Pep Carmona Coll

Linkia FP | Dam m13 – Proyecto final

RestROOM

aplicación web de gestión de pedidos en restauración

Índice de contenido

[INTRODUCCIÓN 2](#_Toc65141986)

[{ MOTIVATION } 2](#_Toc65141987)

[{ OBJETIVOS } 3](#_Toc65141988)

[{ HERRAMIENTAS Y MATERIALES } 3](#_Toc65141989)

[< ENTORNO DE TRABAJO > 3](#_Toc65141990)

[< BASE DE DATOS > 4](#_Toc65141991)

[< API BACKEND > 4](#_Toc65141992)

[< REACTIVE FRONTEND > 5](#_Toc65141993)

[< OTROS > 5](#_Toc65141994)

[{ TECNOLOGÍAS } 6](#_Toc65141995)

[< ALMACENAMIENTO DE DATOS > 6](#_Toc65141996)

[< TRANSMISIÓN DE DATOS > 6](#_Toc65141997)

[< REPRESENACIÓN DE DATOS > 6](#_Toc65141998)

[DESARROLLO 8](#_Toc65141999)

[{ PLANIFICACIÓN } 8](#_Toc65142000)

[{ BASE DE DATOS } 9](#_Toc65142001)

[< ANÁLISIS > 9](#_Toc65142002)

[< DESARROLLO > 9](#_Toc65142003)

[{ API BACKEND } 10](#_Toc65142004)

[< ANÁLISIS > 10](#_Toc65142005)

[< DESARROLLO > 12](#_Toc65142006)

[< CONCLUSIÓN > 15](#_Toc65142007)

[{ REACTIVE FRONTEND } 15](#_Toc65142008)

[< ANÁLISIS > 15](#_Toc65142009)

[< DESARROLLO > 17](#_Toc65142010)

[{ DESPLIEGUE } 19](#_Toc65142011)

[CONCLUSIÓN 20](#_Toc65142012)

[{ SATISFACCIÓN } 20](#_Toc65142013)

[{ CONCEPTOS ADQUIRIDOS } 20](#_Toc65142014)

[{ FUNCIONALIDADES PENDIENTES } 21](#_Toc65142015)

[{ LÍNEAS FUTURAS } 21](#_Toc65142016)

[BIBLIOGRAFÍA 23](#_Toc65142017)

[ANEXOS 24](#_Toc65142018)

# INTRODUCCIÓN

Mi nombre es Pep Carmona Coll, soy un estudiante de 25 años cursando actualmente el último año del CFGS en Desarrollo de Aplicaciones Multiplataforma y empezando un segundo CFGS complementario en Desarrollo de Aplicaciones Web, para compatibilizar y completar un currículum enfocado generalmente al desarrollo de APIs y tecnologías web.

A lo largo de este curso he podido adquirir conocimientos en profundidad sobre el comportamiento de sistemas informáticos, funcionamiento de protocolos web y desarrollo de software responsive, adaptado a las necesidades actuales de los usuarios. He podido, además, observar y depurar diferentes estilos de programación, adquiriendo y aplicando algunas de las buenas prácticas más esenciales en el desarrollo software para crear un código funcional, eficiente, y comprensible.

El proyecto presentado en este documento trata de plasmar los conceptos obtenidos a través de la teoría, los ejercicios prácticos, los documentos académicos y los consejos de los profesores de las diferentes asignaturas cursadas, para desarrollar una aplicación web de principio a fin.

## { MOTIVATION }

The first moment I started noticing what programming was, and the wonderful things which could be done when coding, I decided I wanted to take part in this sector. Specifically, the situation we are living in since last year, due to the COVID-19 pandemic, made me realise even more how necessary and vital was for the whole world the communication. Thus, I thought about developing a project to try and help one of the most affected economic sectors of the moment, the restaurants.

That is how I came across the idea of RestROOM, a web platform that eases the communication between restaurant owners and employees with their customers. It is a project that, in addition, adds one more layer of complexity to the projects I was used to develop during the course practical exercises. That encouraged me to investigate and search for new technologies, implementations and basically explore new ways of programming that I’ve never before heard of.

The main motivation, however, was related to achieving a real, practical example of what I was capable of doing through programming. All my previous experiences with coding were just studying theory related topics, developing prototyping software when applying these theoretical concepts, and reviewing other people’s code, but I never faced a problem like this. Developing a full-stack web application, including database management, backend APIs and logical data manipulation, and frontend data display and formatting is a whole new challenge for me.

## { OBJETIVOS }

­­­­

Al decidirme por el proyecto a realizar, la mayoría del esfuerzo ha ido dirigido al estudio de la documentación de nuevas tecnologías a aplicar. El principal objetivo desde el principio fue ampliar el conocimiento sobre las tecnologías y procesos necesarios para desarrollar una aplicación web. Este objetivo, a medida que ha ido avanzando el estudio y desarrollo del proyecto, ha ido definiéndose en objetivos más pequeños, asequibles y a más corto plazo.

El primero de esos objetivos a corto plazo fue afianzar los conocimientos en manejo de Bases de Datos Relacionales, investigar sobre diseños eficientes de bases de datos y aplicar esos conocimientos a un modelo Entidad-Relación, identificando las entidades necesarias para el funcionamiento de la aplicación y las relaciones entre ellas.

La siguiente meta ha sido entender el funcionamiento de la web, cómo se comparte la información a través de ella y cómo se pueden hacer públicos los datos almacenados para poder acceder a ellos desde cualquier lugar. Era importante para mí profundizar en la comprensión de cómo trabaja la web por dentro.

Finalmente, el punto que considero un pilar fundamental del proyecto ha sido el hecho de poder demostrar que soy capaz de desarrollar un proyecto con una utilidad real con el que ayudar a la gente. Una característica inherente a la informática, y más en la programación, es la necesidad de resolver problemas reales y aportar soluciones eficientes a problemas críticos. Para eso consideraba necesario abarcar un proyecto de mayores dimensiones a las que estaba acostumbrado y así forzarme a aprender y aportar todos los conocimientos necesarios.

## { HERRAMIENTAS Y MATERIALES }

Al inicio del proyecto tenía claras las herramientas básicas que utilizar con las tecnologías que ya conocía, pero otras herramientas eran desconocidas para mi hasta el momento.

De esta forma, he clasificado las herramientas que he ido seleccionando, conociendo y utilizando en tres grandes sectores que abarca el proyecto: el almacenamiento de datos, la presentación visual de la aplicación, y la lógica, extracción y manejo de los datos.

### < ENTORNO DE TRABAJO >

Antes de todo, considero necesario definir las herramientas que conforman el entorno de trabajo en el que se ha desarrollado la aplicación desde un inicio. Estas herramientas son:

* **Microsoft Windows 10 v 10.0**: sistema operativo sobre el que se han instalado y desarrollado las demás herramientas. He seleccionado este SO por su gran compatibilidad con la mayoría de software, y la facilidad de instalación y desarrollo que ofrece.
* **Java Platform, Enterprise Edition 8** **(Java Development Kit 11.0.9+7-LTS, Java Runtime Environment 11.0.9+7-LTS)**: plataforma de desarrollo que ofrece una solución globalizada al desarrollo de software. Es una plataforma que a través del lenguaje de programación Java facilita la distribución de software entre diferentes dispositivos al compilar el código escrito por el programador y posteriormente interpretarlo en cualquier dispositivo mediante la máquina virtual de java.

### < BASE DE DATOS >

Para el almacenamiento de datos, desde el diseño hasta la implementación, se han utilizado las siguientes herramientas:

* **Dia Diagram Editor 0.97.2**: software de dibujo y esquematizado de diagramas que ofrece elementos predefinidos de diagramas de flujo y diagramas de clases UML. Este software se ha utilizado en la etapa de diseño y planificación, ya que permite plasmar con facilidad las entidades que intervendrán en nuestro modelo de datos, así como sus relaciones.
* **MySQL Server 8.0.22**: servicio de manejo de bases de datos relacionales. Se han implementado todos los modelados de datos y relaciones utilizando este servicio, en el que además se han programado algunos disparadores para complementar la inserción manual de datos.
* **MySQL Workbench 8.0.22**: interfaz gráfica del servicio de base de datos de MySQL, que facilita la comunicación con la base de datos, sobre todo en el período de desarrollo e implementación de datos.

### < API BACKEND >

Una vez estructurados y definidos los datos a utilizar, éstos son gestionados a través de lenguaje Java. Para ello se han usado las siguientes herramientas:

* **Apache Netbeans IDE 12.1**: entorno de desarrollo para diferentes lenguajes de programación. Facilita la creación y el empaquetado de nuevos proyectos a través de una interfaz gráfica.
* **MySQL JDBC Driver**: este conector implementa la API de *Java DataBase Connectivity* para establecer una conexión entre la aplicación Java encargada de gestionar los datos y la base de datos MySQL encargada de almacenarlos y ejecutar operaciones sobre estos datos almacenados.
* **NB Spring Boot plugin 3.0**: extensión para integrar el servicio de configuración rápida del framework Spring en el IDE Netbeans (se explica en detalle en el siguiente apartado).
* **Spring Framework 5.16**: framework de Java que implementa el protocolo de Inversión de Control y reduce la cantidad de código necesario para ejecutar tareas más complejas en Java utilizando el sistema de anotaciones. Es un framework con gran variedad de módulos que implementan diferentes funcionalidades. En este proyecto se han utilizado los siguientes módulos:
  + Spring Data JPA: implementa la capa de acceso a los datos a través de la *Java Persistence Api* y un sistema de repositorios para realizar acciones CRUD sobre la base de datos.
  + Spring Web MVC: implementa la capa de acceso externo a la API a través de *endpoints*, que ejecutan una acción diferente dependiendo de la ruta de acceso y del método de petición HTTP.

### < REACTIVE FRONTEND >

Finalmente, se aplica un aspecto visual para facilitar la interacción con el cliente, haciendo uso de las siguientes herramientas:

* **Visual Studio Code 1.53.2**: editor de código flexible y versátil. Acepta multitud de extensiones para facilitar la edición de código fuente.
* **Vue.js Framework 3**: framework que aplica una capa de abstracción sobre el lenguaje JavaScript para facilitar la creación de aplicaciones web reactivas y páginas web dinámicas.
* **NUXT.js 2.14.8**: framework para la creación de web apps del lado de servidor que amplía las bases del framework Vue.js. Además, genera una configuración automática del proyecto para reducir el código necesario a la hora de configurar opciones avanzadas.

### < OTROS >

Además, para otras funcionalidades durante el desarrollo del proyecto se han utilizado las siguientes herramientas:

* **Postman**: herramienta utilizada para testear el comportamiento de la API del backend y comprobar las funcionalidades incorporadas previo a la incorporación de esta API al entorno de frontend.
* **Node.js 14.15.5**: framework de JavaScript que establece un entorno de desarrollo en tiempo de ejecución que proporciona una capa de servidor donde ejecutar el código necesario para que el cliente acceda a los datos.
* **GitHub Desktop 2.6.3**: aplicación de interfaz gráfica de GIT para controlar el flujo de trabajo y el seguimiento del progreso del proceso de desarrollo desde una interfaz más intuitiva, pero aplicando los mismos principios que el aplicativo de terminal de GIT.
* **Trello**: aplicación web utilizada para organizar y planificar tareas de manera sencilla. Distribuye cada tarea en un formato de tarjetas que permite reducir un problema mayor a tareas sencillas.

## { TECNOLOGÍAS }

Durante el desarrollo del proyecto han ido surgiendo necesidades técnicas que se han podido resolver gracias a la incorporación de diferentes tecnologías, para tratar de resolver paradigmas y problemas lógicos a la hora de tratar con los datos, comunicar el backend con el frontend, o modificar los datos en tiempo real. Las tecnologías principales utilizadas han sido las siguientes:

### < ALMACENAMIENTO DE DATOS >

* **OOP** (*Object Oriented Programming*): la *Programación Orientada a Objetos* es un paradigma que establece una metodología de desarrollo de código donde el código es una extrapolación directa de los objetos reales, definiendo así atributos y acciones propios de estos objetos.
* **ORM** (*Object-Relational Mapping*): el *Mapeo Objeto-Relacional* es el término utilizado para referirse a la técnica utilizada para referenciar un campo en una base de datos y convertirlo en un objeto.
* **CRUD** (*Create, Read, Update, Delete*): es un conjunto de funciones básicas que se pueden ejecutar sobre un objeto en la capa de acceso a datos.

### < TRANSMISIÓN DE DATOS >

* **API** (*Application Programming Interface*): una *Interfaz de Programación de Aplicaciones* es un conjunto de protocolos y acciones que se incorporan a un software para permitir y facilitar la comunicación entre aplicaciones. En nuestro caso, estas especificaciones han sido la base para establecer una conexión entre el módulo de backend y el de frontend.
* **REST** (*Representational State Transfer*): la *Transferencia de Estado Representacional* es una arquitectura de software que se utiliza para establecer una comunicación entre aplicaciones a través de métodos HTTP y transferir información entre ellas en forma de objetos, con un formato XML o JSON.
* **HTTP** (*HyperText Transfer Protocol*): el *Protocolo de Transferencia de HiperTexto* es un protocolo común en el ámbito web utilizado para establecer una serie de reglas básicas y usos comunes en la transferencia de información entre usuarios a través de internet. La estructura básica de una petición HTTP contiene:
  + un método, que indica la acción de la petición. i.e: *GET, POST, PUT, DELETE …*
  + una ruta, que indica el directorio o endpoint al que dirigir la petición.
  + un cuerpo, donde se sitúa la información que se quiere transferir.
  + un encabezado, que define información adicional sobre la petición.

### < REPRESENACIÓN DE DATOS >

* **SSR** (*Server Side Rendering*): normalmente el renderizado de la página se realiza del lado del cliente, pero con la metodología SSR el servidor es el encargado de recibir los datos de la API, procesarlos si es necesario, fabricar una presentación visual y transferir esta representación completa al cliente para que esté la muestre en el navegador.
* **fetch API**: esta API de JavaScript permite la comunicación de aplicaciones cliente con aplicaciones backend para solicitar datos a la API y poder introducirlos en los documentos HTML necesarios para mostrarlos.
* **HTML** (*HyperText Mark-up Language*): este lenguaje de marcas nos permite definir elementos a través de los cuales representar la información y el texto necesarios para presentar una aplicación web al cliente a través de un navegador.
* **CSS** (*Cascade Style Sheets*): las hojas de estilo permiten aportar al texto en el navegador una apariencia visual más atractiva y un cierto grado de interactividad.

# DESARROLLO

Este proyecto ha sido estructurado desde un primer momento en una organización por capas o niveles, para poder afrontar los problemas de la manera más aislada posible y así ir ampliando paso a paso las funcionalidades partiendo de que la capa anterior es ya una versión estable. Este modelo de desarrollo es una adaptación e implementación del modelo MVC al ámbito de DevOps.

De todas formas, cabe destacar que en este proyecto no se ha materializado el proceso de paso a producción de la aplicación. La última capa de desarrollo consistiría en la compilación del código y la publicación de la aplicación web en un servicio de hosting que permita la ejecución tanto del software backend como de la aplicación web cliente. Más adelante se detallan algunas especificaciones de este proceso.

Desde un punto de vista general, la planificación del proyecto ha sido la primera fase de desarrollo, y se ha utilizado con el fin de poder visualizar la idea global de la aplicación objetivo.

De la misma forma, también se ha realizado un seguimiento del progreso de cada fase, para poder visualizar los cambios realizados en cada versión, así como el proceso de integración de nuevas funcionalidades.

## { PLANIFICACIÓN }

La primera fase es una de las más importantes a la hora de definir cualquier proyecto, pues es la que permite identificar los objetivos a corto y medio plazo, así como las necesidades y requerimientos para las siguientes fases.

El primer paso para iniciar la planificación del proyecto fue definir el objetivo principal a conseguir, así como realizar una investigación en profundidad para analizar las tecnologías y metodologías más óptimas para materializar ese objetivo. En el anexo se adjuntan las referencias más significativas a los recursos online que han facilitado la definición de los procesos de desarrollo necesarios para esta aplicación web.

En esta fase de planificación se definieron en primera instancia las características principales que debería tener la aplicación final, organizándolas en dos grupos: características críticas para el funcionamiento y características adicionales. Esta división ha permitido a lo largo del desarrollo poder establecer un orden de prioridades en las tareas pendientes de realizar, para poder asegurar en cualquier caso que la aplicación final a día de la entrega dispondría de unas funcionalidades básicas para poder ponerse en funcionamiento.

Entre estas funcionalidades críticas se encontraban básicamente la API de backend al completo, que permitirá la conexión con la base de datos, la manipulación de los datos extraídos y el establecimiento de endpoints para la comunicación externa de la API. Además, se requería también como característica imprescindible una capa de presentación básica desde la cual el usuario general pudiera interactuar con la API de una manera gráfica e intuitiva.

Además, otro aspecto fundamental a definir en esta fase, antes de avanzar con la programación del proyecto, propiamente dicha, es la división del proyecto en fases que agrupen tareas específicas relacionadas entre sí, para así poder trabajar en una sección del proyecto de manera continuada y focalizarse en terminar esa sección antes de avanzar. Para este apartado ha sido necesario definir también la temporalización de las distintas fases, y así obtener una planificación orientativa en el tiempo que permitiera hacerse una idea del esfuerzo requerido para cada una de las fases.

En este aspecto, desde un punto de vista objetivo y global, y una vez finalizado el proyecto, considero que se podría haber dedicado más tiempo y esfuerzo en esta fase. De esta forma se habría podido reducir el tiempo dedicado a aspectos más intrascendentes y se habrían priorizado tareas más apremiantes y críticas.

## { BASE DE DATOS }

Una vez establecida la planificación del proyecto, se procedió con las fases propias de programación. En estas fases se empieza a introducir código y a dar forma a la aplicación web, desde el interior al exterior. Esto quiere decir que las fases de programación empiezan en la parte más interna de la aplicación, o la menos visible para el usuario final, pero que constituye la base - o *core* – imprescindible del proyecto. En nuestro caso, este *core* está formado por una Base de Datos donde almacenar la información necesaria para los procesos de interacción con la aplicación.

### < ANÁLISIS >

Tras el proceso de investigación sobre las posibilidades que había en el mercado de software gratuito referentes al almacenamiento de datos, las herramientas disponibles se organizaban en dos grandes grupos: Bases de Datos Relacionales o SQL frente a Bases de Datos NoSQL. Las principales diferencias entre ambas residen en el sistema que utilizan para organizar los datos almacenados.

Para este proyecto en particular se ha seleccionado una Base de Datos Relacional ya que estas permiten una organización de los datos en forma de tablas, así como permite también la relación de las tablas entre ellas. Este proyecto requería una Base de Datos donde cada tabla hiciera referencia a un objeto de la vida real, donde cada campo de la tabla fuera un atributo de este objeto, y que permitiera establecer relaciones entre las tablas para definir objetos contenedores y contenidos. Al finalizar el proceso de investigación se seleccionó el Sistema Gestor de Bases de Datos MySQL Server como herramienta para almacenar la información utilizando la estructura de datos necesaria.

### < DESARROLLO >

El primer paso una vez seleccionada la tecnología a utilizar, fue definir las unidades básicas necesarias para el funcionamiento de la aplicación. Por lo tanto, se procedió a realizar un diagrama básico de las entidades participantes en el proyecto, así como las relaciones entre ellas, en un formato de diagrama de flujo, para tener una idea general de cómo se establecerían las conexiones entre los objetos. Una vez definido este esquema básico se procedió a su concreción a través de un diagrama de clases, utilizando la convención de simbología UML, donde se especificaban los atributos necesarios para cada objeto, así como las claves primarias para identificar inequívocamente un objeto en concreto, y sus claves foráneas, utilizadas para referenciar otro objeto a través de una relación entre las tablas.

Este diagrama de clases Entidad-Relación fue posteriormente utilizado para materializar la estructura de la base de datos a través de programación SQL. Para ello se utilizó una combinación de interfaz gráfica y línea de comandos mediante el software MySQL Workbench, que además resultó ser tremendamente útil a la hora de visualizar y organizar las diferentes tablas que se iban creando a medida que se materializaba la estructura definida en el diagrama.

Al finalizar la definición de la estructura se procedió con un volcado de datos de prueba para poder realizar testeos en la integridad de la base de datos. En este aspecto, los testeos se realizaron de forma manual, ya que la estructura era relativamente sencilla y no requería de un proceso demasiado largo a la hora de comprobar la mayoría de funcionalidades de la Base de Datos. Aun así, desde un punto de vista crítico, se debería haber investigado sobre las posibilidades de desarrollar tests automáticos para poder comprobar la funcionalidad en el caso de escalar la aplicación para abarcar funcionalidades mayores.

## { API BACKEND }

La estructura de datos es la base para poder avanzar en la programación del proyecto ya que define la estructura y la organización de la información para que cualquier software pueda acceder y administrar los datos más adelante. Es por ello que, únicamente al tener la Base de Datos completamente definida y funcional, fue cuando se avanzó a la siguiente fase de programación, referente al desarrollo de una API que funcionaría de backend para el proyecto en general.

### < ANÁLISIS >

De la misma forma que en la fase de programación anterior, gran parte del tiempo y el esfuerzo fue dedicado al proceso de investigación y análisis de tecnologías. En este apartado, la cantidad de diferentes tecnologías disponibles para realizar tareas relativamente similares fue abrumador al principio, pero a medida que se profundizaba en los conocimientos técnicos relativos a esta área se fueron concretando las opciones.

En primera instancia, la elección más básica para el desarrollo de una API es identificar que protocolos seguir para llevar a cabo la transmisión de información entre las aplicaciones implicadas. En este aspecto, se identificaros dos principales arquitecturas de desarrollo: REST y SOAP. La principal diferencia entre ellas reside en el formato de la información que utilizan para comunicarse. El primer protocolo es más flexible en cuanto a los formatos disponibles, permitiendo transmitir información en XML, JSON, binario, HTML, etc. El segundo, por su parte, es más restrictivo y solo acepta el formato XML. Es por esta razón, principalmente, por lo que se ha seleccionado la arquitectura REST para desarrollar la estructura de la API.

La segunda decisión importante, toda vez que se da por supuesto que el desarrollo de backend se va a realizar con lenguaje Java, fue investigar sobre las posibilidades que ofrecía el lenguaje por sí mismo para realizar conexiones entre aplicaciones y transmisión de información entre ellas. De la misma forma, la investigación también incluía identificar qué frameworks eran más útiles y qué ventajas ofrecían respecto al uso del lenguaje sin frameworks.

La plataforma de Java Enterprise Edition es una distribución especializada enfocada en APIs y servidores de aplicaciones, por lo que incluye características necesarias y esenciales para el desarrollo de nuestro proyecto, como son las librerías de persistencia, de SQL y de transacciones, entre otros. Este es, por tanto, la base de las herramientas que se utilizarán para el desarrollo de esta fase.

Sobre la plataforma Java EE existen una gran variedad de frameworks que aportan una capa de abstracción sobre las funcionalidades que ofrece la distribución básica. En este proyecto, la API que se desarrollará seguirá una arquitectura REST, estará conectada a una base de datos relacional y deberá entregar unos endpoints que devolverán los recursos solicitados. Para todo ello existe un framework que cumple con la mayoría de las características necesarias: Spring Framework.

Spring es un framework muy extenso que proporciona una capa de abstracción sobre la mayoría de funcionalidades de la plataforma Java EE, utilizando un sistema de anotaciones para reducir el código escrito y programar de una manera más intuitiva. Al ser tan extenso, el framework en sí cuenta con diferentes módulos para utilizar solamente los necesarios para el proyecto en cuestión. En nuestro caso, y como hemos definido en el párrafo anterior, necesitamos cubrir unas funcionalidades muy concretas para las que utilizaremos únicamente los siguientes módulos de Spring:

* Spring Data JPA simplifica todo el proceso relativo a la extracción y gestión de datos, así como la conexión con la Base de Datos. Este módulo proporciona un sistema de interfaces llamadas *repositorios* que se encargan de establecer la relación entre las instrucciones de Java y las propias de MySQL. De esta forma, mediante un *Query Language* propio, es posible definir consultas e interacciones con la base de datos con mucho menos código, dejando al módulo encargarse de generar las instrucciones necesarias más tediosas, como crear una *EntityManagerFactory* o una *Session*.
* Spring Web MVC proporciona una colección de anotaciones muy útiles a la hora de establecer endpoints y administrar la interacción de la API con las aplicaciones externas. Las anotaciones más importantes, como *@GetMappint* o *@PostMapping* permiten asignar una respuesta a una petición HTTP con el método y la ruta definidos en la anotación. Es así como se harán públicos los datos que se extraen y manipulan de la base de datos.

Además, el framework de Spring cuenta con una implementación, Spring Boot, que funciona como una herramienta de autoconfiguración que se encarga de realizar una configuración básica de los módulos instalados que normalmente es funcional para los proyectos más básicos. Esta configuración básica puede ser modificada más adelante y así permitir una gran flexibilidad.

Finalmente, el último requerimiento para finalizar la fase de investigación de esta capa de desarrollo cabe mencionar la inclusión del software de gestión y administración de proyectos a utilizar. Este tipo de software se encarga de administrar los requisitos y dependencias del proyecto asignado. En el caso de este proyecto en cuestión, se ha seleccionado Maven como software de gestión de proyecto, ya que ofrece una estructura muy clara en formato xml y tiene una utilidad contrastada en infinidad de proyectos.

### < DESARROLLO >

Las fases del desarrollo del proyecto engloban multitud de tareas más pequeñas y que se encargan de objetivos básicos y esenciales. En la sección del desarrollo de la API se divide la tarea general en tareas más concretas a corto plazo, para poder realizar un seguimiento efectivo del avance del proyecto.

De esta forma, el primer paso para empezar con la programación de la API es definir y estructurar el proyecto. Para ello se ha utilizado el ya mencionado Spring Boot, que ofrece un plugin para el IDE de NetBeans, lo cual permite una interacción e integración entre ambos. Es así como, desde el mismo IDE se procede a la creación de un proyecto nuevo del tipo Spring Boot Initializr. Esta acción abre una nueva pestaña donde se ejecuta la funcionalidad Initializr, que es usado como guía para la inicialización y configuración rápida de un nuevo proyecto.

#### - INICIALIZACIÓN -

Es el mismo IDE, a través del plugin de Spring Boot, el que guía al usuario a través de la configuración del nuevo proyecto, pasando por diferentes pasos. Entre ellos, se define el nombre del proyecto, el package donde se almacena, la definición del artifact de Maven, y el paso más importante, la configuración de dependencias, donde se seleccionaron en el caso de este proyecto los módulos de Spring anteriormente mencionados, además del conector JDBC para MySQL. Una vez finalizada la configuración, se creará el directorio de carpetas del proyecto, y las dependencias seleccionadas se introducirán automáticamente en el archivo *pom.xml* de Maven.

Una vez inicializado, se definen las carpetas necesarias para estructurar el código de la API de una manera legible y entendible. Así pues, siguiendo los principios SOLID y un modelo basado en la arquitectura MVC, se crean las carpetas principales:

* La carpeta *data* almacenará las clases POJO, que definen la estructura de los objetos donde serán mapeadas las entidades de la base de datos, y será el equivalente a la capa de Modelo en el proyecto.
* La carpeta *service* almacenará los métodos de interacción entre las POJO y la base de datos. Para este fin, se utilizará el sistema de repositorios propio de Spring Data JPA mencionado anteriormente, y que contarán con su propia carpeta de *repositories*. Estas carpetas conforman una capa intermedia entre la capa Modelo y la capa Controlador para mantener la separación de responsabilidades y facilitar el mantenimiento del código a largo plazo.
* La carpeta *controller* almacenará la sección de los endpoints, donde se definen una serie de rutas que el servidor resolverá cuando se envíen peticiones HTTP a estas direcciones. Estas rutas se mapean con unas anotaciones como *@GetMapping* o *@PostMapping* que definen el método HTTP necesario para aceptar la petición y la ruta en la que se escucha. Al realizar una petición correcta el *controller* ejecuta la función asignada a este mapeo de ruta y devuelve una respuesta HTTP con el objeto resultante de la función ejecutada. Esta carpeta cumple la función de Controlador en el modelo MVC

#### - ESTRUCTURA DE DATOS -

Con la estructura de código ya definida, es hora de empezar con la programación propiamente dicha. En primer lugar, se definen las clases correspondientes a cada una de las entidades definidas en la base de datos como clases POJO dentro de la carpeta *data*. Estas clases contienen los atributos requeridos para las operaciones a realizar, así como un constructor vacío para crear nuevos objetos con esta clase y los métodos Getters y Setters necesarios para operar con estos objetos creados. Además, pueden contener métodos simples de manipulación de atributos.

Estas clases POJO son utilizadas posteriormente para mapear las entidades correspondientes de la base de datos. Para ello cuentan con una anotación concreta -*@Entity*- que define este comportamiento. También pueden contener anotaciones que definan sus relaciones con otras entidades -*@OneToMany* / *@ManyToMany*- y otras que definan el nombre de la columna que contiene el atributo en cuestión -*@Column*-, entre muchas más.

#### - ACCESO A DATOS -

Para que las entidades de la base de datos puedan ser accesibles desde el entorno de desarrollo Java de la API, el módulo Spring Data JPA nos ofrece su sistema de repositorios. Estos repositorios son interfaces que incorporan los métodos básicos para la interacción con la capa de acceso a datos. Spring ofrece una gran variedad de repositorios, todos partiendo de una definición general encontrada en la interfaz *Repository*. El tipo concreto de repositorio que se utiliza en este proyecto es la interfaz *CrudRepository*, que por defecto ya incluye métodos para crear –*save*-, leer –*findAll* / *findById*- y eliminar –*delete*- entidades de la base de datos, entre otros. Además, dentro de las interfaz repositorio se pueden definir métodos personalizados para ejecutar clausulas Select en la base de datos, siguiendo un lenguaje de consulta propio basado en el nombre de los propios métodos. Así, para realizar una consulta SQL tipo *‘ SELECT \* FROM users WHERE name = ‘pep’ ‘* bastaría con crear un método dentro del repositorio de la entidad *users* con el nombre *findByName(String name)*. Estos métodos definidos en los repositorios deben estar vacíos, ya que no hay que olvidar que los repositorios son interfaces que posteriormente serán implementadas en las clases de servicios, que se tratan a continuación.

#### - SERVICIOS -

Como se ha comentado en el apartado de repositorios, éstos son solamente interfaces, por lo que no se puede ejecutar ningún código en ellos aparte de definir la estructura de los métodos que implementan. Es por ello que se utilizarán unas clases que funcionarán como servicios y servirán de puente entre los repositorios y los controladores.

Estos servicios representarán una sección de la aplicación final, e implementarán los repositorios de todas las entidades requeridas en esta sección. De esta forma el código se divide por responsabilidades y es más sencillo trackear un problema al tener claramente diferenciados los distintos ámbitos de la aplicación. Otra funcionalidad básica que cumplen los servicios es el manejo de errores a la hora de llamar a los métodos de los repositorios, evitando esta gestión en el apartado de controladores.

Para que Spring identifique que una determinada clase Java tiene como función principal utilizarse como servicio se usa la anotación *@Service*. En estos servicios se utilizan también las anotaciones *@Autowired*, que sirven para definir y asignar automáticamente un repositorio al servicio en cuestión.

#### - CONTROLADORES -

Finalmente, con los métodos de interacción con la base de datos definidos en los repositorios y asignados en los servicios, la última capa para poder tener una API funcional es la capa de controladores. En esta sección es donde todo el trabajo previo se muestra como público a través de conexiones HTTP.

Cabe destacar que, por defecto, en Spring los controladores son clases que contienen métodos para devolver vistas dependiendo de las peticiones que reciban. En el caso de este proyecto la capa de Vista será administrada por un framework reactivo de JavaScript, externo a Java, y por tanto externo a Spring, por lo que no es necesaria la implementación tradicional de un controlador.

De todas formas, aún es necesario poder establecer una comunicación con la API para poder solicitar los datos que consumirá la capa de Vista de JavaScript. Es por ello que se utiliza una implementación de la capa de controlador siguiendo una estructura REST, esto significa que el controlador no devuelve una vista completa, sino información en formato JSON para ser interpretada libremente por cualquier otro software. Para este objetivo necesitamos marcar la clase controlador con la anotación *@RestController*, que se encargará de que todas las respuestas a peticiones devuelvan los datos en forma de objetos, normalmente en formato JSON.

Una vez configurado el comportamiento de la clase controlador, se procederá a crear los endpoints necesarios para comunicarse con las aplicaciones externas. Cada uno de los endpoints que se configuren deberán tener la misma estructura:

* Contarán primeramente con una anotación inicial del tipo *@GetMapping*, donde se indica el método necesario para realizar la petición utilizando este endpoint.
* El siguiente elemento es la ruta, es decir, la dirección o cadena de caracteres con la que se puede referir a cada endpoint en concreto. La misma ruta puede utilizarse en más de un endpoint siempre y cuando utilicen métodos diferentes para realizar la petición. Dentro de la ruta se pueden introducir variables, que serán usadas para realizar operaciones y retornar el resultado. Estas variables pueden ser definidas con *@PathVariable* si forman parte de la ruta en sí misma o con *@RequestParam* si son introducidas en la ruta a modo de query parameter.
* Cada endpoint, además, puede recibir más información a través de un body con *@RequestBody*, lo que significa que la aplicación de origen emitirá un contenido a la API para que esta trabaje con él.
* Con todos los elementos necesarios se procede a la ejecución de la función asociada al endpoint. Esta función puede llamar a los métodos almacenados en los diferentes servicios para operar con los objetos de la base de datos. Es en esta sección donde se introduce la mayoría de la lógica de negocio del proyecto, manteniendo a todas las demás secciones ajenas a esta lógica.

### < CONCLUSIÓN >

De esta forma es como se establece una relación directa entre las entidades almacenadas en la Base de Datos MySQL con cualquier aplicación exterior. Cabe destacar también que entre todo el proceso también se llevan a cabo las tareas de conexión con la base de datos, que en este caso en particular se realiza a través de la interfaz gráfica del propio IDE NetBeans, mediante la pestaña de *Servicio.*

Una vez finalizado todo el proceso de desarrollo, la API es completamente funcional. Para interactuar con ella es necesario ejecutar el archivo, y de esta forma poner en marcha un servidor de aplicaciones interno TomCat que de momento funcionará únicamente de forma local. Es necesario que este servidor esté activo toda vez que se necesite interactuar con la API.

En el apartado de Despliegue se profundizará en las posibilidades de hacer que este servidor funcione en una máquina externa y proporcione servicio de manera continuada sin depender de nuestra máquina local.

## { REACTIVE FRONTEND }

Con una API completamente funcional y en marcha, el proyecto de momento sirve como mediador para interactuar con una base de datos a través de peticiones HTTP. Para hacer este servicio más visible para el público en general se ha optado por crear una capa de visualización de los datos por encima, aplicando las tecnologías Reactive para generar un contenido dinámico e interactivo.

### < ANÁLISIS >

En el momento actual de desarrollar este proyecto, el mundo de desarrollo web está volcado en la programación reactiva de páginas web. Esto significa, la intención global en el mercado es la de crear sitios web que reaccionen a las interacciones del usuario y actualicen los datos que se muestran sin interrumpir la experiencia del usuario y, por tanto, sin recargar la página constantemente al requerir datos nuevos.

La estructura básica de cualquier aplicación web normalmente está basada en la trilogía de tecnologías HTML, CSS y JavaScript. Estas tecnologías se encargar de ofrecer al programador todas las herramientas necesarias para publicar un sitio web con una estructura de datos firme, una presentación visual atractiva, y una experiencia de usuario rica e interesante.

Si bien estas tres tecnologías ofrecen un gran abanico de posibilidades, a la hora de planificar sitios web reactivos pueden quedarse cortas, o necesitar de una gran cantidad de código para realizar tareas aparentemente sencillas. Aquí es donde entran en juego los frameworks reactivos de JavaScript. De la misma forma que pasa con Java y Spring, estos frameworks están desarrollados en el mismo lenguaje en el que se aplican, en el caso de Spring en Java y en el caso de estos frameworks en JavaScript. Su principal función, entonces, es la de ofrecer al programador una manera más sencilla de realizar tareas básicas, en este caso en el ámbito de la reactividad.

En el mercado actualmente existen tres grandes opciones en cuanto a reactividad en JavaScript: Angular, React y Vue. Los tres frameworks tienen unas características y funcionalidades muy similares, y el debate entre ellos sobre cuál es el más óptimo para cada proyecto normalmente suele acabar en un empate técnico o en diferencias de rendimiento que en este proyecto están en un segundo plano. Es por ello que una vez analizados los tres frameworks, se ha optado por implementar Vue.js principalmente por preferencias personales y por la claridad y cantidad de explicaciones y ejemplos disponibles en la documentación oficial de la librería. Además, al no haber trabajado en ningún proyecto anterior con el concepto de reactividad, la curva de aprendizaje de Vue.js requiere bastante menos esfuerzo que las demás opciones.

Sobre Vue, además, existen otros proyectos o frameworks que amplían sus funcionalidades. Es el caso de Nuxt.js. Este framework desarrollado sobre Vue facilita el desarrollo de aplicaciones reactivas en JavaScript basadas en servidor. Esto significa que se puede trabajar bajo los principios de web reactiva en un ámbito de servidor, desde donde realizar todas las operaciones necesarias y entregar a cada cliente la vista que solicite. Este framework de Nuxt está basado también en la implementación de Node.js, que es un conocido framework de JavaScript para crear servidores de aplicaciones que ejecuten código en este lenguaje.

La estructura de diseño de estos frameworks se basa en un concepto llamado componentes. Los componentes son secciones de código que se resuelven de forma autónoma y autocontenida. Es decir, dentro de un componente se definen tanto el aspecto visual, la información a mostrar y la interacción con el usuario, de forma independiente al resto de componentes del proyecto. Esto permite un desarrollo modular y reutilizable de los componentes, y facilita la organización del código de frontend, sobre todo en proyectos grandes.

Estos frameworks, además, cuentan con sus propios plugins para facilitar el desarrollo de módulos específicos, como pueden ser la autorización de usuarios, o la conexión HTTP. En este proyecto se considera indispensable la utilización del plugin *axios*, que aplica unas funcionalidades similares a la API fetch nativa de JavaScript pero del lado de servidor, y permite realizar peticiones complejas a direcciones HTTP mediante un sistema de Promises, que simplifica el desarrollo asíncrono de estas funciones para poder realizar la carga de datos de manera simultánea a la carga de la parte visual de la web.

De esta forma, quedan definidas las herramientas para el desarrollo de la sección visual del proyecto.

### < DESARROLLO >

El primer paso, antes de tan siquiera valorarse empezar a aplicar la reactividad al sitio web, fue definir unos *mockups* o bocetos sobre la distribución de los elementos, tratando de mantener el equilibrio y la intuición en el diseño de usuario. Este será el punto de partida para la maquetación final del aplicativo web.

#### - CONEXIÓN API -

El primer punto a tratar dentro del desarrollo del frontend es qué metodología y estructura utilizar para proceder con la hidratación de datos. Hidratación es un término utilizado en los desarrollos de sitios webs dinámicos para indicar el proceso en el que una página web con diseño estático recibe e inserta datos desde un repositorio de datos según lo requiera el usuario.

En este aspecto, el proyecto cuenta con su propio API de backend que se encarga de administrar la capa de acceso a datos. Para realizar peticiones a esta API será necesario establecer cierto paralelismo entre la estructura de datos de la API y la del propio frontend. Para ello se define una carpeta dentro de la estructura del frontend donde residirán las clases donde serán mapeados los objetos que devuelva la API. Estas clases contaran con los atributos necesarios para adaptar las características del objeto recibido.

Una vez definida la estructura de datos, se deben crear también los servicios para cada objeto o entidad a mapear, donde se definirán los métodos de interacción con el propio objeto. Estos métodos utilizarán el plugin de *axios* anteriormente mencionado para llamar a la ruta correspondiente en la API y realizar una petición adecuada para obtener el resultado deseado. De esta manera, si se desea definir un método para crear un nuevo usuario, se deberá definir dentro del servicio del objeto Usuario un método *crearUsuario()* que realice una petición post mediante el plugin axios a la ruta correspondiente en nuestro API, por ejemplo */api/usuarios/nuevo*. Esta petición, como ya se especificó en la sección del backend, puede requerir query parameters o un body para realizar la acción necesaria. Dentro del método del servicio en cuestión se pueden definir también estos aspectos.

Al ser un sistema basado en Promises, la llamada a la API realizada por *axios* se realizará de forma asíncrona, es decir, en background, y el propio software esperará a que esta llamada se haya completado con éxito para proceder con la manipulación de los datos obtenidos. Esto se consigue mediante la cláusula *then*, que indica las acciones a realizar una vez se ha resuelto con éxito la Promise. En los servicios se utilizará esta cláusula *then* para asignar la respuesta de la consulta de *axios* al objeto correspondiente definido en la estructura de datos del frontend.

Por último, Nuxt es un *opinionated* framework. Esto significa que establece una serie de normas básicas para mantener cierto orden en el código. Además, al ser un framework basado en JavaScript, cada uno de los objetos o servicios está contenido dentro de su propio archivo js. Para poder interactuar con ellos es necesario establecer un sistema de exportación/importación para acceder a los métodos u objetos de estos archivos desde archivos externos a los mismos. Puesto que los servicios son la base del proyecto de frontend para la interacción con la API, y por tanto con la base de datos, estos archivos serán solicitados constantemente. Para facilitar su utilización, Nuxt ofrece un directorio propio de plugins, donde definir código JavaScript propio que podrá ser accedido desde cualquier parte del proyecto. Es en este directorio donde se definirá un script sencillo para acceder a los servicios de una manera mas simple.

#### - RENDERIZADO -

Con la interacción con la API ya resuelta, los datos recogidos pueden ser utilizados para hidratar las páginas web. Nuxt, y más concretamente Vue, ofrece un sistema de plantillas muy optimizado y útil a la hora de trabajar con hidratación de datos. Este sistema de plantillas es embebe dentro de la estructura propia de un formato HTML, y define una serie de funcionalidades propias para facilitar la renderización.

Entre las funcionalidades más útiles, y las que se utilizarán en este proyecto, se encuentran la introducción de datos computados, la renderización condicional, o renderización en bucle. Para todas estas funcionalidades Vue especifica unas palabras claves que pueden ser utilizadas dentro del miasma cláusulas HTML. En el caso de la renderización condicional, puede ser definida mediante el comando *v-if* / *v-else*, mientras que el renderizdo en bucle es definido con *v-for*. Estas funcionalidades permiten un renderizado flexible que se adapta a los datos disponibles.

El modelo de renderizado de Vue ofrece también la posibilidad de definir todos los aspectos de un componente –estructura, diseño y interacción (HTML, CSS y JS)- dentro de un mismo archivo con extensión *.vue*. Estos archivos contienen tres secciones para definir cada uno de los aspectos: la sección <template> es donde se define el HTML del componente, la sección <style> contiene el CSS y la sección <script> contiene los métodos de JavaScript. De esta forma se pueden organizar los componentes de una manera mucho más limpia.

Dentro de cada la sección <script> de cada componente es donde se llama a la API a través de los servicios mencionados anteriormente para extraer la información necesaria para renderizar la vista en cuestión. Esta información es extraída mediante el método fetch, y asignada a un objeto definido dentro del mismo componente. Este objeto, contenedor de la información extraída de la API, será el utilizado en la sección <template> para insertar la información requerida en esta vista en particular.

Finalmente, si se quiere habilitar una navegación entre distintos componentes, Nuxt ofrece la posibilidad de definir un componente dentro de una carpeta *pages*. Con este procedimiento, el framework se encarga de administrar este componente como si fuera una página principal, asignarle una dirección de enrutado y así permitir la navegación a este componente a través de links.

## { DESPLIEGUE }

Una vez finalizado el desarrollo del proyecto al completo, incluyendo la definición de la Base de Datos, la funcionalidad de la API y la renderización del frontend reactivo, el próximo paso lógico sería proceder con el despliegue de la aplicación. En el caso de este proyecto, no se ha llegado al proceso de despliegue de forma práctica, pero sí se han analizado las distintas opciones de cara a una futura ampliación de ámbito de desarrollo.

El primer paso para proceder con la fase de despliegue es la generación de ejecutables a partir del código desarrollado. Para ello Spring Boot ofrece la funcionalidad de ejecutar una build del proyecto, lo que genera un archivo *.jar* ejecutable desde el cual lanzar la API a producción. De momento el ámbito de aplicación de este proyecto es meramente formativo e investigador, por lo que no se procederá con tareas de despliegue al público de momento.

Con la API ya lista para pasar a la fase de producción, empaquetada en un archivo ejecutable jar, la aplicación de frontend deberá pasar por el mismo proceso. En este caso, se trata de un proyecto desarrollado en Node.js, que ofrece su propio sistema gestor de paquetes *npm*, del cual se pueden obtener la mayoría de funciones necesarias mediante líneas de comandos. Durante el desarrollo, sin ir más lejos, se han utilizado las instrucciones *npm install* para importar librerías necesarias, como *axios* al proyecto, o *npm run dev*, para ejecutar el proyecto en un entorno de desarrollo donde poder testear el resultado final. Para construir el proyecto, el comando *npm run build* es el que se encargará de generar una versión de producción del software, que podrá inicializarse con *npm run start*.

Así, ya estarían disponibles la versión de producción de la API en el archivo .jar, que pondría en marcha el servidor TomCat, y la versión de producción del servidor del frontend, a través del comando npm run start, que iniciaría el servidor Node.js. Para pasar al siguiente paso y realizar un despliegue público sería necesario contar con un sistema de hosting que permitiera mantener los servidores de aplicaciones activos para poder acceder a ellos desde cualquier lugar. Este aspecto se considera fuera del ámbito de conocimiento que intenta abarcar este proyecto, y aunque se han investigado opciones como Docker y Heroku para realizar esta tarea, son aspectos a considerar en futuras líneas de expansión.

# CONCLUSIÓN

## { SATISFACCIÓN }

En general este proyecto ha sido para mi un reto monumental. Los conocimientos con los que empecé el desarrollo me permitieron entender las bases de los procesos necesarios para completar el proyecto, pero no me consideraba capaz de lograr todo lo que he conseguido. De todas formas, considero un poco frustrante no haber podido llegar en el plazo definido, a la publicación a producción de la aplicación. Quedan muchas funcionalidades que incorporar y hay mucha más información que investigar.

Por otra parte, en este proyecto me he basado únicamente en la parte práctica, sin prestar atención a la optimización del rendimiento del software, lo que, a la larga, considero que es una irresponsabilidad para cualquier desarrollador.

Dicho esto, la satisfacción general con el resultado del proyecto es medianamente buena. La cantidad de conocimientos nuevos adquiridos, y la confianza de ver la posibilidad de crear un software funcional y capaz de resolver problemas reales es una motivación que me empujará a seguir investigando, desarrollando y ampliando conocimientos. En general ha sido una experiencia constructiva, estresante y satisfactoria.

## { CONCEPTOS ADQUIRIDOS }

Como comentaba con anterioridad, los conocimientos con los que contaba a la hora de iniciar el desarrollo considero que me daban una base sólida sobre la que construir nuevas metodologías de trabajo. Aun así, esos conocimientos no eran, ni de lejos, suficientes para proceder con un desarrollo a la escala del proyecto presentado. Es por ello que, atendiendo a la necesidad, he adquirido algunos conocimientos clave necesarios para ejercer esta profesión.

Quizás el conocimiento que más agradezco haber descubierto y puesto en práctica no es más que la organización. Aunque considero que no he ejercido una organización óptima del flujo de trabajo, sí que he podido observar como la habilidad de planificar un proyecto de manera exhaustiva puede ser la diferencia entre el éxito o el fracaso del mismo.

Otro aspecto fundamental que considero adquirido es el conocimiento sobre la comunicación HTTP. Hoy en día es un pilar básico para la conexión de aplicaciones a través de la red, y es un aspecto que puede parecer algo oscuro y opaco al principio pero que agradezco enormemente haber podido poner en práctica.

Finalmente, el hecho de haber incluido algunos de los frameworks más populares en los lenguajes utilizados me ha permitido identificar porque son usados por tanta gente. La cantidad de funcionalidades que añaden, o la manera en la que las hacen accesibles a la mayoría de desarrolladores desde un punto de vista mucho más simple hace que sean una herramienta difícil de obviar a la hora de pensar en la planificación de un nuevo proyecto.

## { FUNCIONALIDADES PENDIENTES }

Aun sintiéndome tremendamente orgulloso por el proyecto presentado, no puedo dejar de sentir cierta frustración a la hora de pensar en la cantidad de funcionalidades que han quedado fuera del proyecto por la necesidad de presentarlo en el plazo determinado. Imagino que es un sentimiento común para la mayoría de desarrolladores, sobre todo en el mundo laboral real donde los plazos los establecen los clientes.

Considero que un fallo de novato ha sido intentar abarcar muchas funcionalidades dentro de un proyecto donde no dominaba gran parte del stack que se iba a utilizar. También pienso que algunas funcionalidades deberían haber sido priorizadas sobre otras, viendo a posteriori que no ha sido suficiente el tiempo dedicado para implementar el núcleo de funcionalidades previsto.

Aunque el proyecto, en este punto, permite una interacción prácticamente plena con los menús o cartas de los restaurantes –añadir nuevos platos o menús, modificarlos y eliminarlos, realizar una comanda como cliente … -, me gustaría haber podido incluir un sistema de registro de usuarios para dar de alta nuevos restaurantes, y un administrador de perfiles de usuario para que los clientes pudieran definir algunas de sus prioridades. Otro punto fundamental que creo que se ha quedado en el tintero es el desarrollo de un sistema de rating de los restaurantes.

De todas formas, considero más que satisfactorio el resultado final, sobre todo teniendo en cuenta que es un desarrollo a tiempo parcial, compaginado con un trabajo de jornada completa y el seguimiento de las clases online de otras asignaturas.

## { LÍNEAS FUTURAS }

Veo mucho potencial en este proyecto y me gustaría seguir dedicándole tiempo tras la entrega del proyecto. En este punto la aplicación web tiene un aspecto tosco y no está optimizado en cuanto a interacción con el usuario, lo que considero un aspecto fundamental por si en cualquier caso fuera posible hacer público el proyecto para que más gente disfrutara de él.

También veo completamente necesario adjuntar un sistema de registro y administración de usuarios, a la vez que se implementa un sistema de autenticación y seguridad para salvaguardar los datos de los clientes registrados con algún tipo de encriptación, como mínimo, de las contraseñas.

Otro punto de desarrollo futuro que veo factible en el próximo año es ampliar la compatibilidad con los dispositivos de los usuarios. Actualmente la parte visual está diseñada únicamente para dispositivos de escritorio, lo que es algo inadmisible si se quiere estar bien posicionado en Google hoy en día y, lo más importante, que los usuarios tengan una buena experiencia de uso de la aplicación.

Finalmente, me gustaría también migrar este proyecto a una App de Android / iOS. Es verdad que dentro de las aplicaciones web hay cada vez más posibilidades, y que las diferencias entre app y aplicación web cada vez se reducen más, por lo que es un tema a investigar. Las Single Page Aplications y la nueva funcionalidad de Chrome de instalar aplicaciones web para acceder a ellas con más facilidad son aspectos a tener muy en cuenta en cuanto a desarrollo de líneas futuras.

# BIBLIOGRAFÍA

* La idea principal de esquema de desarrollo de la aplicación fue inspirada en este tutorial – [Building a Full Stack Web App](https://milanwittpohl.com/projects/tutorials/full-stack-web-app/the-backend-with-java-and-spring)
* [Patrón MVC en Java con NetBeans](https://code.google.com/archive/p/gestion-matricula/wikis/MVC.wiki)
* [Spring MVC CRUD Example](https://www.javatpoint.com/spring-mvc-crud-example)
* [Full Stack Java development with Spring Boot and Vue.js](https://www.danvega.dev/blog/2021/01/22/full-stack-java-vue/)
* [Spring Data JPA Docs](https://spring.io/projects/spring-data-jpa)
* [Spring Web MVC Docs](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.h)
* [Vue 3 Docs](https://v3.vuejs.org/guide/introduction.html)
* [Nuxt.js Docs](https://nuxtjs.org/docs/2.x/get-started/installation)
* [StackOverflow](https://stackoverflow.com/)

# ANEXOS

- SOURCE CODE -

Este repositorio de GitHub contiene todo el código desarrollado relacionado con el proyecto RestROOM → [PepCarmona/RestROOM](https://github.com/PepCarmona/RestROOM)