



GESTIÓN DE RESTAURANTE

Bea Couchoud Ruiz

Memoria del Proyecto de DAW

IES Abastos

Curso 2019-2020

Grupo 7K

Valencia, 4 de junio de 2020

Tutor individual: Carlos Furones

Tabla de contenido

1. Introducción	3
1.1 Justificación.....	3
1.2 Objetivos	3
1.3 Idea inicial	3
2. Diseño.....	4
2.1 Análisis.....	4
2.1.1 Tecnologías y Herramientas	4
2.2 Planificación.....	8
2.2.1 Requisitos.....	8
2.2.1.1 Requisitos funcionales.....	8
2.2.1.2 Requisitos no funcionales.....	10
2.2.2 Gestión del proyecto	10
2.2.2.1 Ciclo de vida del proyecto.....	11
2.2.2.2 Diagrama de Gantt	12
2.3 Modelo Entidad-Relación y Base de Datos	14
2.4 Casos de Uso.....	15
3. Desarrollo.....	18
3.1 Codificación.....	18
3.1.1 Estructura de Carpetas	18
3.1.1.1 Front-end.....	19
3.1.1.1.1 Páginas y enrutamiento.....	20
3.1.1.1.2 Componentes.....	22
3.1.1.1.3 Servicios	23
3.1.1.1.4 Formularios y seguridad.....	25
3.1.1.2 Back-end.....	28
3.1.1.2.1 Seguridad.....	29
3.1.1.2.2 Rutas	30
3.1.1.2.3 Sesiones	31
3.2 Implantación y configuración de la aplicación.....	32
3.3 Control de versiones	33
4. Conclusiones	34
4.1 Posibles mejoras.....	34
4.2 Dificultades	34

Gestión de restaurante

4.3	Logros.....	34
4.4	Resultado.....	34
5.	Referencias y Bibliografía	35

1. Introducción

En el siguiente documento se pretende presentar la aplicación desarrollada, comentar el origen y la motivación que llevó a hacer realidad este proyecto, los objetivos del mismo, y detallar todas las características técnicas que permitan entender qué hace la aplicación y cómo lo hace.

Justificación

Actualmente, a causa de la COVID-19, muchos restaurantes se están viendo obligados a ofrecer sus servicios a través de internet mediante la venta online de sus productos. Detectada esta necesidad se ha buscado la creación de una herramienta fácil de usar que permita a cualquier restaurante aceptar y gestionar pedidos online y mantener una pequeña página web de manera sencilla y rápida.

Por tanto, este proyecto no busca innovar, si no crear una herramienta útil y funcional que permita tanto realizar como gestionar pedidos a domicilio de un determinado restaurante.

Objetivos

El objetivo principal del proyecto es crear una herramienta o “plantilla de herramienta” consistente en una aplicación web en la cual, por un lado, los usuarios puedan obtener información sobre el restaurante y pedir comida a domicilio y, por otro lado, que el restaurante pueda gestionar rápidamente tanto estos pedidos como cualquier otro aspecto del restaurante (qué se muestra en la web a los usuarios, creación de menús nuevos de manera sencilla, etc).

Como objetivo secundario, se ha buscado realizar la aplicación utilizando tecnologías que no se han visto a lo largo del curso, aprovechando así la realización de este proyecto para obtener nuevos conocimientos.

Idea inicial

En un principio las necesidades que se buscan cubrir con esta aplicación web son, en función del rol del usuario:

Los posibles clientes (personas que entran en la web a obtener información) deben poder informarse de los menús disponibles, consultar la carta y conocer la ubicación del restaurante. Los usuarios registrados, además, deben poder realizar pedidos, consultar sus pedidos previos y administrar sus datos de usuario. Por otra parte, los empleados deben poder consultar rápidamente los pedidos, su estado y modificarlos (cancelar el pedido, actualizar su estado...). Por último, el administrador tiene que poder crear platos y menús nuevos, editar los ya existentes y gestionar los usuarios de la aplicación.

Todo esto debe poder controlarse mediante una interfaz gráfica intuitiva y sencilla de manejar.

2. Diseño

Este apartado está dedicado a la fase de diseño del proyecto. Basándose en las necesidades del sistema y la información de la etapa de análisis, se detallarán los componentes que forman el sistema, con el fin de llevar a cabo su desarrollo.

El objetivo de la etapa de diseño, es la definición de la arquitectura del sistema, tanto hardware como software, y la especificación detallada de sus distintos componentes, así como del entorno tecnológico necesario para dar soporte a su funcionamiento.

2.1 Análisis

Aquí se recoge toda la información relativa a la fase de análisis del sistema. Quedan definidos tanto las tecnologías y herramientas necesarias para llevar a cabo este proyecto como los requisitos de software necesarios para usar la aplicación y todo ello sirve de base para la siguiente etapa del proyecto.

2.1.1 Tecnologías y Herramientas

Las tecnologías se han elegido para realizar este proyecto son las siguientes:

SPA

Las SPA son aplicaciones de una sola página, es decir, la navegación entre las diferentes secciones del sitio se realiza de forma dinámica, al instante y sin refrescar la página en el navegador en ningún momento. Estas características han incrementado el uso de estas en los últimos años.

Angular / TypeScript

Angular es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), haciendo que el desarrollo y las pruebas sean más fáciles.

Primero lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript.

Angular se basa en clases tipo "Componentes", cuyas propiedades son las usadas para hacer el binding de los datos. En dichas clases hay propiedades (variables) y métodos (funciones a llamar).

Algunas de las ventajas que ofrece son:

Gestión de restaurante

- **Mejora de la productividad:** el planteamiento de arquitecturas estándar repercute directamente en la productividad del proyecto. Angular proporciona controladores, servicios y directivas para organizar el proyecto.
- **Generación de código:** Angular convierte tus plantillas en código altamente optimizado para las máquinas virtuales de JavaScript de hoy en día, ofreciéndote todas las ventajas del código escrito a mano con la productividad de un framework.
- **División del código:** Las aplicaciones de Angular se cargan rápidamente gracias al nuevo enrutador de componentes. Éste ofrece una división automática de códigos para que los usuarios sólo carguen el código necesario para procesar la vista que solicitan.
- **Reutilización de código:** El diseño de Angular adopta el estándar de los componentes web. Un componente en Angular es una porción de código que es posible reutilizar en otros proyectos de Angular sin apenas esfuerzo, en otras palabras, te permiten crear nuevas etiquetas HTML personalizadas, reutilizables y auto-contenidas, que luego se pueden utilizar en otras páginas y aplicaciones web. Esto facilita el desarrollo de aplicaciones de manera mucho más ágil, pasando de un "costoso" MVC a un juego de puzles con los componentes.
- **Angular CLI:** Las herramientas de línea de comandos permiten empezar a desarrollar rápidamente, añadir componentes y realizar test, así como previsualizar de forma instantánea la aplicación.

Aunque Angular no te obliga a usar TypeScript, se ha decidido usarlo para la realización del proyecto por varias razones, una de las primeras es la consistencia en la documentación. Por ejemplo, ES6 (o sea, ECMAScript 2015) ofrece varias formas diferentes de declarar un objeto, lo cual puede confundir a muchos. Con TypeScript esto no pasa, y toda la sintaxis y la manera de hacer las cosas en el código es la misma, lo que añade coherencia a la información y a la forma de leer el código. Además, la búsqueda de errores de runtime en JavaScript puede ser una tarea imposible. TypeScript proporciona detección temprana de errores (en tiempo de compilación), y tipado fuerte de clases, métodos, así como de objetos y APIs JavaScript ya existentes.

Además, TypeScript es un superset de ECMAScript 6, es decir, incluye todas las funcionalidades de ES6, y además incorpora una capa por encima con funcionalidades extra.

Esto, entre otras cosas, significa que puedes mezclar código TypeScript con código ES6 estándar sin problema, y el compilador de TypeScript seguirá pasando el código a ES5 (tanto ES6 como TypeScript se transpilan a ES5 para que los navegadores actuales puedan ejecutar el código).

Express / NodeJS (JavaScript)

Express es un framework web, escrito en JavaScript y alojado dentro del entorno de ejecución NodeJS.

Node (o más correctamente: *Node.js*) es un entorno que trabaja en tiempo de ejecución, de código abierto, multiplataforma, que permite a los desarrolladores crear

toda clase de herramientas de lado servidor y aplicaciones en JavaScript. El entorno omite las APIs de JavaScript específicas del explorador web y añade soporte para APIs de sistema operativo más tradicionales que incluyen HTTP y bibliotecas de sistemas de ficheros. Algunas de sus ventajas son:

- El código está escrito en "simple JavaScript", lo que significa que se pierde menos tiempo ocupándose de las "conmutaciones de contexto" entre lenguajes cuando estás escribiendo tanto el código del explorador web como del servidor.
- JavaScript es un lenguaje de programación relativamente nuevo y se beneficia de los avances en diseño de lenguajes cuando se compara con otros lenguajes de servidor web tradicionales (Python, PHP, etc.).
- El gestor de paquetes de *Node* (NPM del inglés: Node Packet Manager) proporciona acceso a cientos o miles de paquetes reutilizables. Tiene además la mejor en su clase resolución de dependencias y puede usarse para automatizar la mayor parte de la cadena de herramientas de compilación.

Express es el framework web más popular de *Node*, y es la librería subyacente para un gran número de otros frameworks web de Node populares. Proporciona mecanismos para:

- Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
- Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
- Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
- Con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla.

A pesar de que *Express* es en sí mismo bastante minimalista, los desarrolladores han creado paquetes de middleware compatibles para abordar casi cualquier problema de desarrollo web. Hay librerías para trabajar con cookies, sesiones, inicios de sesión de usuario, parámetros URL, datos POST, cabeceras de seguridad y *muchos* más.

En contraposición, esta flexibilidad es una espada de doble filo. Hay paquetes de middleware para abordar casi cualquier problema o requerimiento, pero deducir cuáles son los paquetes adecuados a usar algunas veces puede ser muy complicado. Tampoco hay una "forma correcta" de estructurar una aplicación, y muchos ejemplos que puedes encontrar en Internet no son óptimos, o sólo muestran una pequeña parte de lo que necesitas hacer para desarrollar una aplicación web.

PhpMyAdmin (XAMPP) / MySQL

El modelo relacional es el modelo de datos que emplean la mayoría de las bases de datos actuales. MySQL es un sistema gestor de bases de datos que emplea el modelo relacional y que utiliza SQL como lenguaje de consulta.

- MySQL es el sistema de gestión de bases de datos relacional más extendido en la actualidad al estar basada en código abierto, permitiendo su uso gratuito.

Gestión de restaurante

- Es muy fácil de usar. Podemos empezar a usar la base de datos MySQL sabiendo unos pocos comandos.
- Pocos requerimientos y eficiencia de memoria. Tiene una baja fuga de memoria y necesita pocos recursos de CPU o RAM.
- Es compatible con Linux y Windows.

Por otra parte, phpMyAdmin, un cliente de MySQL muy popular, es una herramienta gratuita, que permite de una manera muy completa acceder a todas las funciones de la base de datos MySQL, mediante una interfaz web muy intuitiva. Esta aplicación nos permitirá realizar las operaciones básicas en base de datos MySQL, como son: crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar sentencias SQL, administrar claves de campos, administrar privilegios y exportar datos en varios formatos. La función de exportar datos se emplea muchas veces para realizar backups de la base de datos y poder restaurar esta copia de seguridad en el futuro a través de phpMyAdmin mediante la opción “importar”. Para utilizar phpMyAdmin instalaremos XAMPP.

XAMPP es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl.

Una de las ventajas de usar XAMPP es que su instalación es de lo más sencilla, basta descargarlo, extraerlo y comenzar a usarlo. Las configuraciones son mínimas o inexistentes, lo cual nos ahorra bastante tiempo.

Bootstrap

Bootstrap es uno de los frameworks CSS más populares utilizado para crear sitios web y aplicaciones responsivas y fiables. Se trata de un paquete compuesto por una estructura de archivos y carpetas de código estandarizado (HTML, CSS, documentos JS, etc.) que se pueden utilizar para apoyar el desarrollo de sitios web, como base para comenzar a construir un sitio.

Se ha elegido Bootstrap frente a otras librerías de estilos como Material porque, entre otras cosas, ofrece más posibilidades, tiene una gran comunidad de usuarios (lo que facilita cualquier tipo de consulta), más plugin desarrollados y una buena documentación. Sin duda, este framework es la mejor elección a día de hoy para comenzar una página web.

Visual Studio Code

Visual Studio Code es un potente editor de código fuente multiplataforma Windows, Mac, Linux con reconocimiento de sintaxis de código y coloreado de una multitud de lenguajes e integración con Git.

VSC es un editor ligero, no un IDE completo, es decir no es un sustituto del Visual Studio, sino que más bien es una herramienta básica para cubrir las necesidades de edición de código simple. Lo más importante de este editor es su gratuidad, que es multiplataforma y que ofrece muchas posibilidades gracias a múltiples extensiones.

Git

Git es el software de control de versiones más utilizado. Entre otras muchas cosas, Git te permite subir y actualizar el código de tu página web a la nube de GitHub. De esta forma siempre puedes disponer de él cuando lo necesites. Pero, además puedes:

- Conocer quién ha sido el responsable de una determinada modificación y cuándo la ha realizado.
- Realizar comparaciones entre versiones de una aplicación.
- Observar la evolución del proyecto con el paso del tiempo.
- Contar con una copia del código fuente para poder volver atrás ante cualquier imprevisto en la página web.
- Estar al tanto de los cambios en el código fuente.
- Tener una copia de seguridad del proyecto al completo.
- Disponer de un historial en el que se detallen las modificaciones realizadas en el código del sitio web.
- Git es software libre y open-source.

2.2 Planificación

En esta fase, el objetivo prioritario es identificar cada una de las necesidades que deben ser satisfechas al desarrollar el sistema final. Esto se hará efectivo mediante la obtención de requisitos que deben estar presentes en el sistema de información final, con la intención de dar solución a las necesidades identificadas con anterioridad. Al finalizar esta primera etapa en el desarrollo del producto se consiguen los requisitos, tanto funcionales como no funcionales, y su modelado mediante los casos de uso, desarrollados estos últimos en diagramas. Todo esto facilita la comprensión del sistema haciendo que pueda ser diseñado cumpliendo con todos los objetivos y cubriendo las necesidades de forma satisfactoria.

2.2.1 Requisitos

En esta sección la finalidad es detallar de una manera clara y concisa cada uno de los objetivos que presenta el proyecto, así como todas sus características.

Es de suma importancia para que el desarrollo del proyecto concluya con éxito que antes de empezar se tenga una completa y plena comprensión de los requisitos del software.

Para organizar de la forma más clara posible todos los requisitos se utilizará esta tabla. El formato del identificador será RF o RNF (requisito funcional o requisito no funcional respectivamente), seguido de un número (un índice de dos dígitos). Estos identificadores podrán ser después usados para organizar tareas con mayor facilidad. Por ejemplo: "RF - 01".

IDENTIFICADOR
NOMBRE:
PRIORIDAD:
DESCRIPCIÓN:

2.2.1.1 Requisitos funcionales

Gestión de restaurante

RF - 01
NOMBRE: Creación y edición de platos
PRIORIDAD: Muy alta
DESCRIPCIÓN: El administrador debe poder crear y editar platos a través de un formulario sencillo. Los platos se crearán habilitados por defecto y una vez creados se podrán habilitar y deshabilitar.

RF - 02
NOMBRE: Creación y edición de menús
PRIORIDAD: Muy alta
DESCRIPCIÓN: El administrador debe poder crear y editar menús a través de un formulario sencillo. Los menús se crearán habilitados por defecto y una vez creados se podrán habilitar y deshabilitar.

RF - 03
NOMBRE: Creación y edición de usuarios
PRIORIDAD: Muy alta
DESCRIPCIÓN: Los usuarios deben poder registrarse en la página y editar sus datos una vez se ha creado su cuenta. Además, el administrador debe poder modificar el rol del usuario y habilitarlo o deshabilitarlo.

RF - 04
NOMBRE: Creación y edición de pedidos
PRIORIDAD: Muy alta
DESCRIPCIÓN: Los usuarios deben poder realizar pedidos fácilmente y cancelarlos en caso necesario. A su vez, los empleados deben poder ver los pedidos y su estado a simple vista, rechazarlos y modificar su estado, tanto para su propia información como para mantener al usuario al corriente del estado de su pedido.

RF - 05
NOMBRE: Creación y edición de pedidos
PRIORIDAD: Muy alta
DESCRIPCIÓN: Los usuarios deben poder realizar pedidos fácilmente y cancelarlos en caso necesario. A su vez, los empleados deben poder ver los pedidos y su estado a simple vista, rechazarlos y modificar su estado, tanto para su propia información como para mantener al usuario al corriente del estado de su pedido.

RF - 06
NOMBRE: Inicio y cierre de sesión
PRIORIDAD: Muy alta
DESCRIPCIÓN: Los usuarios deben poder iniciar/cerrar sesión en todo momento.

Gestión de restaurante

RF - 07
NOMBRE: Contenidos de la web
PRIORIDAD: Media/Baja
DESCRIPCIÓN: El administrador puede modificar los textos que aparecen en la web que ven los clientes.

2.2.1.2 Requisitos no funcionales

RNF - 01
NOMBRE: Tiempo
PRIORIDAD: Alta
DESCRIPCIÓN: El tiempo de desarrollo debe ser de 25 horas.

RNF - 02
NOMBRE: Seguridad
PRIORIDAD: Alta
DESCRIPCIÓN: Control de sesiones, autenticación de nivel de usuario y encriptación de contraseñas.

RNF - 03
NOMBRE: Interfaz de usuario
PRIORIDAD: Media/Alta
DESCRIPCIÓN: La interfaz debe ser simple y fácil de manejar. Los colores se usarán de forma estratégica para permitir identificar acciones/estados de diferentes elementos de la página a simple vista. Es importante dar feedback al usuario del resultado de su interacción con la página.

RNF - 04
NOMBRE: Validación de formularios
PRIORIDAD: Media
DESCRIPCIÓN: Los formularios tienen que validarse en el HTML y en el controlador. Además, deberán sanitizarse los datos en el servidor antes de realizar las consultas a base de datos. La base de datos no contendrá información extremadamente sensible y el riesgo de ataque al servidor es muy baja, pero es preferible mantener esta medida de seguridad.

2.2.2 Gestión del proyecto

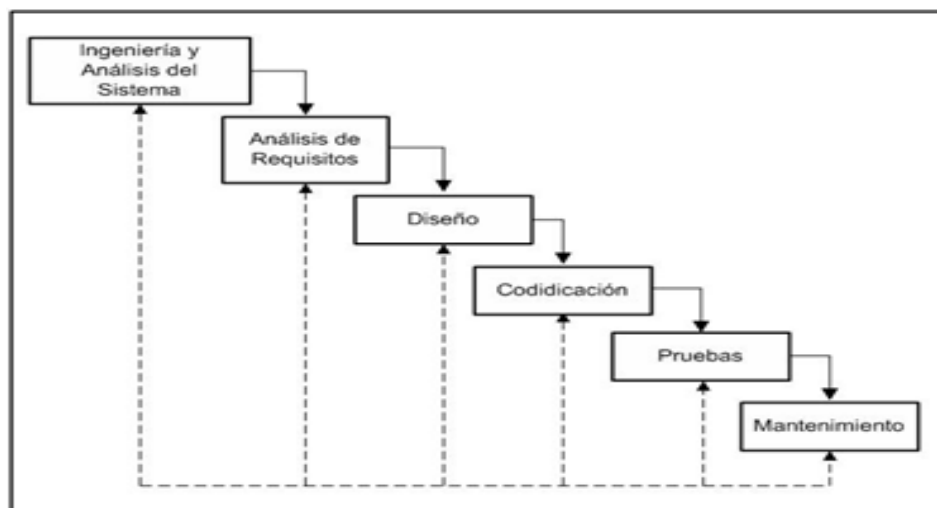
Para gestionar de manera correcta el proyecto se necesita elegir el ciclo de vida más apropiado para el proyecto en función de nuestras necesidades. Otra parte muy importante de la gestión del proyecto es la estimación de tiempo de las diferentes tareas, para lo cual se realiza un diagrama de Gantt.

2.2.2.1 Ciclo de vida del proyecto

El proceso para el desarrollo de software, también denominado ciclo de vida del desarrollo de software, es una estructura aplicada al desarrollo de un producto de software. Hay varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque distinto para diferentes actividades que tienen lugar durante el proceso.

El paradigma elegido es el desarrollo en cascada con retroalimentación. Este modelo admite la posibilidad de hacer iteraciones, es decir, durante las modificaciones que se hacen en el mantenimiento se puede ver, por ejemplo, la necesidad de cambiar algo en el diseño, lo cual significa que se harán los cambios necesarios en la codificación y se tendrán que realizar de nuevo las pruebas, es decir, si se tiene que volver a una de las etapas anteriores al mantenimiento hay que recorrer de nuevo el resto de las etapas.

Después de cada etapa se realiza una revisión para comprobar si se puede pasar a la siguiente.



Paradigma de desarrollo en cascada con retroalimentación

Los motivos principales para su elección han sido que la planificación es muy sencilla, la calidad del producto resultante es alta, permite retroceder en cualquier momento si fuera necesario y es el más sencillo de utilizar cuando el desarrollador tiene poca experiencia.

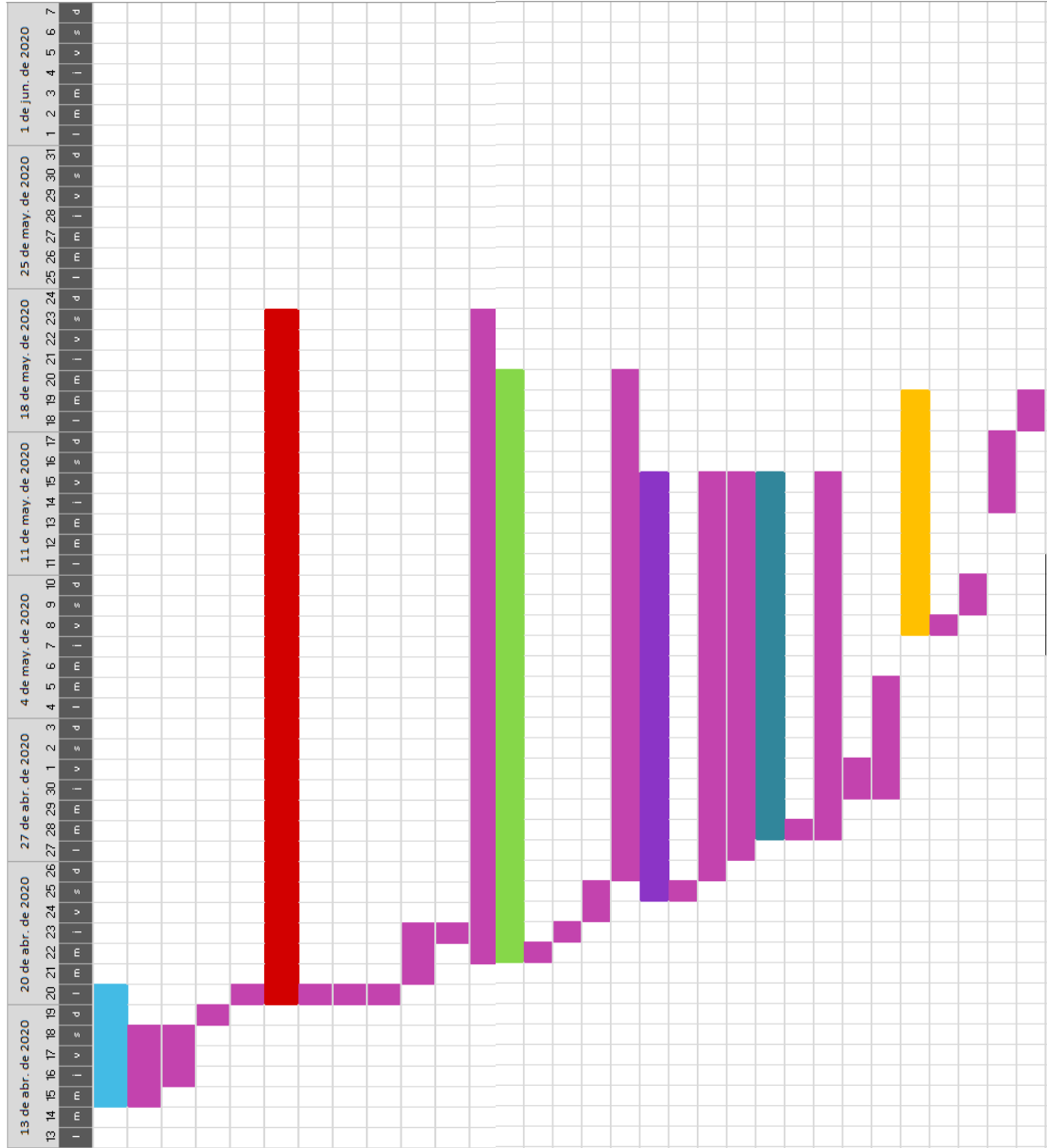
Una vez elegido el modelo de ciclo de vida del proyecto se ha procedido a planificar cada una de sus fases con el objetivo de estimar el tiempo que va a ser utilizado para la realización completa del proyecto.

2.2.2.2 Diagrama de Gantt

En el diagrama de Gantt se puede observar la duración de cada uno de los plazos de realización de tareas, separados por las fases anteriormente descritas. Se puede observar que algunas de las tareas se solapan puesto que pueden ser desarrolladas de forma simultánea, siendo independientes unas con otras, mientras que otras necesitan que las anteriores hayan sido finalizadas para poder iniciarse. La estimación realizada plantea una duración de, aproximadamente, dos meses.

GESTIÓN DE RESTAURANTE Proyecto Final DAW, IES Abastos Bea Couchoud Ruiz	mi, 15/4/2020
	1

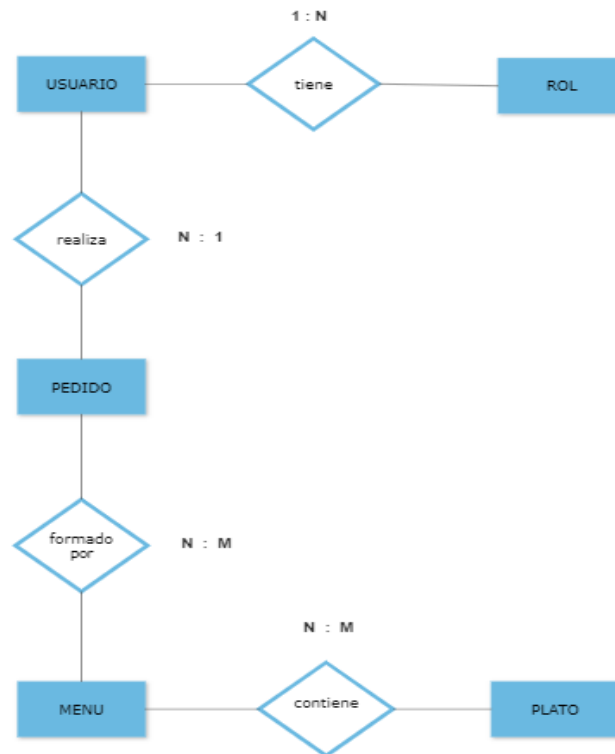
mi, 15/4/2020	1
---------------	---



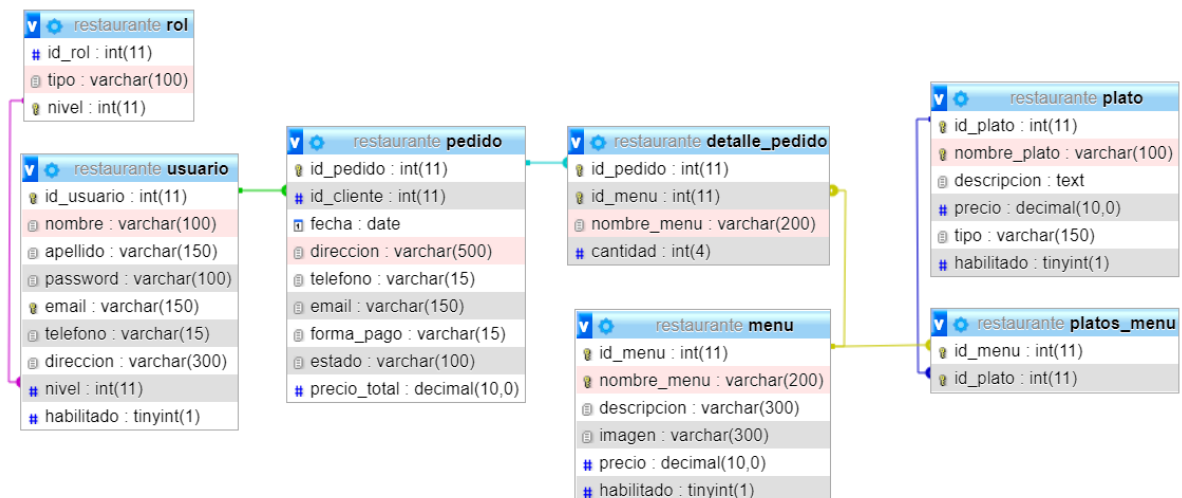
2.3 Modelo Entidad-Relación y Base de Datos

Mediante este diagrama entidad-relación se puede representar todas las entidades que presentan relevancia del sistema, junto con sus propiedades e interrelaciones.

Un primer diagrama ER muy primitivo quedaría así:



Tras añadir los atributos correspondientes y normalizar la base de datos para evitar la redundancia de información y facilitar su gestión posterior, el resultado final sería el que se muestra a continuación:

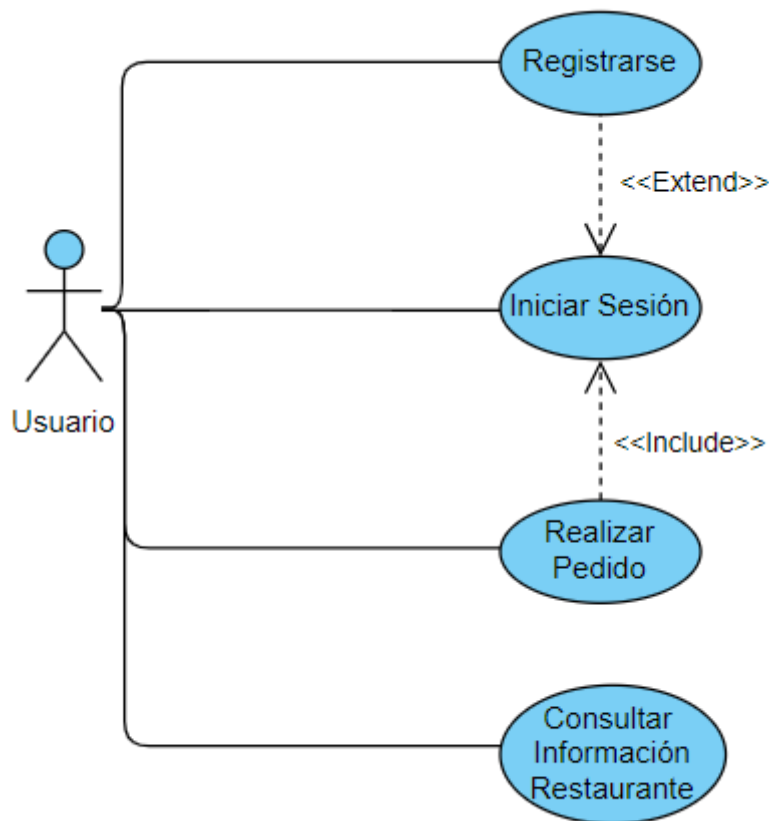


2.4 Casos de Uso

A continuación, se detallarán los casos de uso, describiendo todas las posibles interacciones entre el usuario del sistema y la aplicación con el fin de alcanzar un objetivo. De esta manera se proporcionan diversos escenarios que indican la sucesión de respuestas surgidas ante las acciones iniciadas por los usuarios sobre el sistema. Esta especificación de casos de uso facilita la comprensión del sistema y expresa la intención con la que el usuario interactuará con la aplicación, profundizando así en dichas necesidades, y que, complementándose con los requisitos anteriormente descritos, establece una firme base a la hora de comenzar con la fase de diseño.

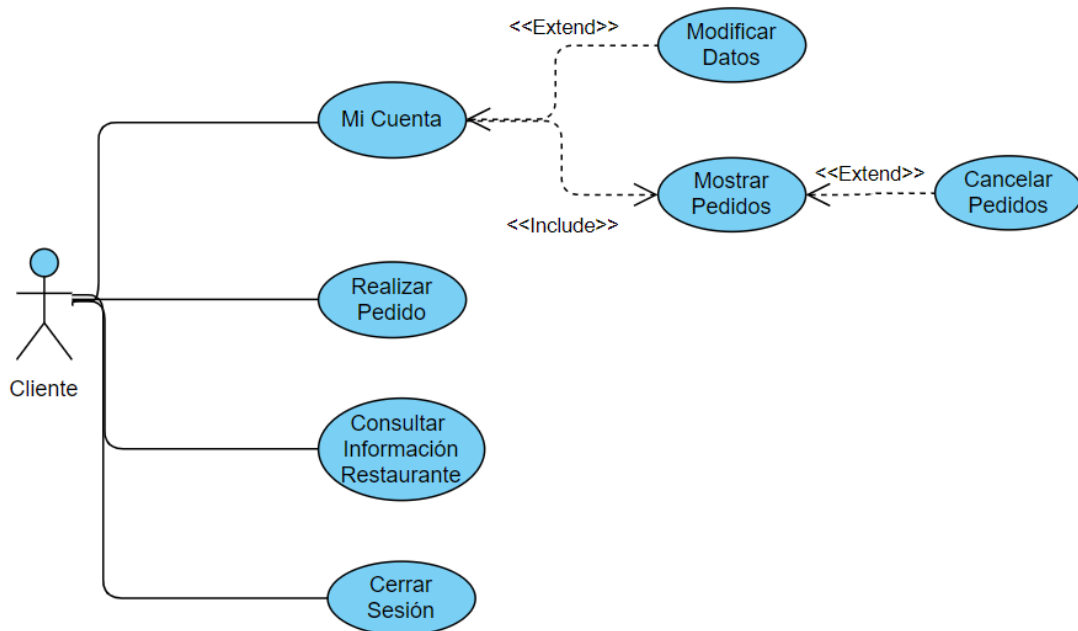
Se ha creado un caso de uso para cada rol de usuario de la aplicación (usuario no registrado, cliente, empleado y administrador).

Caso de uso desde la perspectiva de un usuario no registrado

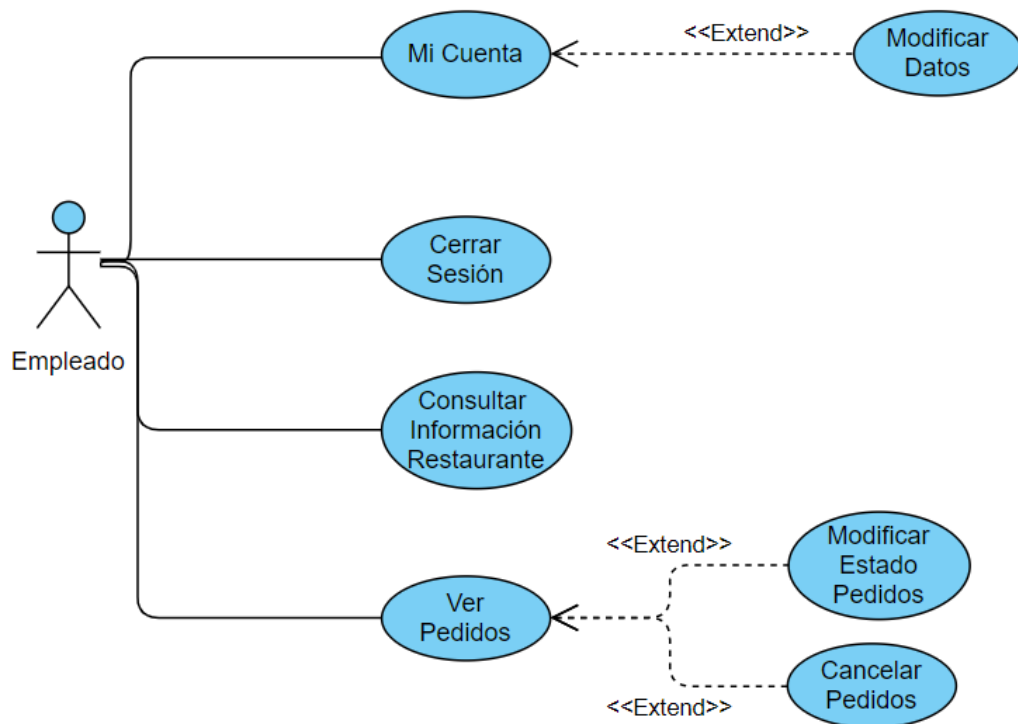


Gestión de restaurante

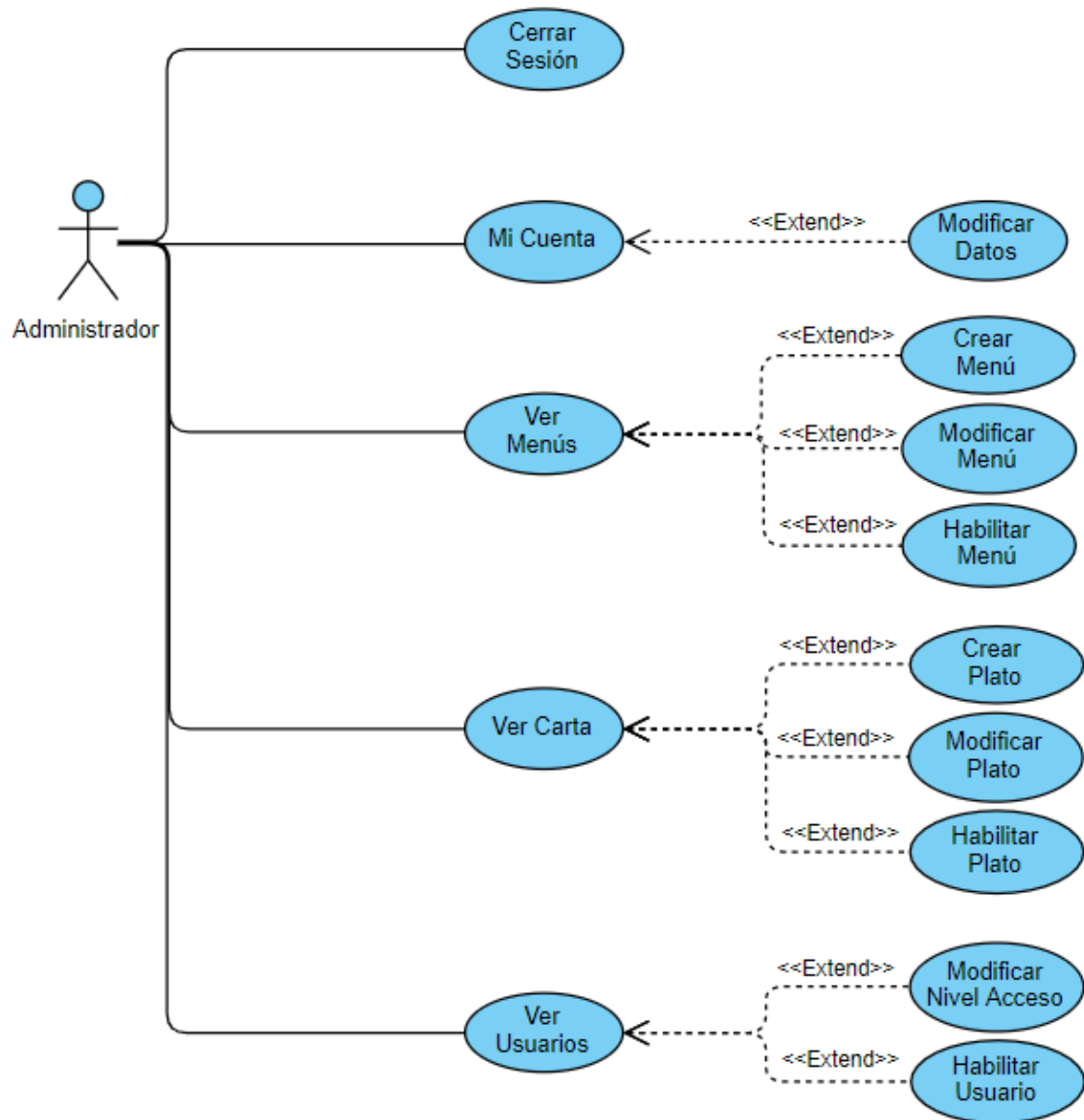
Caso de uso desde la perspectiva de un cliente o usuario registrado



Caso de uso desde la perspectiva de un empleado



Caso de uso desde la perspectiva de un administrador



3. Desarrollo

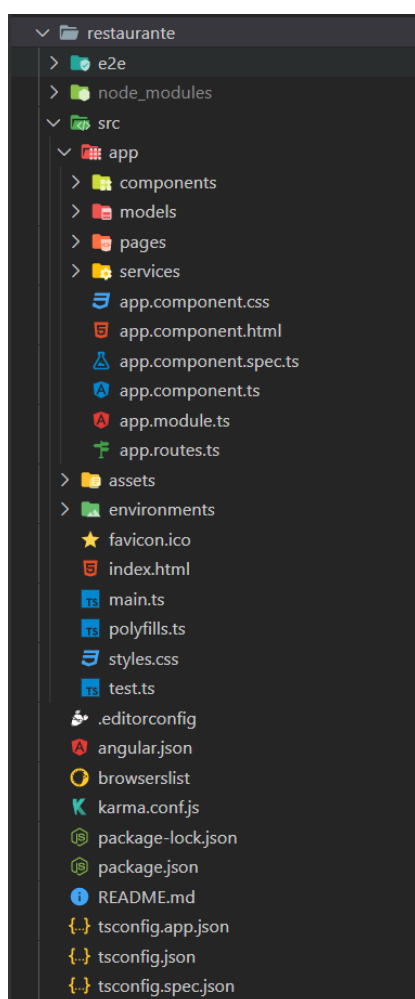
En este apartado se explicará cómo se ha creado todo el código detrás de la aplicación: La estructura escogida, el seguimiento de los archivos, como se desplegaría sobre un servidor...

Esta es la fase en la que se codifica los diferentes módulos y componentes del sistema. Aquí se han creado las funciones y procedimientos para llevar a cabo cada una de las operaciones que deben realizar cada uno. La técnica de desarrollo será la de refinamientos sucesivos. Crear un prototipo lo antes posible e ir añadiendo funcionalidades al sistema paulatinamente.

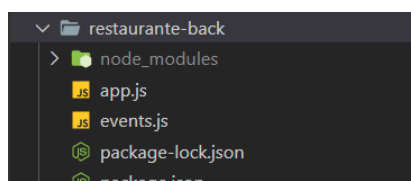
3.1 Codificación

En esta sección se tratará como se han plasmado las ideas en funcionalidades con código. Esta es la fase en la que se realiza el desarrollo del sistema.

3.1.1 Estructura de Carpetas



Estructura de carpetas del front



Estructura de carpetas del back

El archivo **app.js** se usa para crear el servidor web con NodeJS, en el se realiza la conexión a la base de datos y contiene también la configuración de Express. En el también se importan y cargan las rutas después de crearlas.

El archivo **events.js** es el controlador. En el se programan las acciones y operaciones sobre la base de datos. También se definen ahí las rutas a las que responderá nuestra aplicación y en ellas se encontrará la lógica a ejecutar.

El archivo **package.json** contiene todas las librerías y dependencias instaladas en el proyecto. Es el archivo de configuración principal del proyecto y debe encontrarse en la raíz del mismo. En el debe estar reflejado el nombre del proyecto, versión, descripción, scripts, autor, tipo de licencia y algo muy importante las dependencias.

EL archivo **angular.json** contiene configuración sobre el framework.

Gestión de restaurante

El archivo **tsconfig.json** contiene configuración sobre la estructura de del proyecto.

La carpeta **e2e**, cuyo nombre es "end to end", contiene una serie de ficheros que se encargaran de realizar test automáticos, como si un usuario real interactuara con nuestra app.

La carpeta **node_modules** contiene todas las dependencias del proyecto.

En el archivo **.editorconfig** se encuentra la configuración del editor de código.

El archivo **tslint.json** es el linter de TypeScript, se usa para mantenibilidad y sostenibilidad del código.

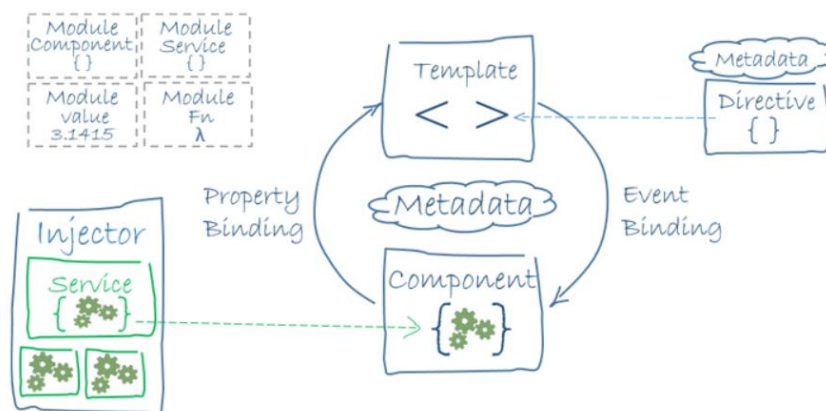
También se puede observar la carpeta **assets**, que contiene los archivos estáticos, y **environments**, las variables de entorno, y muchos otros archivos que forman parte de nuestra aplicación.

Al crear el proyecto se genera, automáticamente, una carpeta llamada **app** la cual tiene un componente y una vista simple como demostración. Esta carpeta se mantiene en el proyecto para poder orquestar la aplicación como el módulo principal de la aplicación.

También se pueden observar las carpetas **components** (contiene los componentes), **models** (aquí se encuentran las interfaces y enumeraciones necesarios), **pages** (donde se agrupan las páginas que se incluyen en la app) y **services** (donde se incluyen todos los servicios que se han ido creando).

3.1.1.1 Front-end

Angular es un framework dedicado a construir aplicaciones web en el cliente, lo que quiere decir que es el navegador el encargado de llevar el comportamiento y renderizado de nuestra "webapp".



Angular architecture - Imagen from angular.io

Diagrama del funcionamiento de la arquitectura de Angular

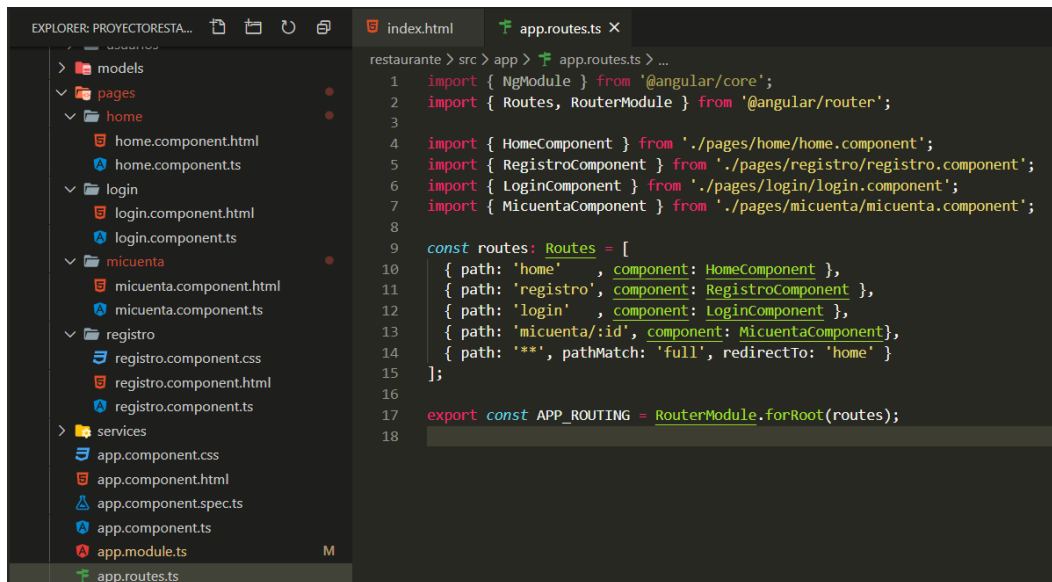
Gestión de restaurante

Para entender este esquema, y por tanto, el funcionamiento del proyecto, se ha dividido la explicación de los elementos que componen Angular en diferentes secciones.

3.1.1.1.1 Páginas y enrutamiento

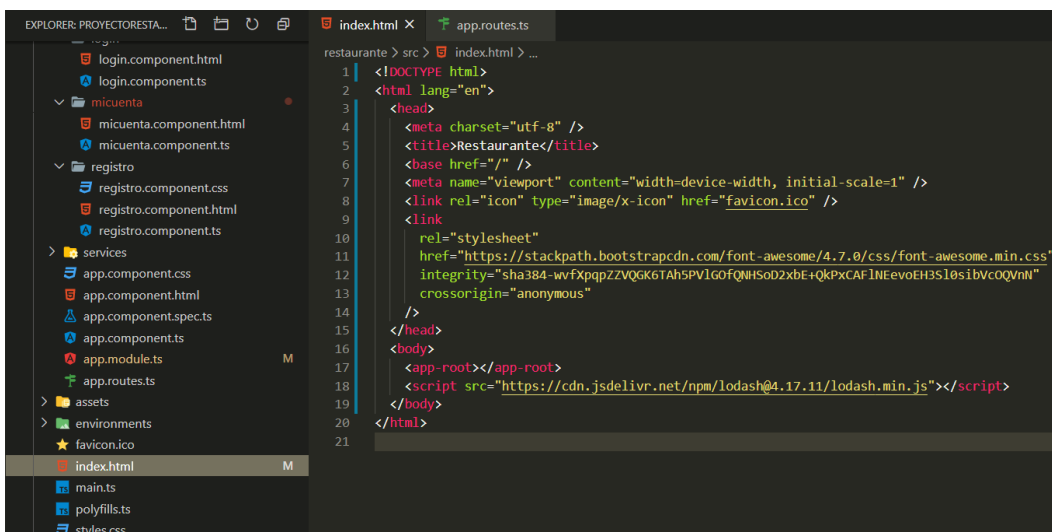
Una de las responsabilidades de las que se hace cargo el front es la de procesar las rutas y determinar cuál será la vista que se deba mostrar en cada dirección. **AppRoutingModule** importa, configura y exporta al **RouterModule**. Y a su vez **AppModule** al importar a **AppRoutingModule** dispone de todo lo necesario para realizar el enrutado. el **RouterModule** expone un par de métodos de configuración. Se llaman **forRoot(routes: Routes)** y **forChild(routes: Routes)** y se usan a nivel raíz o todas las demás situaciones respectivamente.

Ambos reciben una estructura que mantiene un array de rutas y las instrucciones a ejecutar cuando dichas rutas se activen. Las rutas pueden ser estáticas o usar comodines, tal y como se puede observar en el archivo `app.routes.ts` del proyecto.



```
restaurant > src > app > app.routes.ts > ...
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3
4  import { HomeComponent } from './pages/home/home.component';
5  import { RegistroComponent } from './pages/registro/registro.component';
6  import { LoginComponent } from './pages/login/login.component';
7  import { MicuentaComponent } from './pages/micuenta/micuenta.component';
8
9  const routes: Routes = [
10   { path: 'home', component: HomeComponent },
11   { path: 'registro', component: RegistroComponent },
12   { path: 'login', component: LoginComponent },
13   { path: 'micuenta/:id', component: MicuentaComponent },
14   { path: '**', pathMatch: 'full', redirectTo: 'home' }
15 ];
16
17 export const APP_ROUTING = RouterModule.forRoot(routes);
18
```

Archivo `app.routes.ts`



```
restaurant > src > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8" />
5    <title>Restaurante</title>
6    <base href="/" />
7    <meta name="viewport" content="width=device-width, initial-scale=1" />
8    <link rel="icon" type="image/x-icon" href="favicon.ico" />
9    <link
10     rel="stylesheet"
11     href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
12     integrity="sha384-wvXpqpZVZVQ6Tah5PV1GofQNH502xBe+QkPxCAF1NEvoEH3S10sibvcoQvN"
13     crossorigin="anonymous"
14   />
15 </head>
16 <body>
17   <app-root></app-root>
18   <script src="https://cdn.jsdelivr.net/npm/lodash@4.17.11/lodash.min.js"></script>
19 </body>
20 </html>
21
```

Ejemplo de uso de `<app-root>`.

Gestión de restaurante

El caso es que necesitamos mostrar un componente u otro en función de una ruta. Por tanto, se utiliza el **<router-outlet>**. Este es un componente que viene con el RouterModule y actúa como un contenedor dinámico, incrustando el componente adecuado para cada ruta. Otra directiva que se ha utilizado es la directiva **routerLink**. Una directiva es una extensión del HTML propia de Angular. Se emplea como si fuese un atributo de cualquier elemento y durante la compilación genera el código estándar necesario para que lo entiendan los navegadores.

```
index.html  app.component.html X  app.routes.ts

restaurante > src > app > app.component.html > ...
1  <app-navbar></app-navbar>
2  <main>
3  <router-outlet></router-outlet>
4  </main>
5  <app-footer></app-footer>
```

Código del template al que hace referencia la etiqueta `<app-root>`. Ejemplo de uso del componente `<router-outlet>`.

```
index.html  app.component.html  navbar.component.html X  app.routes.ts

restaurante > src > app > components > shared > navbar > navbar.component.html > ...
1  <nav class="navbar navbar-expand-lg navbar-light bg-light p-3 px-md-4 mb-3 border-bottom box-shadow">
2
3      <a class="navbar-brand" [routerLink]='["home"]'>Company name</a>
4      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavDropdown"
5          aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
6          <span class="navbar-toggler-icon"></span>
7      </button>
8
9      <div class="my-2 my-md-0 mr-md-3 navbar collapse navbar-collapse" id="navbarNavDropdown">
10         <div class="d-flex justify-content-xs-center justify-content-md-start flex-column flex-md-row ">
11             <a class="p-2 text-dark" href="#menus">Menús</a>
12             <a class="p-2 text-dark" href="#carta">La Carta</a>
13             <a *ngIf="!isLoggedIn || isClient" class="p-2 text-dark" href="#contacto">Contacto</a>
14             <a *ngIf="isEmployee" class="p-2 text-dark" href="#pedidos">Pedidos</a>
15             <a *ngIf="isAdmin" class="p-2 text-dark" href="#usuarios">Usuarios</a>
16             <a *ngIf="isLoggedIn" class="p-2 text-dark" href="#">Mi cuenta</a>
17         </div>
18
19         <div class="d-flex align-self-end justify-content-md-end ml-auto col-12 col-md-8 pl-0">
20             <a *ngIf="!isLoggedIn" class="btn btn-outline-primary mx-2" [routerLink]='["login"]'>Iniciar sesión</a>
21             <a *ngIf="!isLoggedIn" class="btn btn-outline-secondary" [routerLink]='["registro"]'>Registrarse</a>
22
23             <a *ngIf="isLoggedIn" class="btn btn-outline-danger mx-2" (click)="logout()">Cerrar sesión</a>
24         </div>
25     </div>
26 </nav>
```

En el template del navbar se puede observar como el atributo href queda relegado a la navegación en la propia página y los cambios de ruta se hacen mediante routerLink

En concreto esta directiva, que también viene en el módulo routerModule, se usa en sustitución del atributo estándar href. Instruye al navegador para que no solicite la ruta al servidor, sino que el propio código local de JavaScript se encargue de procesarla.

```
22  public logout(): void {
23      this.usuSrv.logout()
24      .subscribe(
25          () => this.router.navigate(['/login']),
26          (error) => console.error(error)
27      );
28  }
```

En el botón de cerrar sesión la redirección se realiza desde el componente porque además de cambiar la ruta necesitamos eliminar los datos de la sesión

Hay situaciones en las que, dada una ruta, se quiere enviar al usuario a otra página, quizá respondiendo a problemas o acciones inesperadas del usuario. Para eso se ha utilizado, tal y como se puede observar el archivo app.routes.ts que se muestra al principio de este apartado. el comando de la configuración de ruta **redirectTo**. Y se asigna a todas aquellas rutas desconocidas

usando un el comodín **. Esta entrada especial se sitúa al final del array de las rutas conocidas. Angular evalúa la ruta actual contra todas las disponibles de arriba abajo y elige la primera que coincida.

3.1.1.1.2 Componentes

Un componente controla un espacio de la pantalla, que se denomina vista. Es, en realidad, una clase de JavaScript (ES6) con el decorador **@Component** que incluye las propiedades y métodos disponibles para su template. Se debe incluir en esta clase todo lo referente al controlador de la vista y abstraer todos los demás métodos en servicios que serán inyectados posteriormente. Cada componente contiene una combinación de un archivo .html con un .ts y algunas veces un .css para crear un elemento con características propias tanto de comportamiento como de apariencia que se puede mostrar en un navegador.

Esto resulta de gran utilidad en el software que estamos desarrollando, ya que, como busca ser una “plantilla” que posteriormente pueda ser reutilizada y modificada al gusto de cada cliente nos conviene tener cada elemento de la página separado, con sus propios estilos, sin que se vean afectados por el estilo del resto de componentes.

El **template** es el encargado de definir la vista del componente. En el caso de Angular se trata de un HTML tradicional pero edulcorado con una serie de expresiones y directivas que mejoran el comportamiento de este y facilita el trabajo. En el template de un componente podemos encontrar, además de los tags normales de HTML, otros elementos distintos que utiliza Angular como ***ngIf**, ***ngFor**, **(event)**, **[property]**, **<component>** o la **interpolación** y los **pipes**.

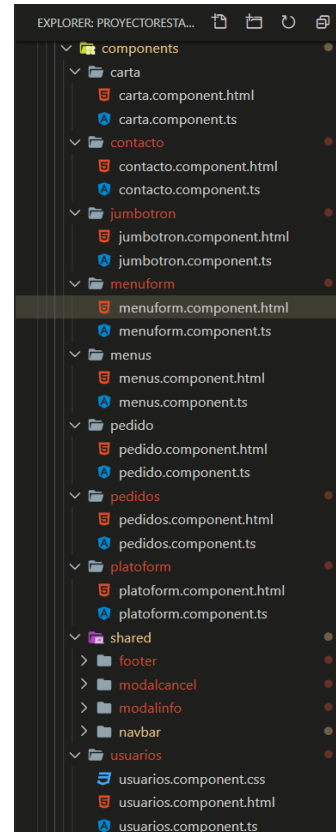


Imagen de los componentes del proyecto.

```
1 <h1 class="text-center">Menú</h1>
2 <button *ngIf="isAdmin" class="btn btn-primary ml-2" type="button" data-toggle="collapse"
3   data-target="#collapseExample" aria-expanded="false" aria-controls="collapseExample">
4   Crear menú</button></h1>
5 <div class="collapse" id="collapseExample">
6   <div class="card card-body">
7     <app-menuform></app-menuform>
8   </div>
9 </div>
10
11 <div class="d-flex flex-row flex-wrap justify-content-xs-around justify-content-md-between justify-content-lg-around">
12   <div *ngFor="let menu of menus; let i = index;" class="card mx-auto my-4" style="width: 18rem;">
13     <img [src]="menu.imagen" class="card-img-top" alt="...">
14     <div class="card-body">
15       <h5 class="card-title">{{ menu.nombre_menu }}</h5>
16       <p class="card-text">{{ menu.descripcion }}</p>
17     </div>
18     <ul class="list-group list-group-flush">
19       <li *ngFor="let plato of menu.platos" class="list-group-item">{{ plato.nombre_plato }}</li>
20     </ul>
21     <div class="card-body col-12">
22       <div *ngIf="isAdmin" class="row">
23         <div class="col-6">
24           <button class="btn btn-secondary" type="button" data-toggle="collapse"
25             data-target="#collapseExample" aria-expanded="false" aria-controls="collapseExample">
26             Editar menú</button>
27         </div>
28         <div class="custom-control custom-switch col-6 my-auto">
29           <input type="checkbox" class="custom-control-input" [id]="menu.i" [checked]="menu.habilitado">
30           <label class="custom-control-label" [for]="menu.i">Habilitado</label>
31         </div>
32       </div>
33     </div>
34     <div *ngIf="isAdmin" class="row">
35       <h5 class="mx-auto">{{ menu.precio }} €</h5>
```

Fragmento del template del componente 'menus'. En el podemos observar el uso de algunos de los componentes mencionados anteriormente.

3.1.1.1.3 Servicios

En las buenas prácticas de Angular se recomienda liberar a los componentes de cualquier lógica no relacionada con la vista. Se debe mover toda esa lógica, tal y como se ha mencionado anteriormente, a un servicio.

Básicamente un servicio es un proveedor de datos, que mantiene la lógica de acceso a los mismos y la operativa relacionada con el negocio y las cosas que se hacen con los datos dentro de una aplicación. Los servicios serán consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos. Para poder usar un servicio es necesario agregarlo a un módulo o componente. Algunos de sus propósitos son, entre otros, contener la lógica de negocio, clases para acceso a datos, encerrar funciones útiles. En este proyecto se han utilizado principalmente para invocar a un servidor HTTP para consumir una API. También es el mecanismo para compartir funcionalidad entre componentes ya que estas clases son perfectamente instanciables desde cualquier otro fichero que las importe

La librería **@angular/common/http** contiene el módulo **HttpClientModule** con el servicio inyectable **HttpClient**. Lo primero es importar dicho módulo. A partir de este momento sólo queda invocar los métodos REST en la propiedad `this.http`. Para cada verbo http tenemos su método en el servicio `HttpClient`. Su primer parámetro será la url a la que invocar. Estos métodos retornan un objeto **observable**. Los observables http han de consumirse mediante el método `subscribe` para que realmente se lancen. Dicho método `subscribe` admite hasta tres callbacks para responder a tres sucesos posibles: retorno de datos correcto, retorno de un error y señal de finalización.

Las comunicaciones entre navegadores y servidores son más lentas que las operaciones en memoria. Por tanto, deben realizarse de manera asíncrona para garantizar una buena experiencia al usuario.

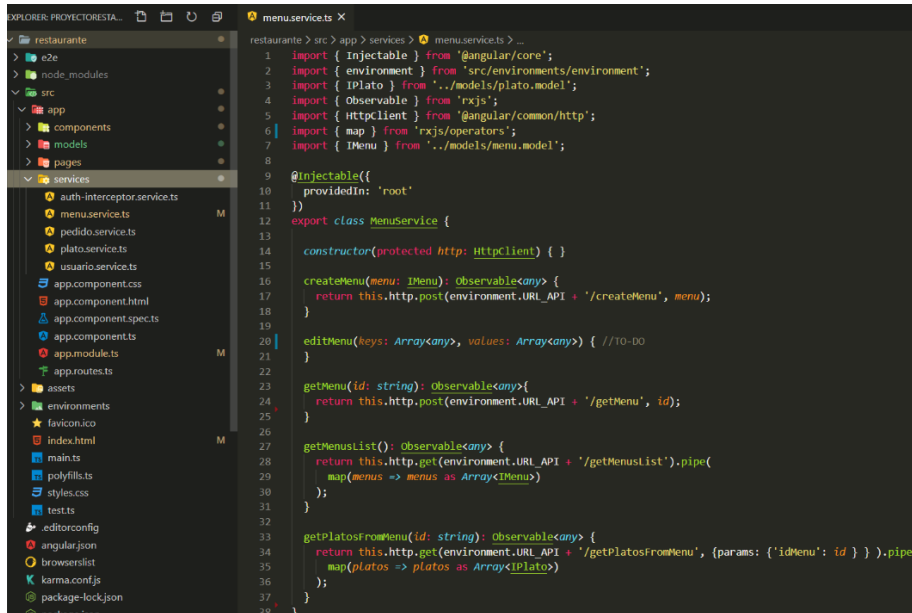
El patrón Observable fue implementado por Microsoft en la librería Reactive Extensions más conocida como **RxJs**. El equipo de Angular decidió utilizarla para el desarrollo de las comunicaciones asíncronas. Basta con escribir

```
import { Observable } from "rxjs/Observable"
```

para obtener la clase usada por Angular para observar la respuesta http. Toda operación asíncrona retornará una instancia observable a la cual habrá que subscribirse para recibir los datos o los errores cuando termine.

Los observables representan también una de las mejores formas de optimizar una aplicación, aumentando su rendimiento. El patrón observable no es más que un modo de implementación de la programación reactiva, que básicamente pone en funcionamiento diversos actores para producir los efectos deseados, que es reaccionar ante el flujo de los distintos eventos producidos. La programación reactiva es la programación con flujos de datos asíncronos.

Gestión de restaurante



Ejemplo de servicio. Se puede observar como se importa el servicio HttpClient, como se usan sus métodos y como devuelven un Observable.

En definitiva, un observable es aquello que se quiere observar, que será implementado mediante una colección de eventos o valores futuros. Un observable puede ser creado a partir de eventos de usuario derivados del uso de un formulario, una llamada HTTP,

un almacén de datos, etc. Mediante el observable es posible suscribirse a eventos que nos permiten hacer cosas cuando cambia lo que se esté observando.

Los métodos subscribe(), aunque vayan vacíos, son imprescindibles para que se ejecute la llamada. Esto puede resultar poco intuitivo, pero la realidad es que los observables Http de Angular sólo trabajan si hay alguien mirando.

Para este proyecto se ha separado la lógica de negocio en 4 servicios independientes en función de con qué elemento estén relacionados (usuarios, menús, platos o pedidos) y un servicio dedicado exclusivamente a la autenticación de los usuarios que es llamado antes de realizar cualquier otro método de los demás servicios.

Estos servicios después se llaman desde los demás componentes o archivos de 'utilidades' del proyecto.



Ejemplo de uso del servicio 'menu' en el componente 'menus'. Se puede ver como se importan los servicios, y la suscripción a los observables que devuelven sus métodos.

3.1.1.1.4 Formularios y seguridad

Validación de formularios

La validación es una pieza clave de la entrada de datos en cualquier aplicación. Es el primer frente de defensa ante errores de usuarios; ya sean involuntarios o deliberados. Validar correctamente los campos protege al proyecto de entradas maliciosas y provee también una mejor experiencia de usuario. La validación de campos reactivos es simple. Por defecto, Angular contiene validaciones para la gran mayoría de casos, pero también permite crear nuevas.

FormBuilder es un servicio del que han de depender los componentes que quieran desacoplar el modelo de la vista. Se usa para construir un formulario creando un **FormGroup**, (un grupo de controles) que realiza un seguimiento del valor y estado de cambio y validez de los datos.

El formulario se define como un grupo de controles. Cada control tendrá un nombre (**formControlName**) y una configuración. Esa definición permite establecer un valor inicial al control. Mientras tanto en la vista html este trabajo previo y extra que se realiza en el controlador se recompensa con una mayor limpieza en la vista. Lo único necesario será asignar por nombre el elemento html con el control Typescript que lo gestionará.

```

44 <input
45   [formControlName]="nombre"
46   type="text"
47   id="defaultRegisterFormFirstName"
48   class="form-control"
49   placeholder="First name"
50 />
51 </div>
52 <div class="col">
53   <!-- Last name -->
54   <input [formControlName]="apellido"
55     type="text"
56     id="defaultRegisterFormLastName"
57     class="form-control"
58     placeholder="Last name"
59   />
60 </div>
61 </div>
62
63 <!-- E-mail -->
64 <input
65   type="email" [formControlName]="email"
66   id="defaultRegisterFormEmail"
67   class="form-control mb-4"
68   placeholder="E-mail"
69 />
70
71 <!-- Direccion -->
72 <input
73   type="email" [formControlName]="direccion"
74   id="defaultRegisterFormEmail"
75   class="form-control mb-4"
76   placeholder="Dirección"
77 />

```

```

1 import { Component, OnInit } from '@angular/core';
2 import { IMenu } from 'src/app/models/menu.model';
3 import { MenuService } from 'src/app/services/menu.service';
4 import { Utils } from 'src/app/models/Utils';
5 import { UsuarioService } from 'src/app/services/usuario.service';
6 import { FormBuilder, FormGroup, FormControl, FormArray } from '@angular/forms';
7 import { IUsuario } from 'src/app/models/usuario.model';
8 import { PedidoService } from 'src/app/services/pedido.service';
9
10 @Component({
11   selector: 'app-pedido',
12   templateUrl: './pedido.component.html',
13   styles: [],
14 })
15 export class PedidoComponent extends Utils implements OnInit {
16   public form: FormGroup;
17   private opciones: Array<IMenu>;
18   private user: IUsuario;
19
20   constructor(
21     private fb: FormBuilder,
22     private mnuSrv: MenuService,
23     private usuSrv: UsuarioService,
24     private pdSrv: PedidoService
25   ) {
26     super(usuSrv);
27     this.opciones = new Array<IMenu>();
28   }
29
30   ngOnInit(): void {
31     this.user = this.userLogged;
32     this.cargaOpciones();
33   }
34
35   private initForm(): void {
36     this.form = this.fb.group({
37       nombre: [this.user.nombre, []],
38       apellido: [this.user.apellido, []],
39       direccion: [this.user.direccion, []],
40       email: [this.user.email, []],
41       fecha: [[null], []],
42       telefono: [this.user.telefono, []],
43       pago: [null, []],
44       detalle_pedido: this.fb.array(this.inicialiceDetallePedido()),
45     });

```

Imagen del código del formulario que permite realizar un pedido. Concretamente del fragmento que solicita al usuario activo que facilite sus datos.

Componente 'pedido'. Se puede observar como se crea y se inicializa, con los datos del usuario activo precargados, el formulario.

De nuevo se dispone de distintas sobrecargas que permiten resolver limpiamente casos sencillos de una sola validación, o usar baterías de reglas que vienen predefinidas como funciones en el objeto Validators del framework. Cuando un control incumple con alguna regla de validación, estas se reflejan en su propiedad errors que será un objeto con una

Gestión de restaurante

propiedad por cada regla insatisfecha y un valor o mensaje de ayuda guardado en dicha propiedad. La validación particular para cada control permite informar al usuario del fallo concreto. Lo que no queremos es llevar de vuelta la lógica a la vista; así que lo recomendado es crear una función auxiliar para mostrar los errores de validación.

```
public registrarse(): void {
  if (this.form.valid) {
    this.usuarioService.createUsuario(this.form.getRawValue())
      .subscribe((usu: IUserario) => this.usuario = usu,
        (error) => this.handleError(error));
  }
}

handleError(error: HttpResponse) {
  this.errorCode = error.error.code;
  if (this.errorCode === 'ER_DUP_ENTRY') {
    this.msg = 'El email ya está en uso';
  } else {
    this.msg = error.error.message;
  }
  this.error = true;
}
```

Aquí se puede observar un ejemplo del manejo de errores del formulario de registro

Gestión de credenciales

Aprovechando el conocimiento adquirido sobre Interceptores y Observables se ha montado un pequeño sistema de gestión de credenciales. La idea es detectar respuestas a llamadas no autenticadas y redirigir al usuario a nuestra pantalla de registro.

```
52 login(user: any): Observable<any> {
53   return this.http.post(environment.URL_API + '/login', user).pipe(
54     map((loggedUser: IUserario) => {
55       sessionStorage.setItem('token', loggedUser.sessionId);
56       return this.loggedUser = loggedUser;
57     })
58   );
59 }
60
61 logout(): Observable<any> {
62   return this.http.get(environment.URL_API + '/logout',{
63     withCredentials: true}).pipe(map(res => {
64     this.loggedUser = null;
65     sessionStorage.removeItem('token');
66     return res;
67   }));
68 }
69
70 getActiveUser(): Observable<any> {
71   if (sessionStorage.getItem('token')) {
72     return this.http.post(environment.URL_API + '/activeUser', {token: sessionStorage.getItem('token')});
73   } else {
74     return this.logout();
75   }
76 }
```

Si el usuario se registra correctamente recibiremos un token (un id de sesión) que lo identifica. Lo que haremos será guardarlo en una cookie de sesión y usarlo en el resto de las llamadas.

De la misma manera, cuando el usuario cierre sesión se eliminará la cookie de sesión.

Fragmento del servicio 'usuario'. Podemos ver como se crea, se recupera y se elimina la información de la cookie de sesión.

```
1 import { Injectable } from '@angular/core';
2 import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpResponse } from '@angular/common/http';
3 import { Observable, throwError } from 'rxjs';
4 import { catchError } from 'rxjs/operators';
5 import { Router } from '@angular/router';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class AuthInterceptorService implements HttpInterceptor {
11
12   constructor(private router: Router) {}
13
14   intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
15     const token: string = sessionStorage.getItem('token');
16
17     let request = req;
18
19     if (token) {
20       request = req.clone({
21         headers: {
22           authorization: 'Bearer ' + token
23         }
24       });
25     }
26
27     return next.handle(request).pipe(
28       catchError((err: HttpResponse) => {
29         if (err.status === 401) {
30           sessionStorage.removeItem('token');
31           this.router.navigateByUrl('/login');
32         }
33         return throwError(err);
34       })
35     );
36   }
37 }
```

Se ha creado un servicio para interceptar los fallos de seguridad. En este caso interesan las respuestas con error 401. Se ha empleado el Router de Angular para obligar al usuario a visitar la página de inicio de sesión cuando esto ocurra.

Servicio para la autenticación de usuarios implementando la interfaz `HttpInterceptor`

Gestión de restaurante

```
auth-interceptor.service.ts  app.module.ts X
restaurante > src > app > app.module.ts > AppModule
29 import { ModalCancelComponent } from './components/modal-cancel.component';
30 import { ModalInfoComponent } from './components/modal-info.component';
31
32 @NgModule({
33   declarations: [
34     AppComponent,
35     HomeComponent,
36     LoginComponent,
37     RegistroComponent,
38     NavbarComponent,
39     FooterComponent,
40     MenusComponent,
41     JumbotronComponent,
42     CartaComponent,
43     ContactoComponent,
44     PedidoComponent,
45     MicuentaComponent,
46     PedidosComponent,
47     MenuformComponent,
48     PlatoformComponent,
49     UsuariosComponent,
50     ModalInfoComponent,
51     ModalCancelComponent
52   ],
53   imports: [
54     BrowserModule,
55     CommonModule,
56     APP_ROUTING,
57     ReactiveFormsModule,
58     NgSelectModule,
59     FormsModule,
60     HttpClientModule,
61     NgbModule
62   ],
63   providers: [
64     UsuarioService,
65     {
66       provide: HTTP_INTERCEPTORS,
67       useClass: AuthInterceptorService,
68       multi: true
69     },
70   ],
71   bootstrap: [AppComponent]
72 })
73 export class AppModule { }
```

Por último, se proveerá el servicio creado en el archivo `app.module.ts`, para que se accese por todos los componentes y servicios de la app.

De esta manera, antes de realizar cualquier petición a la API se añadirá el id de sesión a modo de token en la cabecera de todas las llamadas.

Con todo esto se comprueba que haya una sesión activa y, además, que la id de sesión que se guarda en el navegador del usuario se corresponda con la id de sesión que se guarda en el servidor, evitando así posibles robos de sesión.

En definitiva, los interceptores son una manera sencilla de manipular las peticiones y respuestas, evitando así realizar dicha modificación en cada servicio que realizamos manualmente. Esto nos ayuda a tener un código limpio y mantenible.

Se observa cómo se provee al módulo principal con el servicio creado (línea 65- línea 69)

3.1.1.2 Back-end

El **front-end** muestra la información, pero es el **back-end** quien realmente trabaja con esta y filtra todas las acciones, enruta los accesos y vigila que todo fluya como tenga que fluir evitando accesos no permitidos según el usuario.

En este apartado se tratará todas las funciones relacionadas con el servidor. Tanto la conexión con la base de datos como la lógica de negocio.

En el back se ha construido un API REST usando para ello la tecnología de NodeJS y MySQL como base de datos. Se ha optado por esta solución a causa de la restricción de horas para la realización de este proyecto, ya que brinda la facilidad de escalar y hacer crecer el sistema de una manera muy sencilla. La arquitectura REST expone a los clientes una interfaz uniforme desde el servidor ya que todos los recursos del servidor tienen un nombre en forma de URL o hipervínculo y toda la información se intercambia a través del protocolo HTTP.

```
1  const express = require('express');
2  const session = require('express-session');
3  const cors = require('cors');
4  const bodyParser = require('body-parser');
5  const mysql = require('mysql');
6  const events = require('./events');
7  const expressSanitizer = require('express-sanitizer');
8  const connection = mysql.createConnection({
9    host      : 'localhost',
10   user      : 'restaurante',
11   password  : 'FbrxJLPFMSpLk5RP',
12   database  : 'restaurante'
13 });
14 connection.connect();
15 const port = process.env.PORT || 8080;
16 const app = express({host:'localhost'})
17   .use(session({
18     secret: 'Shh, its a secret!',
19     saveUninitialized: false,
20     resave: true,
21     cookie: {
22       httpOnly: true,
23       maxAge: 60000
24     }
25   }))
26   .use(cors({origin: [
27     "http://localhost:4200"
28   ], credentials: true}))
29   .use(bodyParser.json())
30   .use(expressSanitizer())
31   .use(checkAuth)
32   .use(events.router(connection));
```

Archivo de configuración del servidor

En este archivo se pueden observar algunas de las dependencias requeridas, la conexión con la base de datos, el puerto que se va a usar, la configuración de la app y, por último, la función `.listen()`, que devuelve un objeto `http.Server`, es decir, la instancia del servidor HTTP.

3.1.1.2.1 Seguridad

La seguridad es un tema de suma importancia en la actualidad. En nuestro caso, al no permitir pagos online ni exigir identificación mediante algún documento oficial, podemos afirmar que no se guardan datos extremadamente sensibles de los usuarios y, por tanto, es poco probable que haya vulnerabilidades de seguridad graves. Aun así, se han implementado unas cuantas prácticas para garantizar un mínimo de confianza:

- Se han utilizado cookies de forma segura mediante el middleware `express-session`, que almacena los datos de sesión en el servidor; sólo guarda el ID de sesión en la propia cookie, no los datos de sesión. Se han establecido las siguientes opciones de cookies para mejorar la seguridad:
 - `httpOnly` - Garantiza que la cookie sólo se envíe a través de HTTP(S), no a través de JavaScript de cliente, para la protección contra ataques de scripts entre sitios.
 - `expires` - Se utiliza para establecer la fecha de caducidad de las cookies persistentes.

Se puede observar en el archivo de configuración mostrado más arriba.

- Se ha filtrado y saneado la entrada de usuario, usando `express-sanitizer`, para proteger ataques de scripts entre sitios (XSS) e inyección de mandatos.
- Para defenderse de ataques de inyección SQL se ha hecho uso de consultas parametrizadas.
- Se ha limitado el tamaño del cuerpo de las peticiones. Al igual que se controla el tipo de los datos de entrada, también se ha limitado su tamaño máximo para evitar ataques DoS. Para eso se ha utilizado `body-parser`, otro de los middlewares mas usados, cuya función es llevar a cabo el parseo del cuerpo de las peticiones. Por defecto tiene un tamaño máximo del body de 100Kb.
- Se encriptan las contraseñas para mayor seguridad con la ayuda de `bcrypt`.
- Se ha implementado una autenticación mediante token. Si bien es cierto que se ha hecho de manera manual, sigue siendo un método efectivo.
- Se ha añadido una restricción de acceso por nivel a los distintos métodos de la API.

```
const register = (req, res, next) => {
  nombre = req.sanitize(req.body.nombre);
  apellido = req.sanitize(req.body.apellido);
  email = req.sanitize(req.body.email);
  password = bcrypt.hashSync(req.body.password, 4);
  telefono = req.sanitize(req.body.telefono);
  direccion = req.sanitize(req.body.direccion);
  habilitado = req.sanitize(req.body.habilitado);
  nivel = req.sanitize(req.body.nivel);
  db.query(
    "INSERT INTO usuario (nombre, apellido, email, password, telefono, direccion, habilitado, nivel) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
    [nombre, apellido, email, password, telefono, direccion, habilitado, nivel],
    (error, result) => {
      if (error) {
        res.status(500).json({ code: error.code, message: error.sqlMessage });
      } else {
        res.status(200).json(result);
      }
    }
  );
};
```

Ejemplo de sanitización de datos, encriptación de contraseña y consulta parametrizada en la función de registro de usuarios.

```
function checkAuth (req, res, next) {
  authorized = ['/login', '/register', '/activeUser', '/getPlatos', '/getMenusList', '/getPlatosFromMenu'];
  headers = JSON.parse(JSON.stringify(req.headers));
  if (authorized.includes(req.url.split('?')[0])) {
    console.log('authorization not required');
    next();
  } else if ((events.session() && headers.authorization !== events.session().id) || !req.session || !req.sessionID) {
    events.session() = undefined;
    req.session.destroy((err) => {
      if(err) {
        res.status(500).json(err);
      } else {
        res.status(401);
      }
      res.end();
    });
  } else {
    console.log('authorized');
    next();
  }
}
```

Método que comprueba la autenticación mediante token. Únicamente se realiza esta comprobación en los métodos que están restringidos a usuarios registrados, independientemente del nivel. En caso de error se destruyen los datos de la sesión si los hubieran y se envía al cliente el código 401 (acceso denegado)

Evidentemente quedarían decenas de medidas de seguridad que aún deberíamos tener en cuenta, pero es un buen comienzo como primer acercamiento de fácil aplicación.

3.1.1.3.2 Rutas

El controlador del servidor está formado por las rutas siguientes:

```
1  const express = require("express");
2  const bcrypt = require("bcrypt");
3  let db;
4  var sess;
5  function createRouter(dbConnection) {
6    const router = express.Router();
7    db = dbConnection;
8    router.post("/login", login);
9    router.post("/register", register);
10   router.post("/activeUser", activeUser);
11   router.get("/usersList", usersList);
12   router.get("/logout", logout);
13   router.post("/createPlato", createPlato);
14   router.get("/getPlatos", getPlatos);
15   router.post("/createMenu", createMenu, deletePlatosMenu, addPlatosMenu);
16   router.get("/getMenusList", getMenusList);
17   router.get("/getPlatosFromMenu", getPlatosFromMenu);
18   router.post("/createPedido", createPedido, addDetalle);
19   router.get("/getPedidosList", getPedidosList);
20   router.get("/getDetalleFromPedido", getDetalleFromPedido);
21   router.post("/updateEstado", updateEstado);
22   router.post("/enableMenu", enableMenu);
23   router.post("/enablePlato", enablePlato);
24   router.post("/enableUser", enableUser);
25   router.post("/updateNivel", updateNivel);
26   router.post("/updatePlato", updatePlato);
27   router.post("/updateMenu", updateMenu);
28   router.post("/updateUser", updateUser);
29   router.post("/updatePwd", updatePwd);
30   return router;
31 }
```

Rutas del controlador

Cada ruta se corresponde con un método, o una sucesión de ellos, del controlador. Estas rutas se exportan para poder ser añadidas a la configuración del servidor en el archivo app.js mostrado anteriormente.

3.1.1.2.3 Sesiones

Como se ha comentado anteriormente para el uso de sesiones se ha utilizado el middleware express-session. En el controlador se ha creado una variable global (sess), a la cual, en caso de iniciar sesión correctamente, se le añade la información necesaria. Además, al iniciar sesión, se envía al cliente el objeto Usuario y se añade el id de sesión, para poder incluirla en las posteriores peticiones al servidor. Al cerrar la sesión se destruyen los datos de sesión y se le asigna el valor undefined a la variable global sess.

```
52 const login = (req, res, next) => {
53   db.query(
54     "SELECT * FROM usuario WHERE email = ?",
55     [req.body.email],
56     (error, result) => {
57       if (error) {
58         req.session.authenticated = false;
59         res.status(500).json({ code: error.code, message: error.sqlMessage });
60       } else if (result.length === 0) {
61         res.status(404).json({ message: "Usuario no encontrado" });
62       } else {
63         let data = JSON.parse(JSON.stringify(result));
64
65         if (bcrypt.compareSync(req.body.password, data[0].password.toString())) {
66           sess = req.session;
67           sess.email = data[0].email;
68           sess.userId = data[0].id_usuario;
69           sess.nivel = data[0].nivel;
70           sess.authenticated = true;
71           sess.id = req.sessionID;
72           delete data[0].password;
73           data[0].sessionId = req.sessionID;
74           res.status(200).json(data[0]);
75         } else {
76           res.status(404).json({ message: "Contraseña incorrecta" });
77         }
78       }
79     }
80   );
81 };
82 const logout = (req, res, next) => {
83   sess = undefined;
84   req.session.destroy((err) => {
85     if (err) {
86       res.status(500).json(err);
87     } else {
88       res.status(200);
89     }
90     res.end();
91   });
92 };
```

Funciones de inicio y cierre de sesión del servidor.

En caso de intentar acceder a algún método restringido por nivel o sin autenticar se ejecutaría la función logout por seguridad.

3.2 Implantación y configuración de la aplicación

Para poder levantar la aplicación en local, el primer paso sería clonar el repositorio de GitHub. Una vez tenemos una copia del proyecto en local deberíamos instalar los servidores y levantarlo. Para ello se tiene que descargar Node.js desde su página oficial. Una vez descargado el archivo hay que seguir con la instalación con la instalación, que no es más que seguir el wizard y dar Next. Mencionar que conjuntamente con Node.js se va instalar el gestor de paquetes NPM.

```
PS C:\Users\Bea\Documents\ProyectoRestaurante\restaurante> node -v
v12.16.3
PS C:\Users\Bea\Documents\ProyectoRestaurante\restaurante> npm -v
6.14.4
PS C:\Users\Bea\Documents\ProyectoRestaurante\restaurante>
```

Para verificar que la instalación se ha realizado correctamente hay que ejecutar estos comandos desde la consola.

Desplegar la aplicación

En lo referente al front, se recomienda instalar angular CLI, para ello hay que ejecutar el comando **npm install -g @angular/cli**.

```
PS C:\Users\Bea\Documents\ProyectoRestaurante\restaurante> ng version
Your global Angular CLI version (9.1.4) is greater than your local
version (9.1.3). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".

Angular CLI
-----
Angular CLI: 9.1.3
Node: 12.16.3
OS: win32 x64

Angular: 9.1.3
... animations, cli, common, compiler, compiler-cli, core, forms
... language-service, platform-browser, platform-browser-dynamic
... router
Ivy Workspace: Yes

Package                          Version
-----
@angular-devkit/architect        0.901.3
@angular-devkit/build-angular    0.901.3
@angular-devkit/build-optimizer  0.901.3
@angular-devkit/build-webpack    0.901.3
@angular-devkit/core             9.1.3
@angular-devkit/schematics       9.1.3
@angular/localize                9.1.6
@ngtools/webpack                 9.1.3
@schematics/angular              9.1.3
@schematics/update                0.901.3
rxjs                             6.5.5
typescript                       3.8.3
webpack                          4.42.0
```

Una vez a terminado la instalación se debe ir a la carpeta del proyecto (la que corresponde al front) y ejecutar **ng serve** para desplegar la aplicación.

Para levantar el servidor simplemente habría que ir a la carpeta del back y ejecutar el comando **npm start**.

Ejecutando
'ng version'
se puede
comprobar
que la
instalación
ha sido
correcta.

Base de datos

Para poder acceder a la base de datos se debe instalar XAMPP desde su página oficial, inicializar Apache y MySQL, acceder a phpMyAdmin e importar la base de datos incluida en el repositorio del proyecto.

3.3 Control de versiones

Se llama **control de versiones** a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración de este. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

En este caso se han usado únicamente dos ramas, *master* y *secundaria*. Las funcionalidades que se han ido añadiendo se subían a la rama *secundaria* mientras estaban siendo desarrolladas y, una vez comprobado que funcionaban correctamente y sin errores se hacía un merge a la rama *master*.

4. Conclusiones

4.1 Posibles mejoras

Al ser una herramienta de gestión que puede ser usada por múltiples restaurantes admite muchas posibles mejoras en función de las necesidades de cada uno. En mi opinión, tal vez la primera y más obvia sería permitir también pedir platos de la carta sin necesidad de que estén incluidos en un menú. Otra mejora muy útil sería incluir una plataforma de pago online. También restringir mejor para que fechas y horas hay disponibilidad para realizar pedidos, ya que actualmente se puede realizar el pedido para cualquier día en horario de comidas, quedando a la espera de que el propio restaurante confirme si tienen disponibilidad o no. Por último, para facilitar el trabajo al restaurante, sería interesante que al deshabilitar un plato de la carta se deshabiliten los menús que lo incluyan o, como mínimo, te notifique que ese plato forma parte un menú disponible.

4.2 Dificultades

Debido a las circunstancias excepcionales en las que nos encontramos y la reducción de horas destinadas a la realización del proyecto a casi la mitad, la dificultad más importante ha sido la gestión del tiempo y todo lo que ello conlleva. En un principio la idea de desarrollar este software con algún lenguaje / framework que no conociese resultaba muy interesante, pero, con la reducción de horas se ha convertido en todo un reto. Además, ha sido difícil tener que descartar determinados requisitos por falta de tiempo durante el desarrollo, ya que tal vez eliminar alguna funcionalidad podía llevar a que otras no funcionasen o que incluso la idea sobre la que se empezó este proyecto perdiera bastante sentido.

4.3 Logros

A pesar de las dificultades que mencionaba anteriormente he conseguido terminar el desarrollo a tiempo, a pesar de usar frameworks y softwares hasta ahora desconocidos para mí. Considero también que esta aplicación me ha sido de gran ayuda para ver de manera más cercana todo lo que implica y conlleva el desarrollo de una aplicación desde cero.

4.4 Resultado

Finalmente he conseguido el objetivo de realizar una aplicación útil y escalable que puede ser adaptada a más de un cliente con cambios relativamente sencillos.

Además, he conocido y aprendido nuevos lenguajes, frameworks, librerías y buenas prácticas de manera autodidacta que me pueden facilitar desarrollos futuros.

En conclusión, creo que se puede decir que se han logrado los objetivos iniciales y se han cumplido todos los requisitos que se exigían en la realización de este proyecto de manera satisfactoria.

5. Referencias y Bibliografía

- ❖ <https://angular.io/guide/architecture>
- ❖ <https://getbootstrap.com/docs/4.5/getting-started/introduction/>
- ❖ <https://nodejs.org/es/docs/>
- ❖ <https://expressjs.com/es/>
- ❖ <https://ng-select.github.io/ng-select#>
- ❖ <https://ng-bootstrap.github.io/#/home>
- ❖ <https://www.apachefriends.org/es/index.html>
- ❖ <https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>
- ❖ <https://medium.com/@mridu.sh92/a-quick-guide-for-authentication-using-bcrypt-on-express-nodejs-1d8791bb418f>
- ❖ <https://www.npmjs.com/package/bcrypt>
- ❖ <https://www.npmjs.com/package/express-sanitizer>
- ❖ <https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a>