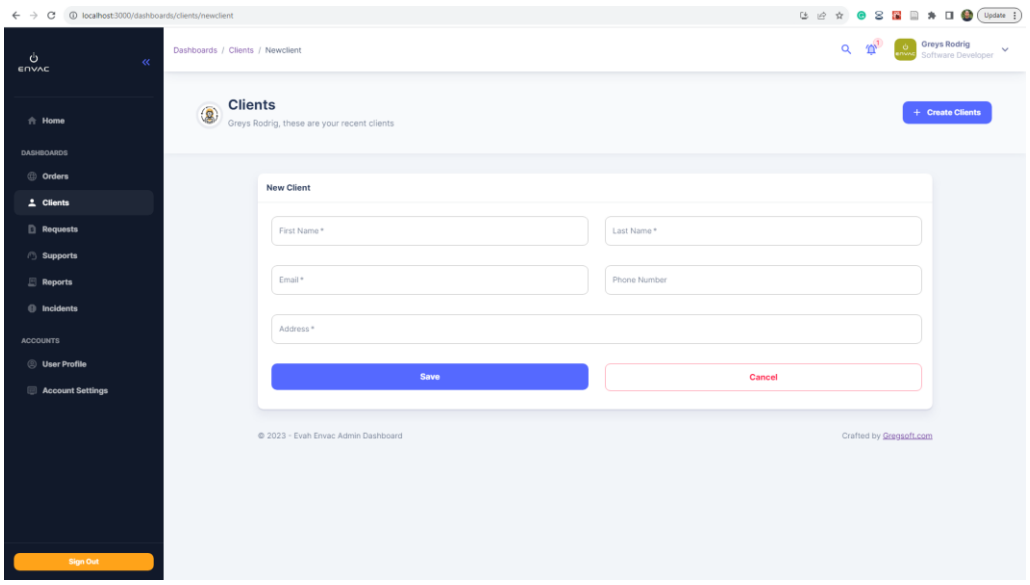


Figura 44: Vista web – Pantalla sobre ingresar/registrar nuevo Cliente.

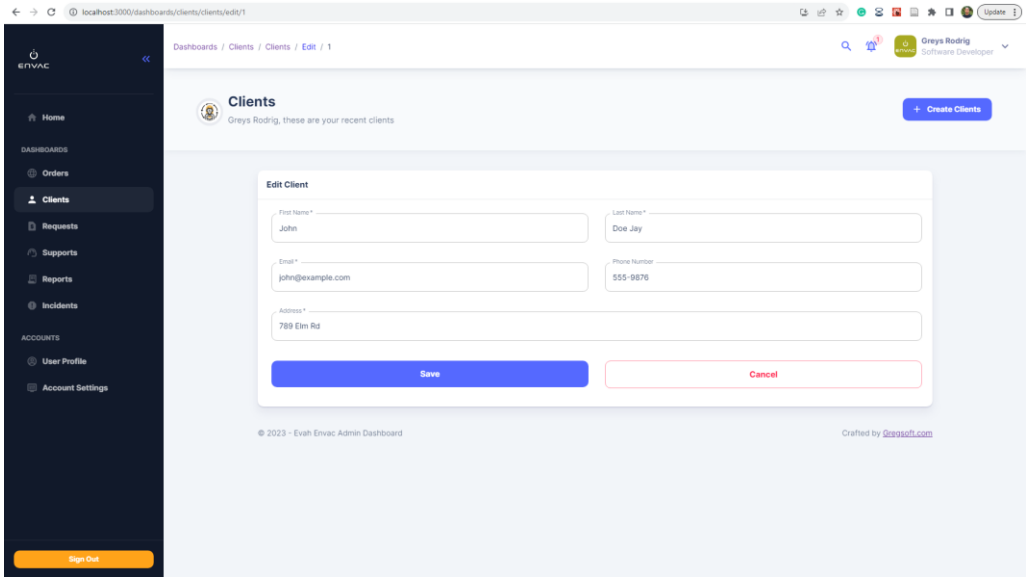


Figura 45: Vista web – Pantalla sobre editar nuevo Cliente.

4.4.3 Simplificación e Integración

Con el fin de alcanzar los objetivos del proyecto y demostrar las diversas posibilidades de interfaz para presentar información relacionada con el negocio, se ha desarrollado la vista de incidencias. Esta vista se ha diseñado cuidadosamente para aprovechar todos los componentes disponibles y mejorar significativamente la capacidad de filtrado en el campo de búsqueda, además de simplificar este proceso mediante la incorporación de botones dedicados.

Entre las mejoras implementadas se incluyen la adición de colores, una mayor capacitación del contenido y una organización más sofisticada de los elementos, todo ello con el propósito de cumplir plenamente con las características requeridas por este proyecto.

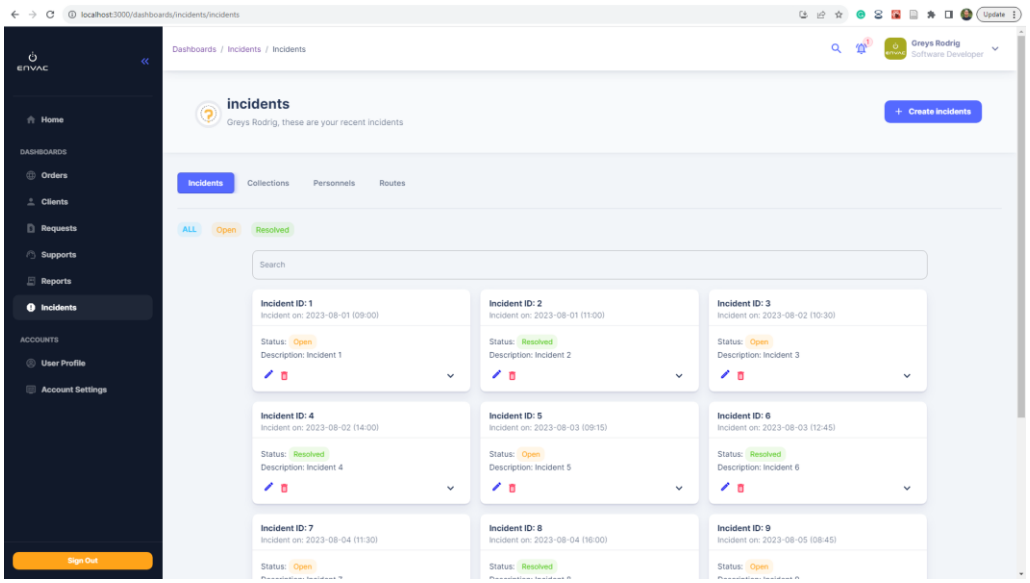


Figura 46: Vista web – Pantalla sobre simplificación e integración de componentes y filtrados.

4.4.3.1 Filtros

La inclusión de botones específicos para filtrar en el campo de búsqueda agiliza significativamente el uso de los componentes, logrando los siguientes objetivos:

- Filtrar mediante los botones ‘Open’ y ‘Resolved’.
- Filtrar mediante el campo de búsqueda.
- Realizar un filtro de contenido utilizando el campo de búsqueda junto con los botones específicos.

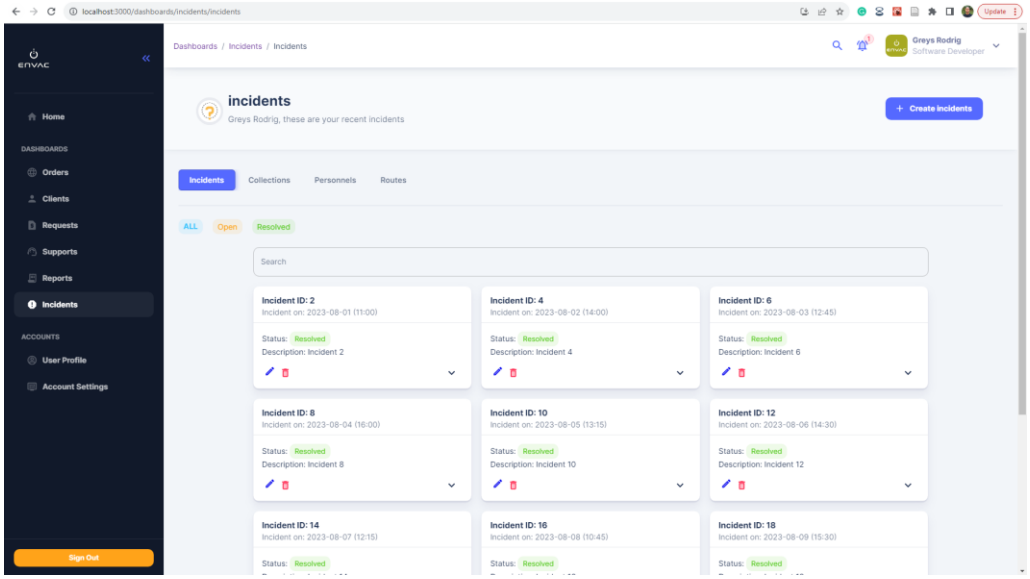


Figura 47: Vista web – Pantalla sobre contenido filtrado mediante el botón 'Resolved'.

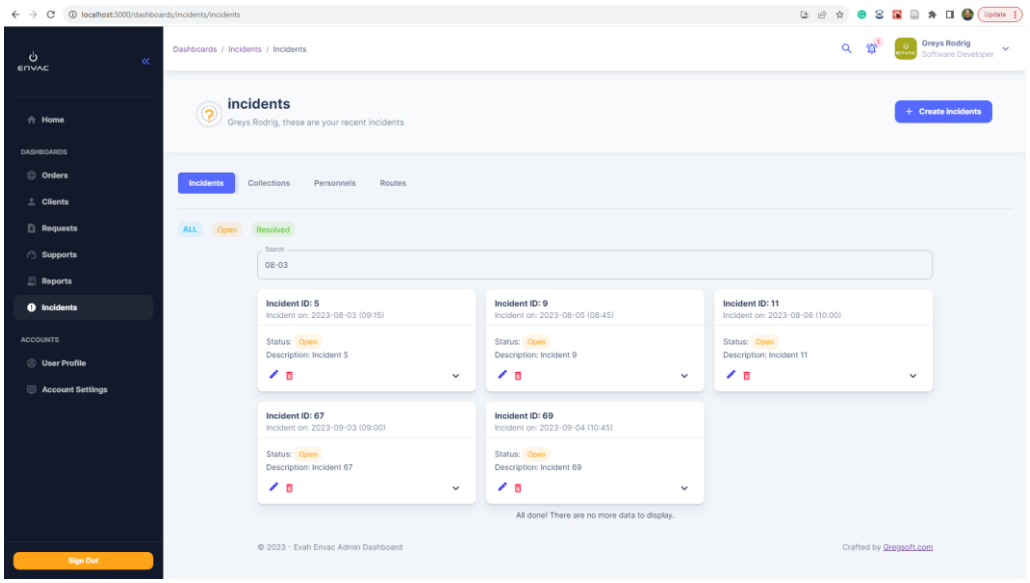


Figura 48: Vista web – Pantalla sobre contenido filtrado mediante el campo de búsqueda y el botón 'Open'.

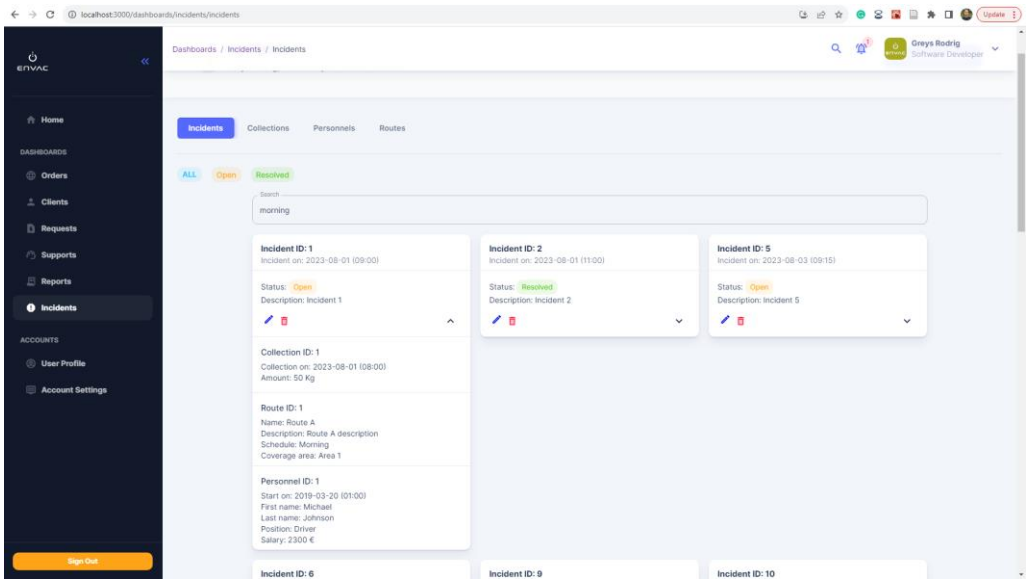


Figura 49: Vista web – Pantalla sobre contenido filtrado mediante el campo de búsqueda y el botón 'All' – Despliegue de Tarjeta.

4.4.3.2 Desplazamiento Infinito (Scroll)

Uno de los desafíos clave al implementar las listas con tarjetas fue la eficiente carga de datos y la necesidad de habilitar un desplazamiento fluido hacia abajo para acceder a los elementos restantes en la lista.

Para abordar este desafío, se implementa una solución eficiente que carga los datos de forma progresiva a medida que el usuario se desplaza hacia abajo en la lista. Por defecto, limitar la cantidad de elementos mostrados a diez (10) a la vez. Esto tiene el beneficio de reducir significativamente el tiempo de espera del usuario antes de que la lista este completamente preparada para su uso.

Además, esta estrategia beneficia tanto al cliente como al servidor. En el lado del cliente minimiza la cantidad de datos que se deben procesar y muestra, mejorando así la experiencia de usuario y evitando desbordamientos de memoria. En el lado del servidor, reduce la carga de trabajo al enviar solo la cantidad necesaria de datos en cada solicitud.

En resumen, esta solución no solo mejora la eficiencia en la carga de datos, sino también optimiza la experiencia del usuario y los recursos tanto del cliente como el servidor.

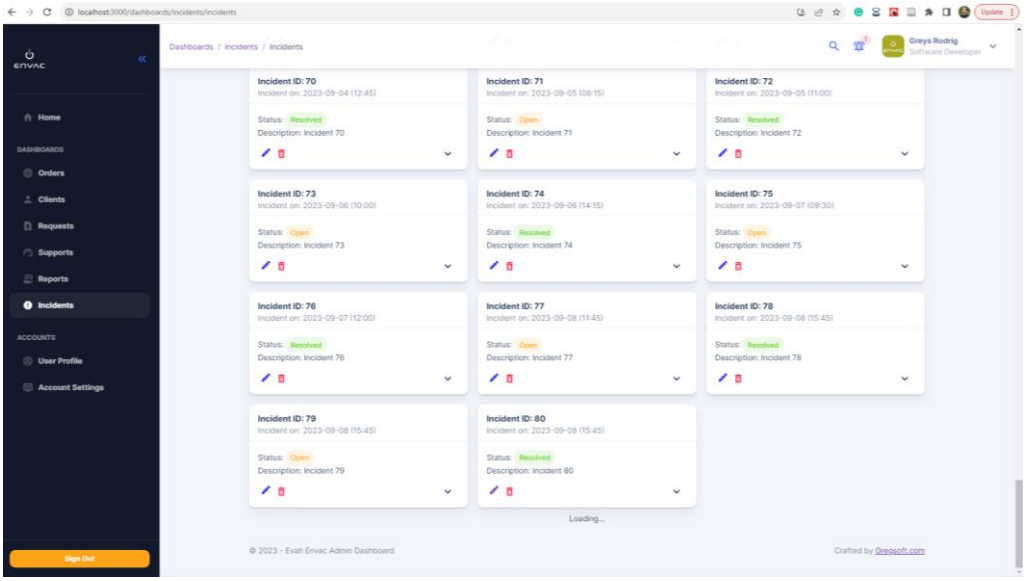


Figura 50: Vista web – Pantalla sobre implementación de desplazamiento infinito (scroll).

```
const [visibleCount, setVisibleCount] = useState(10);
const [itemsToLoad, setItemsToLoad] = useState(10);
const [showLoadingMessage, setShowLoadingMessage] = useState(false);
const [loadingMore, setLoadingMore] = useState(false);

useEffect(() => {
    const handleScroll = () => {
      const scrollY = window.scrollY || window.pageYOffset;
      if (
        window.innerHeight + scrollY >= document.body.offsetHeight &&
        !loadingMore
      ) {
        setLoadingMore(true);
        setShowLoadingMessage(true);
        setVisibleCount((prevVisibleCount) => prevVisibleCount +
itemsToLoad);
        setTimeout(() => {
          setLoadingMore(false);
          setShowLoadingMessage(false);
        }, 1000);
      }
    };
    window.addEventListener('scroll', handleScroll);

    return () => {
      window.removeEventListener('scroll', handleScroll);
    };
  }, [loadingMore, itemsToLoad]);
```

4.4.3.3 Notificaciones

Es esencial mantener a los usuarios debidamente informados acerca de las acciones llevadas a cabo en un sistema de software. Esto contribuye significativamente a la claridad y transparencia en la interacción con la aplicación. La sensación de control y transparencia, junto con la relevancia de los eventos o cambios en el sistema, enriquecen la experiencia de usuario y refuerzan la seguridad, lo que, a su vez, eleva la confiabilidad del software.

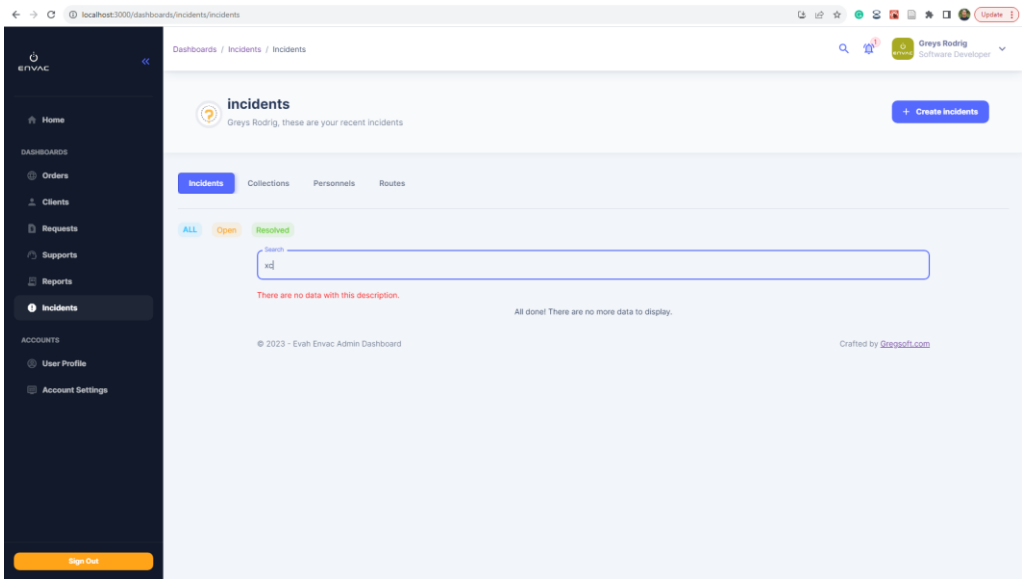


Figura 51: Vista web – Pantalla sobre mensaje de contenido filtrado no encontrado.

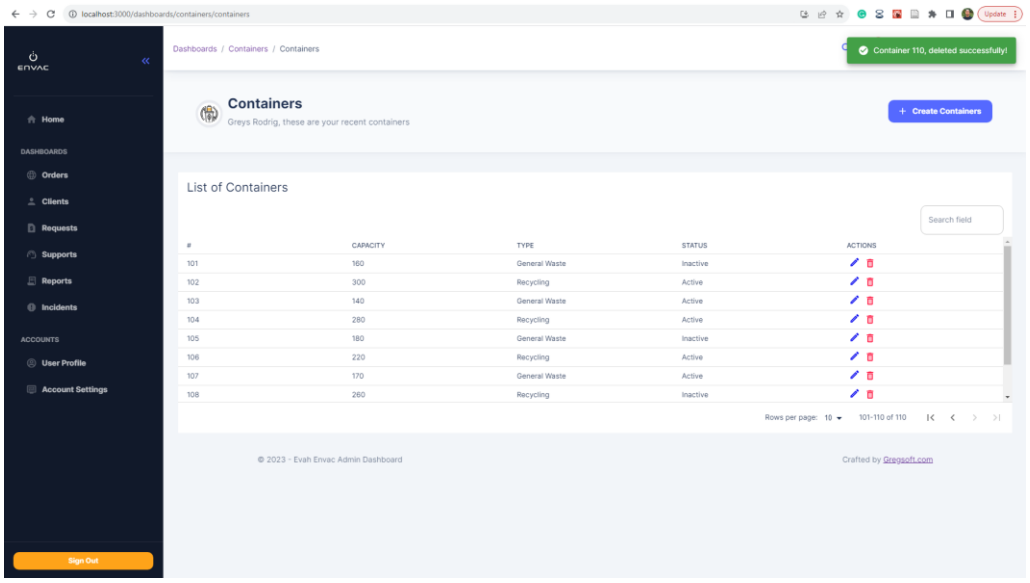


Figura 52: Vista web – Pantalla sobre mensaje de notificación de contenido eliminado de la aplicación.

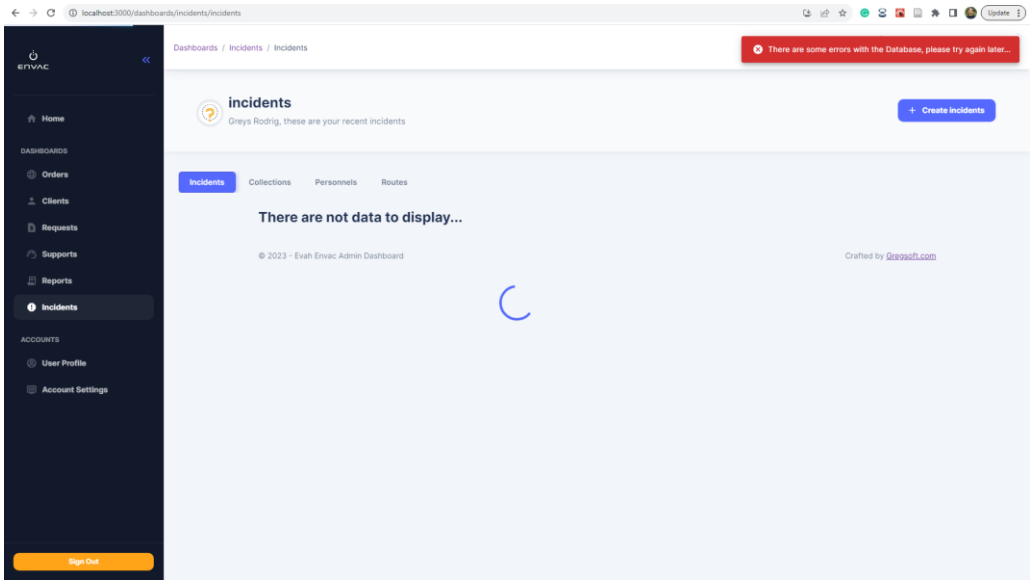


Figura 53: Vista web – Pantalla sobre mensaje de notificación de contenido no disponible desde la base de datos.

5. Conclusiones

A medida que avanzamos en el mundo del desarrollo de software, es esencial seguir explorando y aplicando nuevas tecnologías y metodologías para abordar los desafíos en constante evolución. Este proyecto es un testimonio de la importancia de la innovación y la adaptabilidad en la búsqueda de soluciones efectivas. Ha demostrado que la modernización de la interfaz de usuario de la aplicación EVAH utilizando tecnologías de frameworks avanzados y una arquitectura de microservicios mejora significativamente la eficiencia y la experiencia de usuario.

En última instancia, este trabajo sienta las bases para futuras mejoras y expansiones en la aplicación EVAH y sirve como ejemplo de cómo la investigación y el desarrollo pueden converger para lograr avances significativos en la tecnología y la resolución de problemas del mundo real.

6. Trabajo futuro

En el contexto de una futura actualización del proyecto, se plantean una serie de mejoras que incorporan componentes de filtrado adicionales y ofrezcan alternativas más avanzadas para la obtención de datos altamente específicos. Dada la creciente importancia del Big Data, la capacidad de obtener datos personalizados de manera ágil se vuelve cada vez más esencial. La potencial implementación de estas mejoras podría representar una valiosa adición al proyecto, brindado herramientas más poderosas para satisfacer las demandas cambiantes de datos en un entorno en constante evolución.

Sin duda, continuar aprendiendo sobre nuevos frameworks que permiten una integración con las tecnologías utilizadas en el proyecto que juntas pudieren satisfacer necesidades de mayor complejidad y que brinden funcionalidades significativas en el desarrollo de software web.

7. Coste del proyecto

En el siguiente análisis de costes para nuestro proyecto de desarrollo de aplicación web, se desglosa de manera concisa los gastos relacionados con mano de obra, equipos y servicios, así como software. Este desglose es esencial para una gestión eficiente de recursos y un control financiero efectivo.

7.1 Coste de mano de obra

Para gestionar eficazmente los costes de mano de obra en el desarrollo de esta aplicación durante un período de tres meses, asumiendo una jornada laboral de 8 horas de lunes a viernes, contaremos con un desarrollador web líder, un diseñador UI para mejorar la experiencia del usuario, un ingeniero de base de datos para la gestión segura de los datos y un experto en seguridad para la protección del proyecto. Además, emplearemos un DevOps para administrar los microservicios de manera eficiente y un líder de proyecto ágil para implementar metodologías ágiles.

Concepto	Cantidad	Horas	Coste/Hora	Coste total
Desarrollador web	1	88	35.00	3,080.00
Diseñador de Interfaz de Usuario (UI)	1	88	22.00	1,936.00
Ingeniero de Base de Datos	1	88	25.00	2,200.00
Ingeniero de Seguridad Informática	1	88	31.00	2,728.00
Ingeniero DevOps	1	88	39.00	3,432.00
Líder de proyecto ágil	1	88	33.00	2,904.00
Total	**6**	**528**	**185.00**	**16,280.00**

Tabla 2: Resultados de costes de mano de obra.

En relación con los distintos roles que contempla el proyecto, las labores realizadas por cada miembro del equipo corresponden a:

- **Desarrollador Web**: crear la estructura y la funcionalidad junto a los componentes e implementación de los frameworks y librerías de todas las capas del proyecto, así como convertir los diseños proporcionados por el diseñador UI, a código para el desarrollo de las distintas pantallas como ‘Clients’, ‘Continers’, ‘Incidents’, otras…

- **Diseñador UI**: ha desarrollado toda la apariencia visual y la disposición de los elementos en la interfaz de usuario, asegurando una apariencia coherente y atractiva.

- **Ingeniero de Bases de Datos**: su principal función fue diseñar, implementar y optimizar el sistema de gestión de la base de datos, adoptando una estructura en base a las necesidades del proyecto.

- **Ingeniero de Seguridad informática**: considerar algunas de las necesidades elementales para proteger el ecosistema de la aplicación. Evaluando los posibles riesgos para junto con el equipo incorporación lo considerado.

- **Ingeniero DevOps**: encargado de facilitar la colaboración y la integración continua de los equipos, mitigando la evolución del proyecto y mayor escalabilidad, validando la arquitectura adecuada, la correcta implementación y funcionamiento.

- **Líder de proyecto ágil**: con el objetivo de seguir los principios y practicas agiles, facilito la incorporación de las metodologías a utilizar en el proyecto y la forma para definir la visión y los objetivos del proyecto. Asumió las necesidades de los interesados y planificó y gestiono las iteraciones, priorizando el trabajo de mayor importancia.

7.2 Coste de materiales

Considerando la vida útil de los dispositivos, se procede a calcular los costes de equipos y servicios.

Costes de Hardware

Descripción	Número	Coste	Total
Ordenador Intel I7 con memoria RAM 32bits, 250 SSD	2	1,200.00	2,400.00
Ordenador Intel I5 con memoria RAM 32bits, 250 SSD	2	950.00	1,900.00
Ordenador Intel I5 con memoria RAM 16bits, 250 SSD	2	780.00	1,560.00
Total	6	2,930.00	**5,860.00**

Tabla 3: Resultados de costes de Hardware.

Costes de Servicio

Descripción	Hora	Coste	Total
Internet Trimestral	450	0.6	270.00
Electricidad	500	0.35	175.00
Otros servicios	200	0.20	40
Total			**485.00**

Tabla 4: Resultados de costes de servicio.

Descripción	Total
Costes de Hardware	5,860.00
Costes de Servicios	485.00
Gastos generales	**6,345.00**

Tabla 5: Resultados de costes totales de materiales.

7.3 Gastos generales

Descripción	Total
Costes de mano de obra	16,280.00
Costes de materiales	6,345.00
Gastos generales	**22,625.00**

Tabla 6: Resultados de gastos generales.

8. Bibliografía

[1] S. Rodríguez, «LA GESTIÓN DE RESIDUOS SÓLIDOS URBANOS EN LA CIUDAD HISTÓRICA Y SOSTENIBLE: EL EJEMPLO DE ANDALUCÍA».

[2] D. Alonso, J. Á. Pastor, P. Sánchez, B. Álvarez, y C. Vicente-Chicote, «Generación Automática de Software para Sistemas de Tiempo Real: Un Enfoque basado en Componentes, Modelos y Frameworks», *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 9, n.º 2, pp. 170-181, abr. 2012, doi: 10.1016/j.riai.2012.02.010.

[3] M. Fayad y D. Schmidt, «Object-Oriented Application Frameworks», *Communications of the ACM*, vol. 40, oct. 1997, doi: 10.1145/262793.262798.

[4] martinekuan, «Estilo de arquitectura de microservicios - Azure Architecture Center». https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices (accedido 11 de septiembre de 2023).

[5] J. J. Valdivia-Caballero, «Modelo de procesos para el desarrollo del front-end de aplicaciones web», *Interfases*, n.º 009, Art. n.º 009, mar. 2016, doi: 10.26439/interfases2016.n009.1245.

[6] «Acerca de Projects - Documentación de GitHub», *GitHub Docs*. https://ghdocs-prod.azurewebsites.net/es/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects (accedido 12 de septiembre de 2023).

[7] J. M. Alarcón, «TypeScript contra JavaScript: ¿cuál deberías utilizar?», *campusMVP.es*, 20 de febrero de 2020. https://www.campusmvp.es/recursos/post/typescript-contra-javascript-cual-deberias-utilizar.aspx (accedido 19 de septiembre de 2023).

[8] «Overview - Material UI». https://mui.com/material-ui/getting-started/ (accedido 12 de septiembre de 2023).

[9] «¿Qué es Java Spring Boot? Introducción a Spring Boot | Microsoft Azure». https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-java-spring-boot (accedido 13 de septiembre de 2023).

[10] «Postman API Platform | Sign Up for Free», *Postman*. https://www.postman.com (accedido 14 de septiembre de 2023).

[11] «React». https://es.react.dev/ (accedido 14 de septiembre de 2023).

[12] F. Zammetti, *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker*. Berkeley, CA: Apress, 2020. doi: 10.1007/978-1-4842-5738-8.

[13] «Overview - Material UI». https://mui.com/material-ui/getting-started/ (accedido 14 de septiembre de 2023).

[14] «(11) Manejo y validación de formularios con Formik y Yup. | LinkedIn». https://www.linkedin.com/pulse/manejo-y-validaci%C3%B3n-de-formularios-con-formik-yup-vanegas-p-/?originalSubdomain=es (accedido 14 de septiembre de 2023).

[15] «react-data-table-component», *npm*, 24 de agosto de 2023. https://www.npmjs.com/package/react-data-table-component (accedido 15 de septiembre de 2023).

9. Anexo 1: Manual de Usuario

Este manual de usuario proporciona una descripción exhaustiva y fácil de entender sobre cómo utilizar la aplicación para aprovechar al máximo todas sus características y capacidades.

9.1 Usuarios

No hay usuarios activos en la aplicación ni se han implementado funciones para gestionar la autenticación de usuarios. Esta no fue una sección necesaria en los objetivos originales del proyecto. Sin embargo, hemos considerado la posibilidad de incorporar la gestión de sesiones de usuarios y, como resultado, se crea una página principal que simula el proceso de inicio de sesión en la aplicación.

9.2 Página principal (Home)

Cuando ejecutas la aplicación en tu navegador, la cual estará funcionando en http://localhost:3000, verás una simulación de inicio de sesión como página principal. Al hacer clic en el botón ‘Sign In’, ingresaras al panel principal de administración o Dashboard. Si, por el contrario, deseas dirigirte a la página de Envac, haz clic en ‘Go to Envac’.

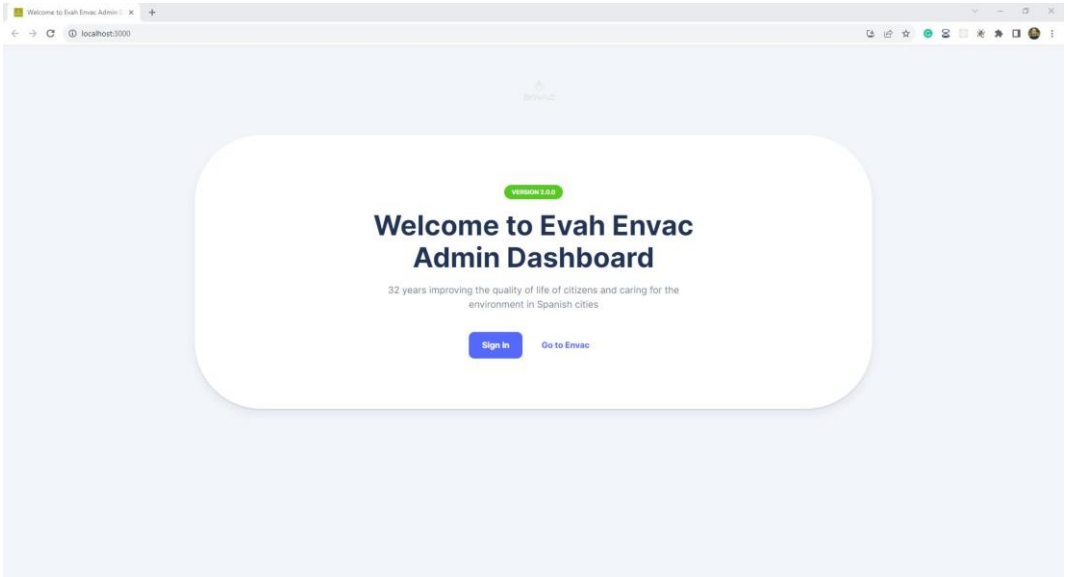


Figura 54: Vista web - simulación de inicio de sesión como página principal

9.3 Panel principal (Dashboard)

En esta página de inicio, se proporciona una vista resumida y visualmente organizada de datos clave y funciones relevantes. Esta información se presenta en forma de tarjetas para facilitar el acceso rápido a las diferentes áreas de la aplicación.

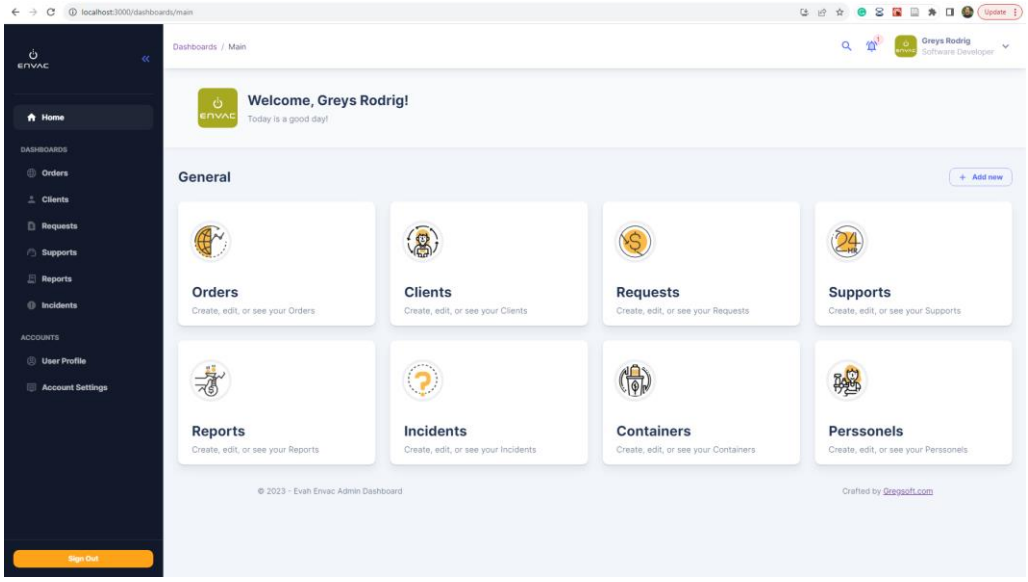


Figura 55: Vista web - acceso al Panel Principal (Dashboard).

9.4 Mapa del sitio

Con el propósito de facilitar la navegación y proporcionar una visión general del contenido de la aplicación, a continuación, podrás identificar de manera rápida la lista de páginas disponibles.

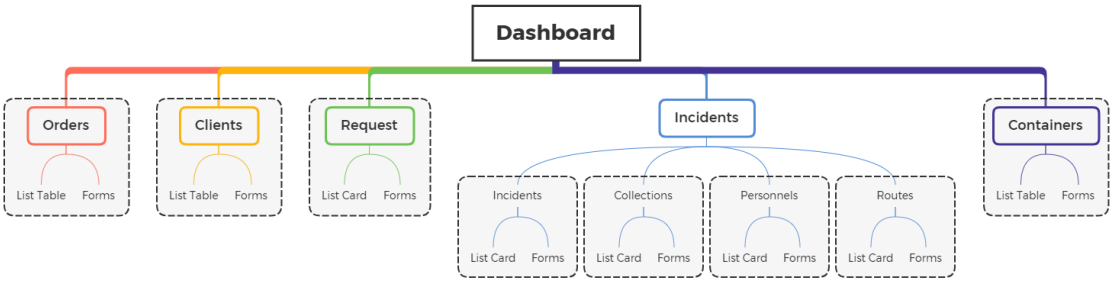


Figura 56: Mapa de sitio web

para comprender mejor el contenido de este mapa, enfoquémonos en identificar la sección llamada ‘Incidents’, que contiene otras subsecciones y podría ser la que genere más confusión.

9.5 Sección Incidencias

En esta sección de incidencias, encontrarás diversas subsecciones que puedes explorar a través de las pestañas ubicadas en su lado derecho. Estas pestañas incluyen ‘Collections’, ‘Personnels’ y ‘Rutas’. Todas estas vistas mantienen el mismo diseño y están relacionas con el manejo de incidencias, con la única diferencia siendo el color de los botones y gestiones de filtrados.

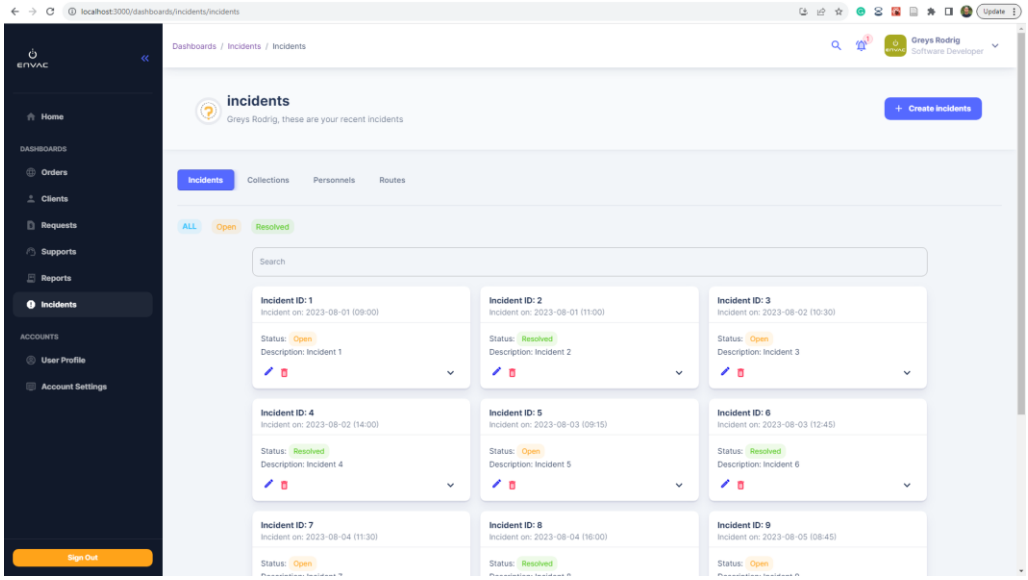


Figura 57: Mapa del sitio - sección de incidencias

9.6 Sesión Órdenes

Esta es la primera vista en nuestra lista, donde podrás ver todas las órdenes registradas y aplicar filtros según su estado.

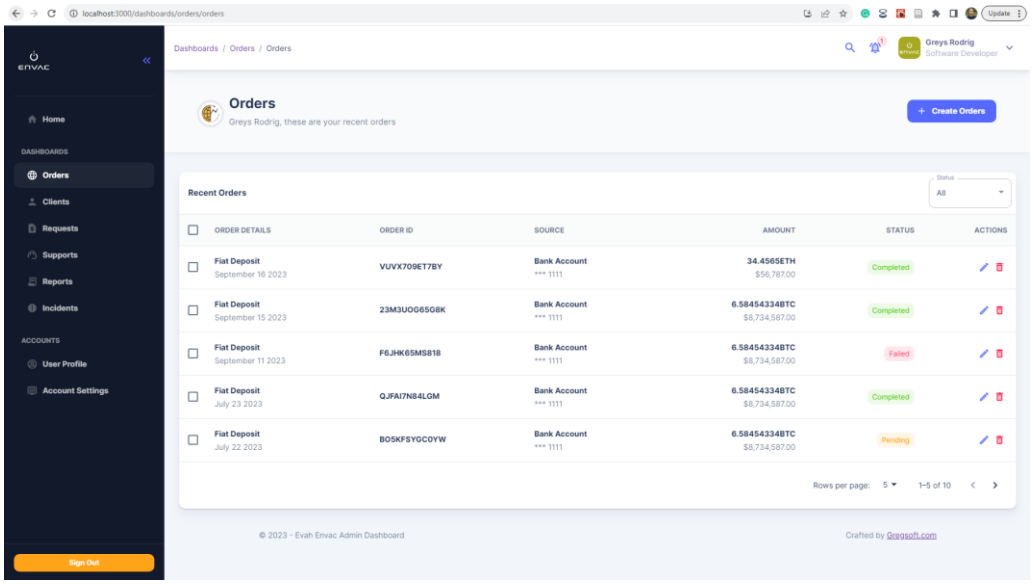


Figura 58: Mapa del sitio - sección de órdenes

9.7 Sesión Clientes

Esta sesión permite visualizar todos los clientes registrados en la aplicación, brindando la posibilidad de acceder, actualizar y eliminar sus datos.

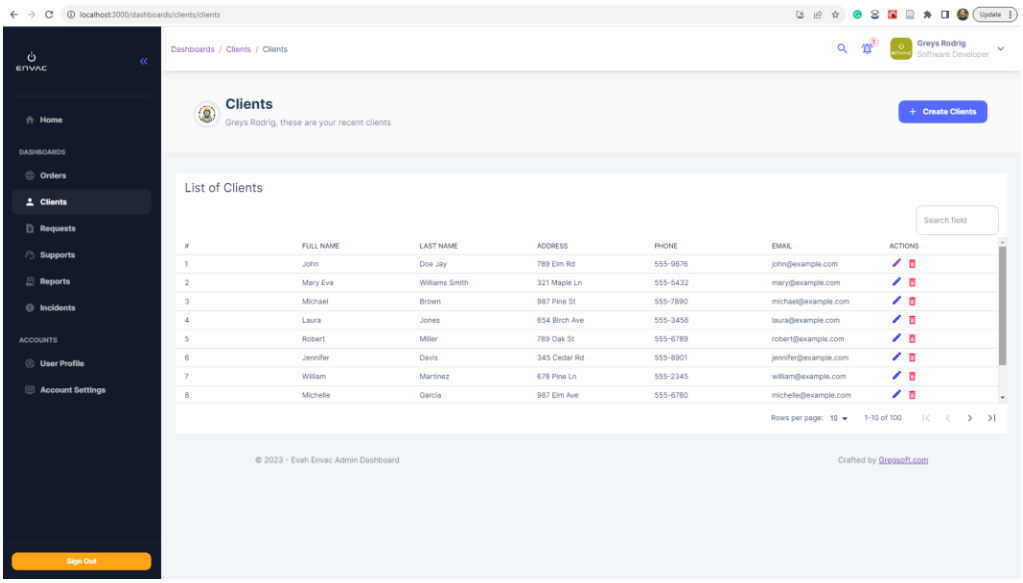


Figura 59: Mapa del sitio - sección de clientes

Para asegurar un reconocimiento rápido y uniforme, hemos desarrollado una interfaz de usuario similar para todas nuestras páginas, basándonos en las tres anteriores. A continuación, presentamos las siguientes páginas que se visualizan cuando se detecta un error 404, 500 u otros problemas similares.

9.8 Páginas de estados

Al presentarse algún problema de estado en la aplicación podrás visualizar estas páginas.

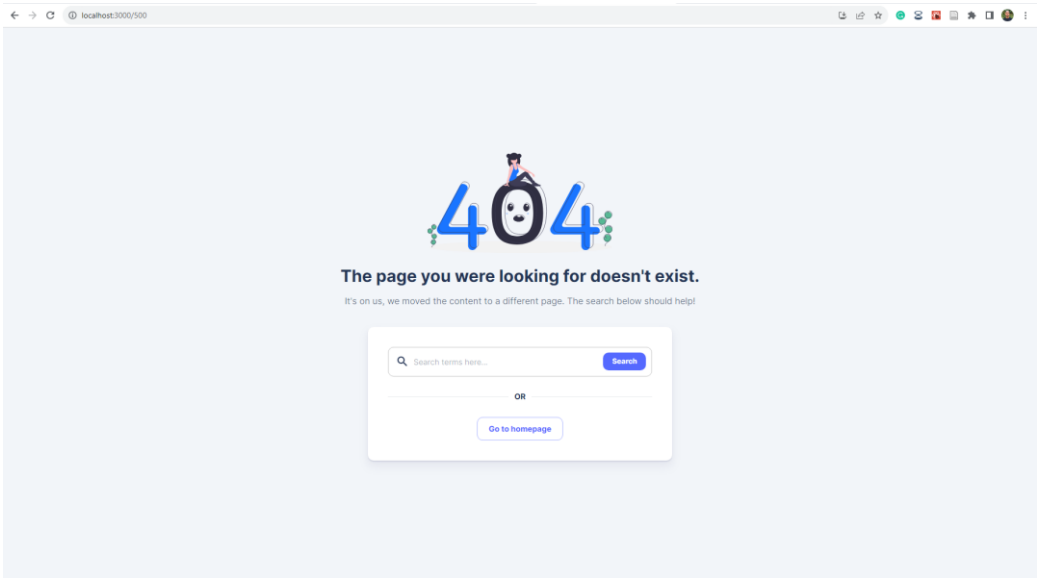


Figura 60: Vista web - página de error 404

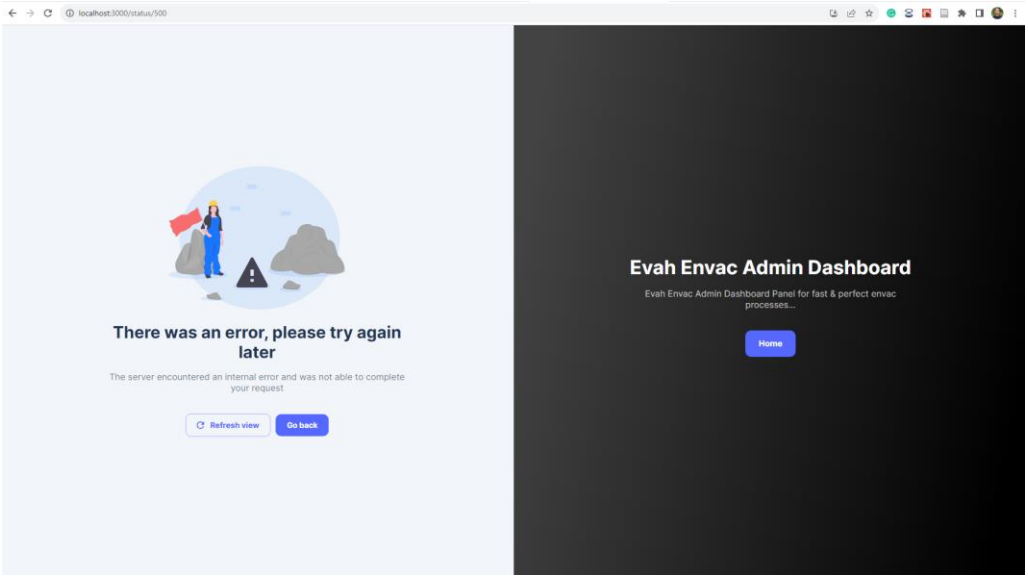


Figura 61: Vista web - página de estado 500

10. Anexo 2: Manual de Despliegue

El manual de despliegue abarca tanto el frontend como el backend de la aplicación, proporciona una guía detalla para lograr un despliegue exitoso de ambas capas del software en un entorno de producción.

10.1 Ejecución o Deploy del Backend

A continuación, se presenta con gran detalle el proceso de implementación del núcleo de la aplicación. Aquí muestra instrucciones concisas y precisas para configurar el backend de manera adecuada en un entorno de producción.

La meta principal con esto es simplificar todo el proceso de despliegue, garantizando que todas las funcionalidades del backend estén plenamente operativas y listas para ser utilizadas. Para lograr este objetivo, es recomendado seguir los siguientes pasos:

10.1.1 Clonación del proyecto

El primer paso crucial es clonar el repositorio de GitHub para obtener una copia completa del código fuente y las dependencias necesarias del backend de este proyecto, preparándonos para un despliegue exitoso. Este proyecto consta de dos microservicios y un servidor Eureka. Antes de comenzar, asegúrate de tener Git instalado en tu computadora y, luego, abre la terminal o línea de comandos y ejecuta el siguiente comando:

10.1.1.1 Eureka Server

```
https://github.com/gregrodrig/eurekaServer.git
```

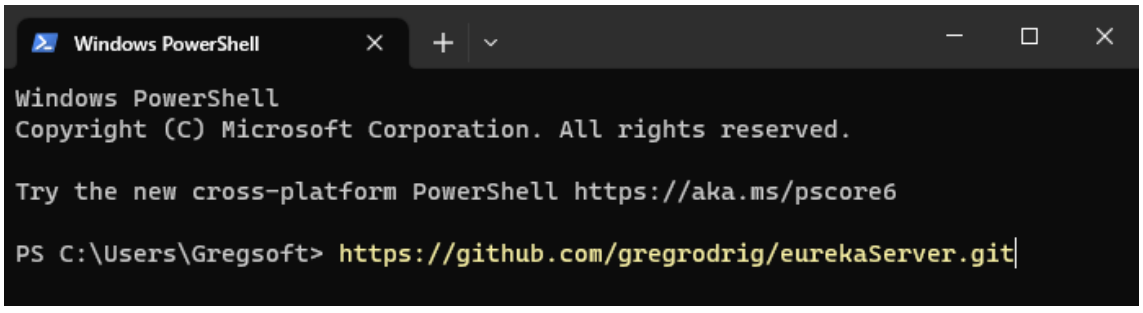


Figura 62: Terminal de comandos – Clonando proyecto Eureka Server.

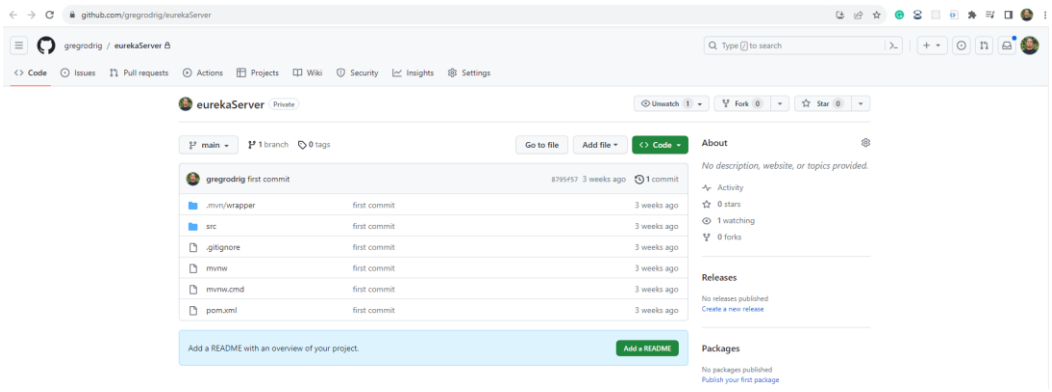


Figura 63: Vista sobre repositorio GitHub de proyecto Eureka Server.

10.1.1.2 Microservicio envahClients

https://github.com/gregrodrig/uahEvahClients.git

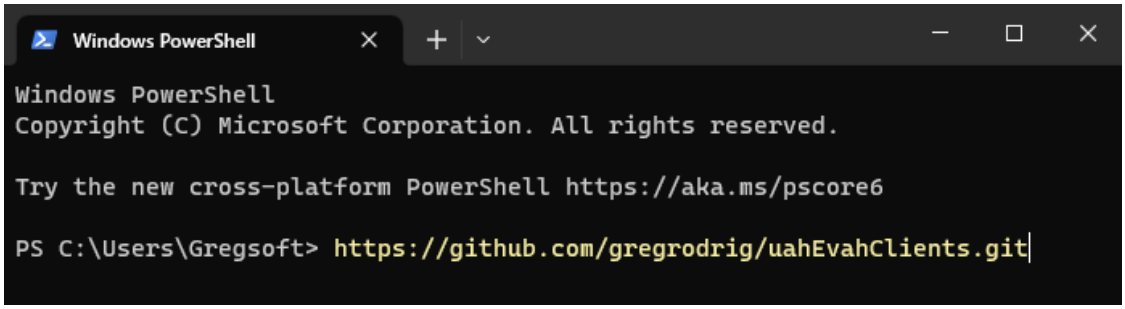


Figura 64: Terminal de comandos – Clonando proyecto envahClients.

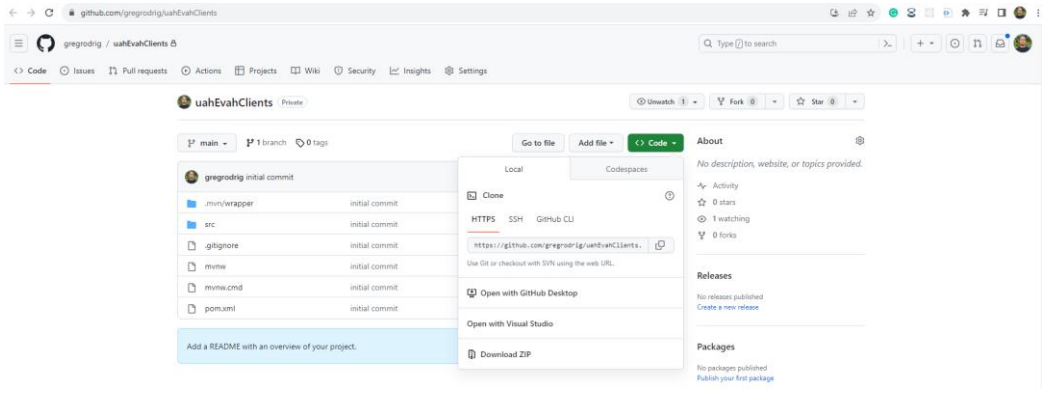


Figura 65: Vista sobre repositorio GitHub de proyecto envahClients.

10.1.1.3 Microservicio envahCollections

https://github.com/gregrodrig/uahEvahCollections.git

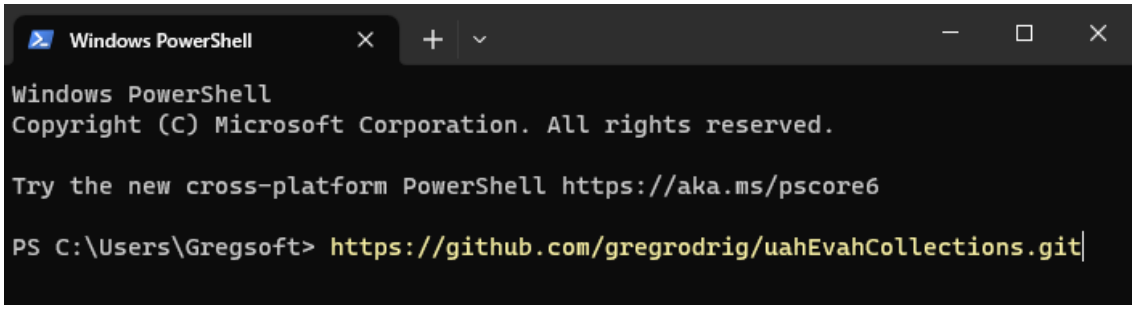


Figura 66: Terminal de comandos – Clonando proyecto envahCollections.

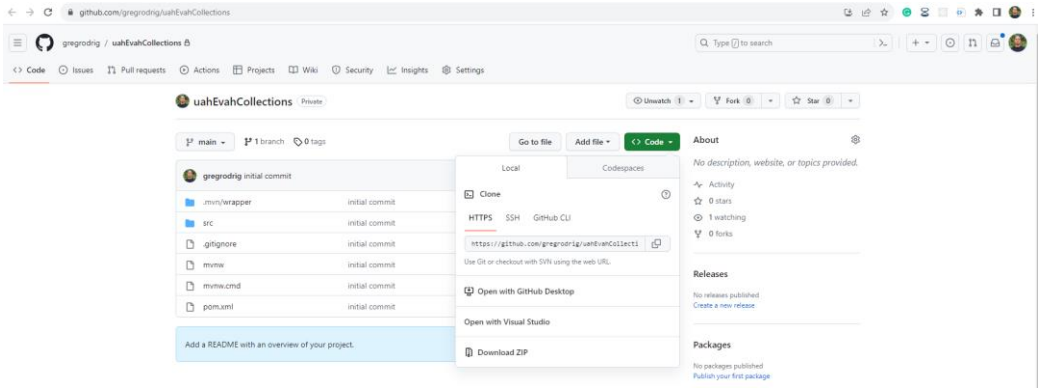


Figura 67: Vista sobre repositorio GitHub de proyecto envahCollections.

10.2 Creación de la base de datos

El segundo paso crucial en el proceso de implementación del backend es la creación de la base de datos en MySQL. Esta base de datos será el repositorio de todos los datos esenciales necesarios para el correcto funcionamiento de la aplicación. Para llevar a cabo este paso, haremos uso del comando 'CREATE SCHEMA' para establecer un nuevo esquema o 'schema' que funcionara como la estructura lógica central de nuestra base de datos.

Es fundamental asegurarse de contar con acceso a un servidor de bases de datos MySQL y disponer de un usuario con los permisos adecuados para crear bases de datos. Luego, procederemos a abrir una terminal o cliente de MySQL y ejecutaremos el siguiente comando:

CREATE SCHEMA envahdb_clients_containers_orders_requests

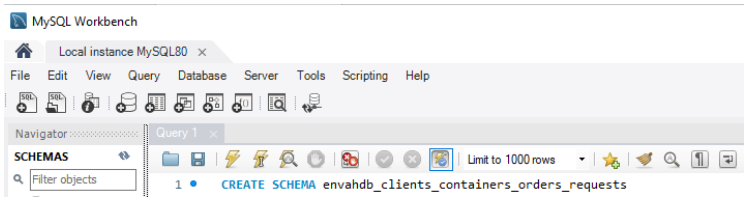


Figura 68: Vista sobre creación de la base de datos para proyecto envahClients.

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
```

```
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DAT
E,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema envahdb_clients_containers_orders_requests
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema envahdb_clients_containers_orders_requests
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `envahdb_clients_containers_orders_requests`
DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `envahdb_clients_containers_orders_requests` ;
-- -----------------------------------------------------
-- Table `envahdb_clients_containers_orders_requests`.`clients`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_clients_containers_orders_requests`.`clients` (
  `ID_Client` INT NOT NULL AUTO_INCREMENT,
  `FirstName` VARCHAR(255) NULL DEFAULT NULL,
  `LastName` VARCHAR(255) NULL DEFAULT NULL,
  `Address` VARCHAR(255) NULL DEFAULT NULL,
  `Phone` VARCHAR(255) NULL DEFAULT NULL,
  `Email` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Client`))
ENGINE = InnoDB
AUTO_INCREMENT = 114
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_clients_containers_orders_requests`.`containers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_clients_containers_orders_requests`.`containers`
(
  `ID_Container` INT NOT NULL AUTO_INCREMENT,
  `Capacity` INT NULL DEFAULT NULL,
  `Type` VARCHAR(255) NULL DEFAULT NULL,
  `Status` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Container`))
```

```
ENGINE = InnoDB
AUTO_INCREMENT = 115
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_clients_containers_orders_requests`.`orders`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_clients_containers_orders_requests`.`orders` (
  `ID_Order` INT NOT NULL AUTO_INCREMENT,
  `OrderDateTime` DATETIME(6) NULL DEFAULT NULL,
  `Status` VARCHAR(255) NULL DEFAULT NULL,
  `ID_Client` INT NULL DEFAULT NULL,
  `ID_Personnel` INT NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Order`),
  INDEX `ID_Client` (`ID_Client` ASC) VISIBLE,
  CONSTRAINT `orders_ibfk_1`
    FOREIGN KEY (`ID_Client`)
    REFERENCES `envahdb_clients_containers_orders_requests`.`clients` (`ID_Client`))
ENGINE = InnoDB
AUTO_INCREMENT = 102
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_clients_containers_orders_requests`.`requests`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_clients_containers_orders_requests`.`requests` (
  `ID_Request` INT NOT NULL AUTO_INCREMENT,
  `RequestDateTime` DATETIME(6) NULL DEFAULT NULL,
  `WasteAmount` INT NULL DEFAULT NULL,
  `WasteType` VARCHAR(255) NULL DEFAULT NULL,
  `ID_Client` INT NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Request`),
  INDEX `ID_Client` (`ID_Client` ASC) VISIBLE,
  CONSTRAINT `requests_ibfk_1`
    FOREIGN KEY (`ID_Client`)
    REFERENCES `envahdb_clients_containers_orders_requests`.`clients` (`ID_Client`))
ENGINE = InnoDB
AUTO_INCREMENT = 102
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

```
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

CREATE SCHEMA envahdb_collection_routes

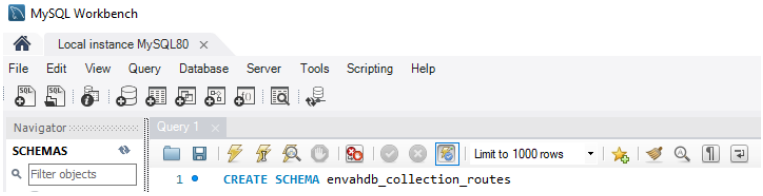


Figura 69: Vista sobre creación de la base de datos para proyecto envahCollection.

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DAT
E,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema envahdb_collection_routes
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema envahdb_collection_routes
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `envahdb_collection_routes` DEFAULT CHARACTER
SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `envahdb_collection_routes` ;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`routes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`routes` (
  `ID_Route` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(255) NULL DEFAULT NULL,
  `Description` VARCHAR(255) NULL DEFAULT NULL,
  `CollectionSchedule` VARCHAR(255) NULL DEFAULT NULL,
  `CoverageArea` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Route`))
ENGINE = InnoDB
AUTO_INCREMENT = 82
```

```
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`collections`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`collections` (
  `ID_Collection` INT NOT NULL AUTO_INCREMENT,
  `ID_Container` INT NULL DEFAULT NULL,
  `ID_Route` INT NULL DEFAULT NULL,
  `CollectionDateTime` DATETIME(6) NULL DEFAULT NULL,
  `WasteAmount` INT NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Collection`),
  CONSTRAINT `collections_ibfk_1`
    FOREIGN KEY (`ID_Route`)
    REFERENCES `envahdb_collection_routes`.`routes` (`ID_Route`))
ENGINE = InnoDB
AUTO_INCREMENT = 102
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `ID_Route` ON `envahdb_collection_routes`.`collections` (`ID_Route` ASC)
VISIBLE;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`personnel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`personnel` (
  `ID_Personnel` INT NOT NULL AUTO_INCREMENT,
  `FirstName` VARCHAR(255) NULL DEFAULT NULL,
  `LastName` VARCHAR(255) NULL DEFAULT NULL,
  `Position` VARCHAR(255) NULL DEFAULT NULL,
  `EmploymentStartDate` DATE NULL DEFAULT NULL,
  `Salary` DECIMAL(38,2) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Personnel`))
ENGINE = InnoDB
AUTO_INCREMENT = 101
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`incidents`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`incidents` (
  `ID_Incident` INT NOT NULL AUTO_INCREMENT,
```

```
  `Description` VARCHAR(255) NULL DEFAULT NULL,
  `IncidentDateTime` DATETIME(6) NULL DEFAULT NULL,
  `Status` VARCHAR(255) NULL DEFAULT NULL,
  `ID_Personnel` INT NULL DEFAULT NULL,
  `ID_Collection` INT NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Incident`),
  CONSTRAINT `incidents_ibfk_1`
    FOREIGN KEY (`ID_Personnel`)
    REFERENCES `envahdb_collection_routes`.`personnel` (`ID_Personnel`),
  CONSTRAINT `incidents_ibfk_2`
    FOREIGN KEY (`ID_Collection`)
    REFERENCES `envahdb_collection_routes`.`collections` (`ID_Collection`))
ENGINE = InnoDB
AUTO_INCREMENT = 103
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `ID_Personnel` ON `envahdb_collection_routes`.`incidents` (`ID_Personnel`
ASC) VISIBLE;

CREATE INDEX `ID_Collection` ON `envahdb_collection_routes`.`incidents`
(`ID_Collection` ASC) VISIBLE;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`maintenance`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`maintenance` (
  `ID_Maintenance` INT NOT NULL AUTO_INCREMENT,
  `ID_Container` INT NULL DEFAULT NULL,
  `MaintenanceStart` DATETIME(6) NULL DEFAULT NULL,
  `MaintenanceEnd` DATETIME(6) NULL DEFAULT NULL,
  `Description` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Maintenance`))
ENGINE = InnoDB
AUTO_INCREMENT = 102
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `envahdb_collection_routes`.`reports`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `envahdb_collection_routes`.`reports` (
  `ID_Report` INT NOT NULL AUTO_INCREMENT,
  `ID_Collection` INT NULL DEFAULT NULL,
```

```
  `ID_Personnel` INT NULL DEFAULT NULL,
  `Description` VARCHAR(255) NULL DEFAULT NULL,
  `ReportDateTime` DATETIME(6) NULL DEFAULT NULL,
  PRIMARY KEY (`ID_Report`),
  CONSTRAINT `reports_ibfk_1`
    FOREIGN KEY (`ID_Collection`)
    REFERENCES `envahdb_collection_routes`.`collections` (`ID_Collection`),
  CONSTRAINT `reports_ibfk_2`
    FOREIGN KEY (`ID_Personnel`)
    REFERENCES `envahdb_collection_routes`.`personnel` (`ID_Personnel`))
ENGINE = InnoDB
AUTO_INCREMENT = 101
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `ID_Collection` ON `envahdb_collection_routes`.`reports` (`ID_Collection`
ASC) VISIBLE;

CREATE INDEX `ID_Personnel` ON `envahdb_collection_routes`.`reports` (`ID_Personnel`
ASC) VISIBLE;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Mediante la ejecución de estas acciones, habremos establecido las bases de datos con sus respectivas tablas para almacenar los datos de ambos microservicios, lo que garantiza el correcto funcionamiento de la aplicación.

10.3 Ejecución del Backend

Una vez que hayamos clonado nuestros proyectos, procederemos a identificar las carpetas y seleccionaremos el IDE de nuestra preferencia. En nuestro caso, hemos optado por utilizar IntelliJ IDEA. En primer lugar, inicializamos Eureka Server, y luego procedemos con nuestros microservicios ‘envahClients’ y ‘envahCollection’.

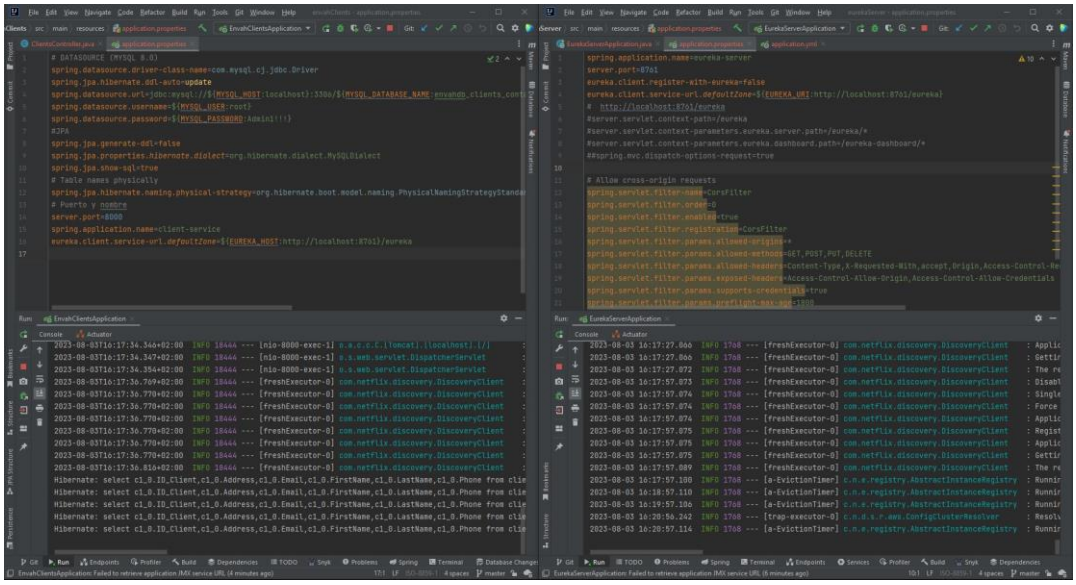


Figura 70: Vista sobre Eureka Server y microservicio envahClient en ejecución.

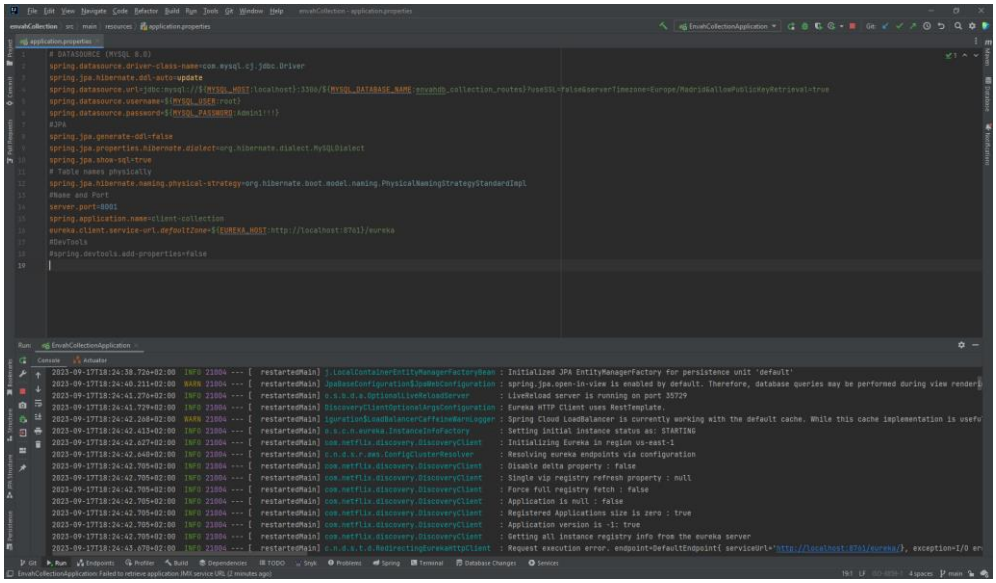


Figura 71: Vista sobre microservicio envahCollection en ejecución.

10.4 Ejecución o Deploy del Frontend

A continuación, se muestra un detallado proceso para poner en marcha la interfaz de usuario del proyecto. El Frontend de la aplicación ENVAH presenta la cara pública a los usuarios, brindando la oportunidad de interactuar de manera intuitiva y atractiva con la información.

10.4.1 Clonación del proyecto Frontal

El primer paso crucial es clonar el repositorio de GitHub para obtener una copia completa del código fuente y las dependencias necesarias del frontend de este proyecto, preparándonos para un despliegue exitoso. Este proyecto consta de los archivos necesarios para poner en funcionamiento el diseño UI. Antes de comenzar, asegúrate de tener Git instalado en tu computadora y, luego, abre la terminal o línea de comandos y ejecuta el siguiente comando:

```
https://github.com/gregrodrig/uahEvah.git
```

Esto descargará el código fuente del proyecto en una carpeta local nombrada ‘uahEvah’.

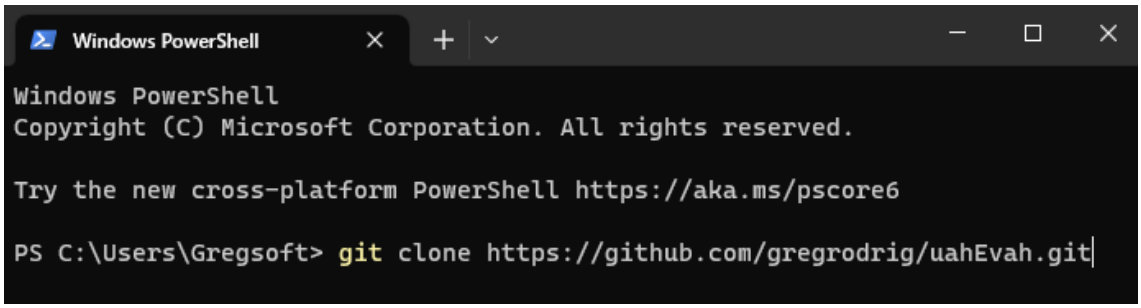


Figura 72: Terminal de comandos – Clonando proyecto frontal uahEnvah.

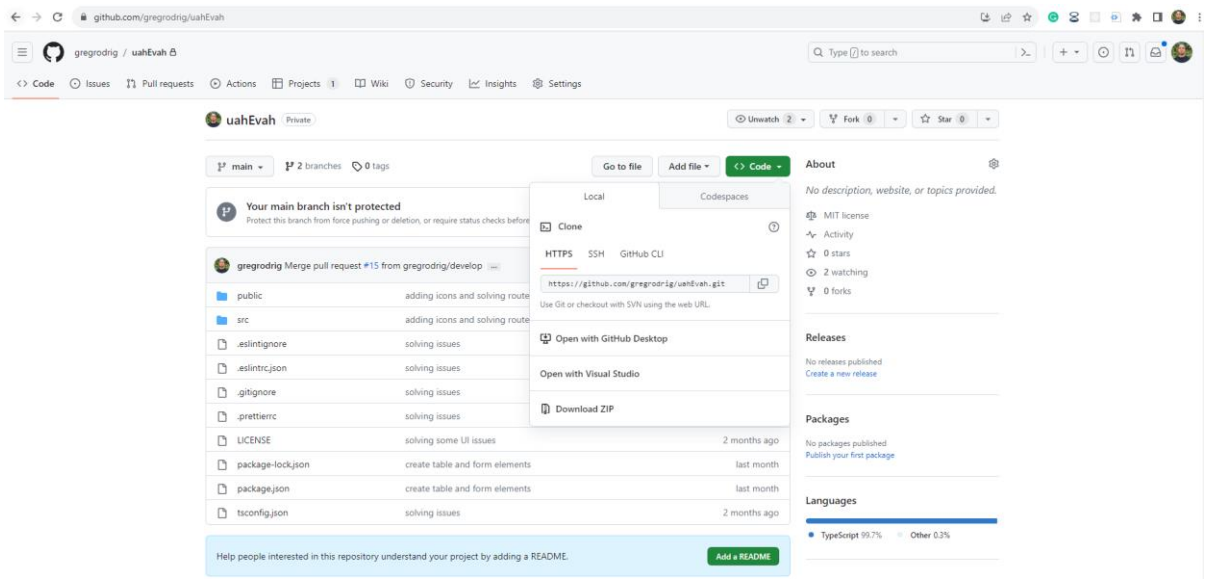


Figura 73: Vista sobre repositorio GitHub de proyecto frontal uahEnvah.

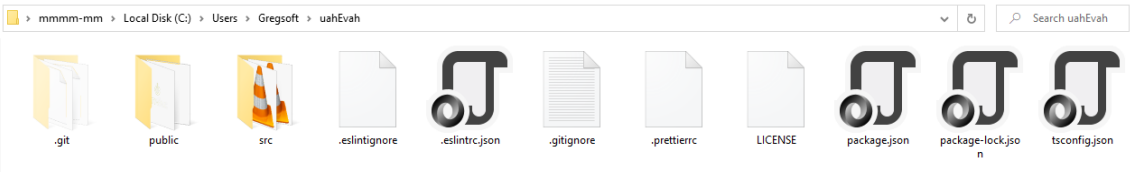


Figura 74: Vista sobre carpeta creado al clonar proyecto frontal uahEvah desde GitHub.

10.4.2 Instalación de dependencias

Una vez clonado el proyecto, es importante instalar todas las dependencias necesarias para que la aplicación funcione sin problemas. Para hacer esto posible, realiza los siguientes pasos:

- Abre tu IDE de preferencias y selecciona la opción para importar el proyecto 'uahEvah'.
- Luego, accede al terminal dentro del entorno de desarrollo.
- Ejecuta el comando 'npm install' tal como se muestra en la siguiente imagen.

Este proceso asegurará que todas las dependencias requeridas se instalen correctamente y la aplicación esté lista para funcionar.

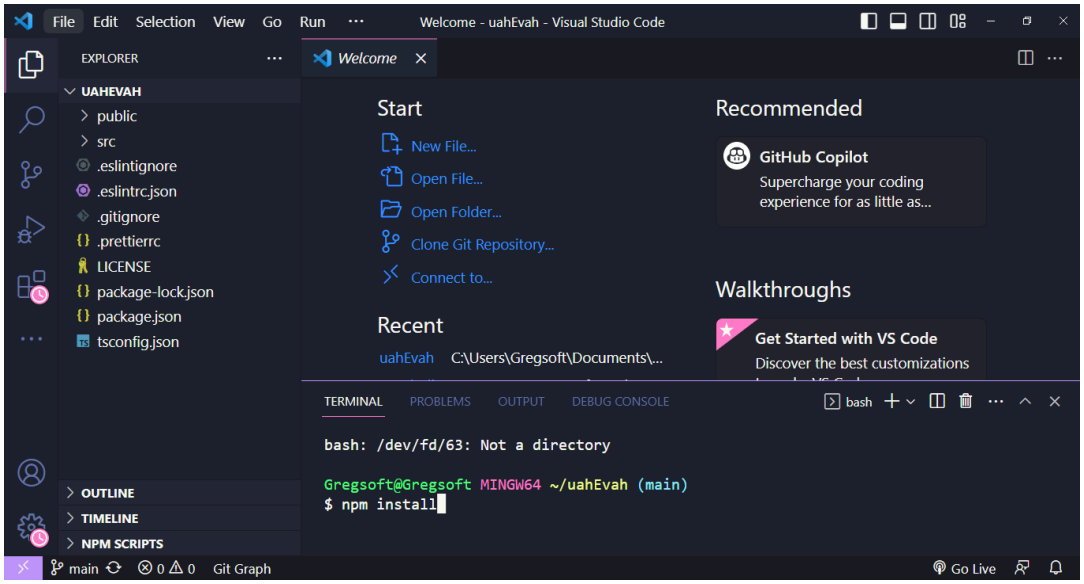


Figura 75: Vista sobre instalación de dependencias mediante el IDE Visual Studio Code.

10.4.3 Ejecución en el Navegador

Con todas las dependencias instaladas y el backend en ejecución, estamos preparados para dar inicio al servidor de desarrollo y abrir la aplicación en el navegador. Para lograrlo, simplemente emplearemos el comando 'npm start'. Este comando ejecutará el servidor de desarrollo y abrirá la aplicación web desarrollada en React.js en la dirección local http://localhost:3000.

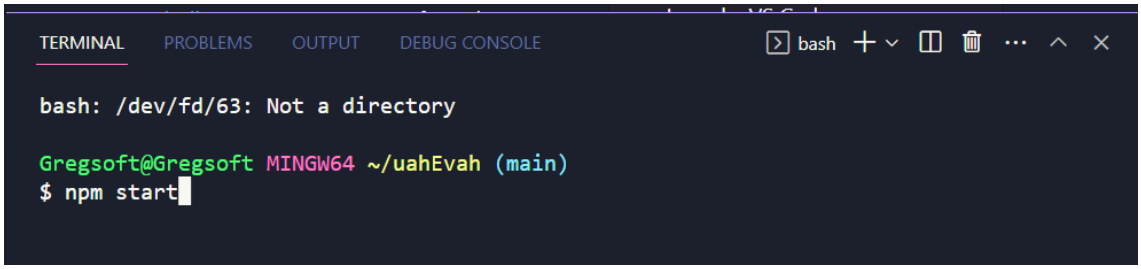


Figura 76: Vista sobre inicialización del proyecto frontal uahEvah mediante el IDE Visual Studio Code.

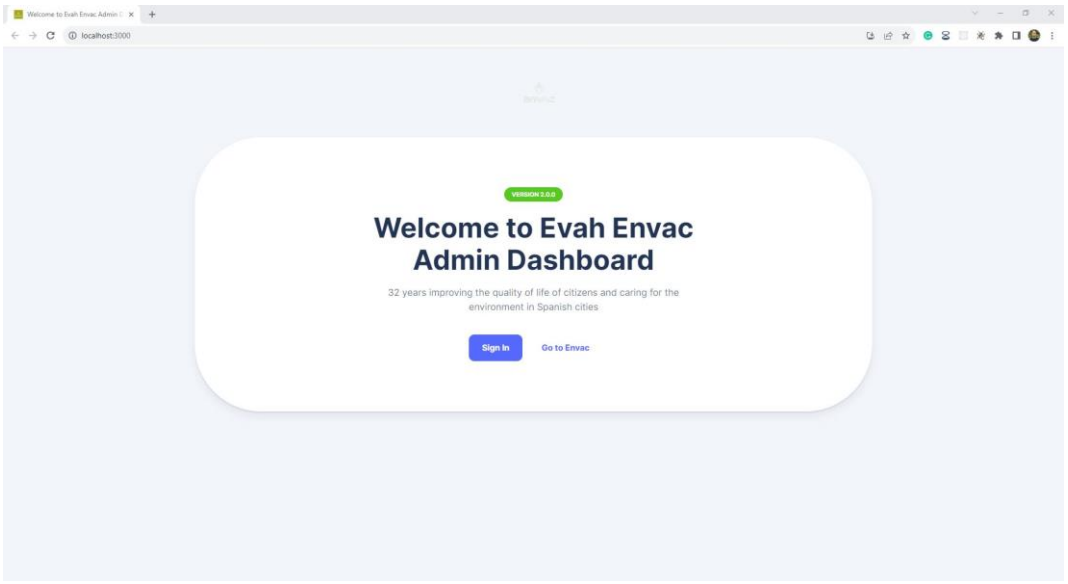


Figura 77: Vista web sobre aplicación web ejecutada en el navegador.

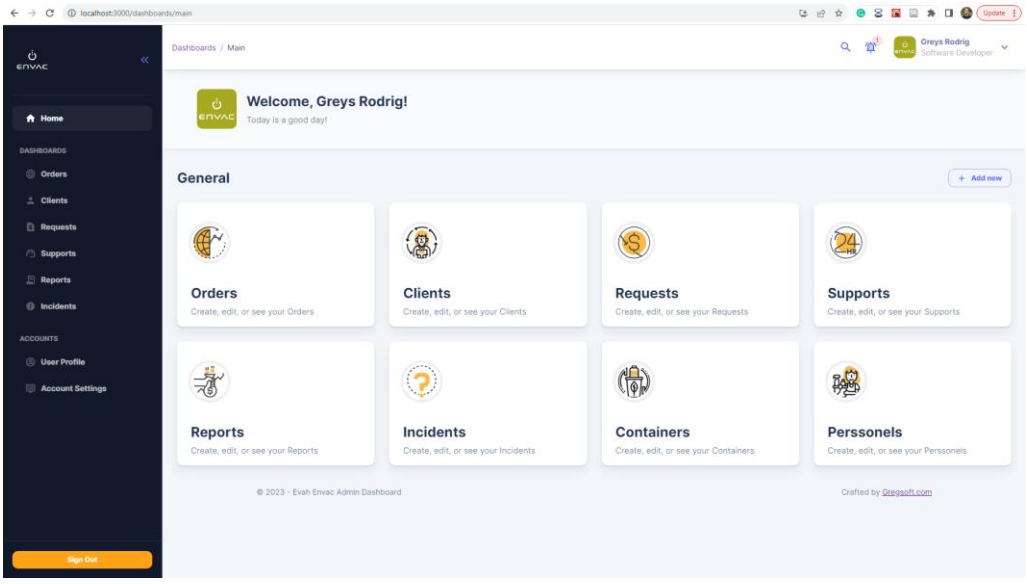


Figura 78: Vista web sobre acceso al Panel Principal (Dashboard).

Universidad de Alcalá