Carta Online

Muestra lo que puedes ofrecer, no lo que te falte.

Nombre del alumno o de la alumna: Samuel Arnaiz de la Rosa

Curso académico: 2º DAM

Tutora/Tutor del proyecto: Víctor Colomo

ÍNDICE PAGINADO

1.RESUMEN

El presente trabajo de fin de grado desarrolla una implementación novedosa en el mundo de la hostelería, pretende dar una funcionalidad extra a los programas de hostelería ya existentes, este es la impresión de la carta acorde al stock que se encuentra disponible en el negocio. Esto permite que clientes y camareros tengan una mejor experiencia a la hora de tomar o pedir la comanda.

Esto se ha logrado con dos servidores con la tecnología NODEJS, uno destinado al sistema de inicio de sesión y de registro de usuarios; el otro destinado a la conexión con el programa de hostelería al que se tenga acceso, en este caso se ha montado acorde al programa DSTnet el cual se ha cedido una demo por parte de la empresa propietaria. También lo forma una aplicación destinada al cliente la cual se ha creado con el framework de Angular con la cual se puede ver la carta y se puede iniciar sesión en el caso de ser el propietario o un trabajador.

No se logra todo el objetivo inicial, por limitaciones de la demo cedida, pero se crea una demo para mostrar las funcionalidades y la utilidad de la implementación una vez se pueda implementar en un programa completo de hostelería. Esta implementación puede comercializarse en un futuro y poder ser de gran ayuda para empresas y negocios del mundo de la hostelería.

2. JUSTIFICACIÓN DEL PROYECTO

El proyecto pretende facilitar el trabajo de camareros, comercios, empresarios y dueños de negocios relacionados con la hostelería y de cara al público. Existen aplicaciones en las que se puede gestionar el stock del negocio e incluso llegar a editar el menú que se muestra, estas serían los principales competidores del proyecto:

- **Miss Tipsi:** Software con varias opciones de negocio, en concreto la versión de hostelería incluye características como acceso en tiempo real a lo que pasa en el restaurante, desde cualquier dispositivo, pagos mediante QR, e incluso poder pedir la comanda a través del dispositivo del cliente.

- **Sighore:** Cuenta con varios programas, el más interesante se llama "BackOffice", incluye compras, proveedores, productos, precios...ofrece la opción de pedidos de compra y control de stock.

- **Foodeo:** Aplicación que permite pedidos a domicilio y para recoger, permite realizar pedidos de menús, los clientes pueden hacer el pedido desde la app.

Estas tres aplicaciones son las más destacadas en el software de restauración, todas ellas permiten una gran cantidad de opciones dentro de un negocio, pero ninguna de ellas tiene implementado nada parecido al proyecto de “Carta Online”, por lo que sería una gran aportación a este sector. Esta nueva implementación aportaría calidad a las aplicaciones.

3. OBJETIVOS

A. OBJETIVO GENERAL

El objetivo principal es proporcionar al negocio rapidez a la hora de tomar la comanda ya que no pierde tiempo el trabajador en comentar lo que está o no disponible para vender; por otra parte, también aporta al negocio en calidad de experiencia de los clientes, ya que no van a pedir algo que no esté disponible teniendo así que cambiar lo que en un principio habían elegido y en ese momento querían o deseaban.

B. OBJETIVOS ESPECÍFICOS

El proyecto, al ser novedoso, podría comercializarse en un futuro. Se puede comercializar como una nueva implementación en los programas ya existentes dentro del mercado, mencionados anteriormente. Con este proyecto valdría para presentarlo a modo de demo.

Este proyecto también lo motiva el ámbito personal, en mi experiencia como trabajador en el mundo de la hostelería, esta aplicación hubiese ahorrado mucho tiempo en la ejecución de mi trabajo y de mis compañeros. En muchos momentos me he visto en situaciones en las que el cliente pedía algo de la carta y daba la casualidad de que no quedaba, por lo que era pasar un momento incómodo y dejar al bar en mala situación.

4. DESARROLLO

FUNDAMENTACIÓN TEÓRICA:

Toda la aplicación consta de 3 “partes”:

1. **Servidor para sistema de LogIn:**

   API (se hablará más delante de las características) para registro y validación de usuarios. Peticiones disponibles a través del puerto 3000:

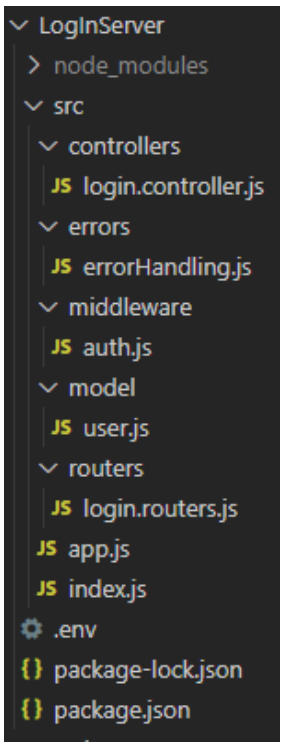
   - **“(POST) /login/{email}{password}”:** Por parámetros hay que pasarle el email y la contraseña del usuario que quiere loguearse, este devolverá un documento “. Json” con información si es correcto o no el login. También se implementa en esta petición el token (del cual se hablará más adelante), tanto enviarlo por la respuesta como guardándolo en la base de datos de “usuarios” para que ese token se vincule correctamente.

   - **“(POST)/registrar/{nombreUsuario}{nombre}{apellidos}{email}{password}”:** Hay que pasar por parámetro la información completa del nuevo usuario para que se haga un correcto registro, incluyendo el nombre de usuario, el nombre de la persona, sus apellidos, el mail y la contraseña. Si hay un usuario que ya esté registrado con esos datos (si coincide el email y el nombre de Usuario) devuelve un documento “.Json” con la información pertinente. Si el usuario ha sido registrado correctamente, este archivo “.Json” incluirá un “status: ‘ok’ ”.

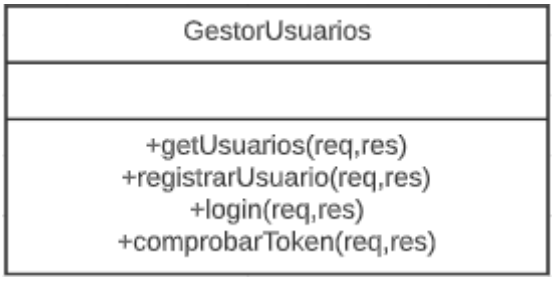
   - **“(POST) /auth/{token}”**: Valida el token pasado por parámetro.

- **"(POST) /authToken/{token}":** Realiza una conexión con la base de datos de usuarios para comprobar si hay un token vinculado con algún usuario. Devuelve el usuario que ha encontrado con ese token.

Estructura carpetas del Servidor para Login

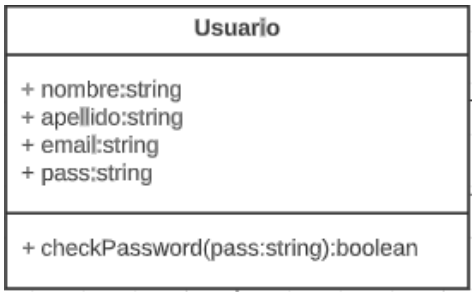


Esta es la estructura típica de un servidor de Node.js, en la parte inferior se encuentran las configuraciones de la aplicación, dentro de **"package.son"** y **"package-lock.json"** hay información de las dependencias necesarias para la aplicación e información de la misma, junto a scripts de ejecución. En el archivo **".env"** se encuentran variables de la aplicación como el puerto de escucha donde se monta el servidor y la Token_Key necesario para el uso de jwt(explicado más adelante). La carpeta **"node_modules"** incorpora todas las dependencias ya instaladas para poder usarlas en la aplicación. Todo el código de ejecución se encuentra dentro de la carpeta **"src"**, en orden se encuentra la carpeta **"controllers"** la cual incluye el documento **"login.controller.js"**, responsable de toda la lógica de conexión a BBDD y gestión de información recibida desde el lado cliente. Incluye una clase llamada "GestorUsuarios"



la cual incluye cuatro funciones útiles para todo el sistema de Login incluyendo para testing la función de conseguir todos los usuarios que existen("getUsuarios()"). A continuación, se encuentra la carpeta **"errors"** la cual incluye el documento **"errorHandling.js"** encargado de gestionar cuando el servidor lance un error 500 para que imprima por pantalla información acerca de este mismo.

Posterior a esta carpeta se encuentra **”middleware”**, con el documento **”auth.js”**, este es el responsable de validar un token de JWT pasado por parámetro. La siguiente carpeta **”model”** contiene el documento **”user.js”** el cual tiene



la definición del objeto Usuario, con los atributos y métodos que se pueden ver en el siguiente diagrama de clase. El método permite comparar la contraseña alojada en la Base de Datos con la que se pasa por parámetro.

La última carpeta llamada **“routers”** contiene el documento **“login.routers.js”**, este documento se ocupa de los endpoints para poder acceder a las funciones del controller, mencionado anteriormente. Estos endpoints son rutas de acceso para posteriormente desde la parte de cliente se pueda llamar a las funciones deseadas del controller, las cuales se han definido al principio del apartado.

Dentro de la carpeta **“src”** hay dos documentos muy importantes, por una parte **“index.js”**, este es el punto de entrada de ejecución de la aplicación, es decir, empieza a ejecutarse por este documento. Este importa el archivo **“app.js”** del cual ahora se hablará, y configura el puerto dónde el servidor escuchará las peticiones. El documento **“app.js”** instancia en el servidor tanto las dependencias necesarias para el correcto funcionamiento, como cors y express. También instancia el documento de las rutas disponibles y el de manejo de errores.

Dependencias

Para el servidor de login se han usado varias dependencias importantes a la hora de su desarrollo y funcionalidad:

- **Express:** proporciona características y herramientas robustas para desarrollar aplicaciones de backend escalables. Te ofrece el sistema de enrutamiento y características simplificadas para ampliar el framework con componentes y partes más potentes en función de los casos de uso de tu aplicación.

- **Cors:** este paquete proporciona un middleware de Conexión/Express que puede ser usado para habilitar CORS (Cross-origin resource sharing) que permite hacer peticiones desde una página web alojada en otro dominio.
- **Nodemon:** es una herramienta que ayuda al desarrollo, restaura automáticamente el servidor cada vez que se hace un cambio y lo configura.
- **Dotenv:** este paquete permite utilizar el archivo ".env" en el cual se alojan las variables de la aplicación.
- **Crypto-js:** dependencia que permite encriptar variables en diferentes formatos de encriptación.
- **Jsonwebtoken:** permite crear y utilizar funciones para la gestión del JWT (JSON Web Token)
- **Mysql2:** permite la consulta y conexión con Bases de Datos SQL.

Base de Datos

Este servidor se conecta, como se ha visto, a una Base de Datos SQL, en concreto, una llamada **"usuariostfg"**, usando la tabla **"usuarios"**. Estos son los campos de la tabla, necesario para el correcto registro de usuarios nuevos y para el inicio de sesión posterior:

Column	Type
apellidos	varchar(45)
email	varchar(45)
idusuario	int(11)
nombre	varchar(45)
nombreUsuario	varchar(45)
pass	varchar(100)
token	varchar(500)

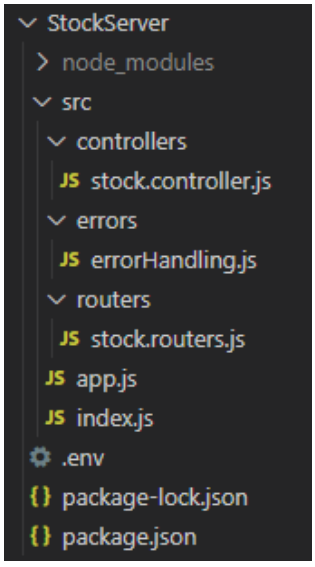
2. Servidor para control de stock:

API encargada de consultar en la Base de Datos del programa de hostelería los artículos y todo lo relacionado con ellos, ya sea stock, precio, características.

Peticiones disponibles a través del puerto 3001:

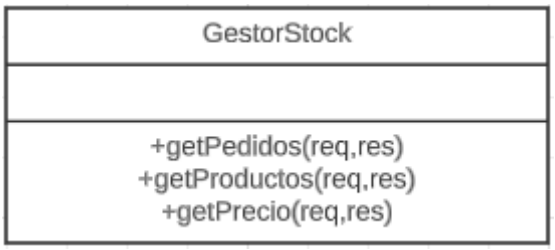
- **"(GET) /productos":** Realiza una conexión con la tabla **"Article"** para enviar todos los productos disponibles en el establecimiento. La respuesta enviada es un Array de Objetos en formato JSON.
- **"(GET) /precio":** Realiza conexión con la tabla **"ArticleTariff"** para sacar información del precio de los productos. La respuesta enviada es un Array de Objetos en formato JSON.
- **"(GET) /pedidos":** Realiza conexión con la tabla **"SalesOrderLine"** para adquirir la información de los pedidos que se realizan a través del programa de hostelería. La respuesta enviada es un Array de Objetos en formato JSON.

Estructura carpetas para el Servidor de Stock



Como en la aplicación de login, esta es la estructura típica de un servidor en Node.js, en la parte inferior se encuentran las configuraciones de la aplicación, dentro de **"package.son"** y **"package-lock.json"** hay información de las dependencias necesarias para la aplicación e información de la misma, junto a scripts de ejecución. En el archivo **".env"** se encuentran variables de la aplicación como el puerto de escucha donde se monta el servidor y la Token_Key necesario para el uso de jwt(explicado más adelante). La carpeta **"node_modules"** incorpora todas las dependencias ya instaladas para poder usarlas en la aplicación. Todo el código de ejecución se encuentra dentro de la carpeta

”src”, en orden se encuentra la carpeta **”controllers”** la cual incluye el documento **”stock.contorller.js”** responsable de toda la lógica de conexión a BBDD y gestión de información recibida desde el lado cliente. Incluye la clase ”GestorStock” la cual



permite sacar información de la aplicación de hostelería, consultando a la Base de Datos la cual se aloja de forma local en el dispositivo donde esté instalada.

A continuación, se encuentra la carpeta **”errors”** la cual incluye el documento **”errorHandling.js”** encargado de gestionar cuando el servidor lance un error 500 para que imprima por pantalla información acerca de este mismo. La siguiente carpeta llamada **“routers”** contiene el documento **“stock.routers.js”**, este documento se ocupa de los endpoints para poder acceder a las funciones del controller, mencionado anteriormente. Estos endpoints son rutas de acceso para posteriormente desde la parte de cliente se pueda llamar a las funciones deseadas del controller, las cuales se han definido al principio del apartado. Dentro de la carpeta **“src”** hay dos documentos muy importantes, por una parte **“index.js”**, este es el punto de entrada de ejecución de la aplicación, es decir, empieza a ejecutarse por este documento. Este importa el archivo **“app.js”** del cual ahora se hablará, y configura el puerto dónde el servidor escuchará las peticiones. El documento **“app.js”** instancia en el servidor tanto las dependencias necesarias para el correcto funcionamiento, como cors y express. También instancia el documento de las rutas disponibles y el de manejo de errores.

Dependencias

Las cuatro principales se usan también en el servidor de login.

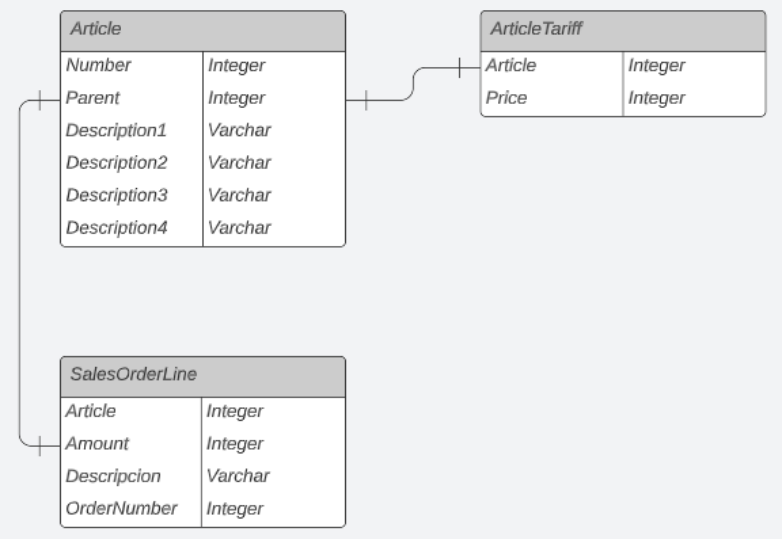
- **Express:** proporciona características y herramientas robustas para desarrollar aplicaciones de backend escalables. Te ofrece el sistema de enrutamiento y características simplificadas para ampliar el framework con

componentes y partes más potentes en función de los casos de uso de tu aplicación.

- **Cors:** este paquete proporciona un middleware de Conexión/Express que puede ser usado para habilitar CORS (Cross-origin resource sharing) que permite hacer peticiones desde una página web alojada en otro dominio.
- **Nodemon:** es una herramienta que ayuda al desarrollo, restaura automáticamente el servidor cada vez que se hace un cambio y lo configura.
- **Dotenv:** este paquete permite utilizar el archivo ".env" en el cual se alojan las variables de la aplicación.
- **Mssql:** dependencia necesaria para la conexión con la Base de Datos de la aplicación.

Base de Datos

Este servidor se conecta a una base de datos alojada en local, exactamente en SQL Server de Microsoft. Como esta Base de Datos contiene muchas tablas, y estas a su vez muchos campos, se va a mostrar un esquema de los campos y tablas usados para la aplicación. La relación es 1:1 entre el campo Number de la tabla **"Article"** y las otras dos tablas su respectivo campo Article.



- **Article:** proporciona toda la información sobre un artículo, el campo parent permite dividir en categorías todos los productos
- **ArticleTariff:** permite sacar la información del precio
- **SalesOrderLine** es donde se reflejan los pedidos que hay cerrados y el cliente ya ha pagado por ellos, permitiendo así llevar un control de los productos consumidos.

3. Lado Cliente:

Proyecto Angular dividido en componentes, se van a mostrar las funcionalidades de cada componente:

- **Header:** Parte superior de la página, en él se encuentran el logo de la aplicación y el nombre en la parte izquierda. En la parte derecha si el usuario no se ha logueado puede ver las opciones de **"LogIn"** o de **"Registrarse"**, las cuales le llevarán a los respectivos componentes. Si el usuario está registrado (comprobando el token, del cual se hablará más adelante), se mostrará un mensaje de bienvenida junto con el nombre del usuario y un botón para cerrar sesión, el cual eliminará la cookie y redirige al componente Home. Este componente se muestra en todas las páginas.

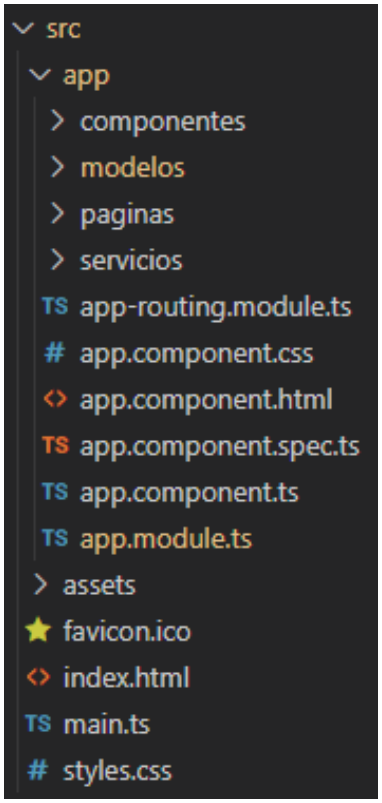
- **Footer:** Parte inferior de la página, en él se encuentra un logo centrado e información de copyright, carga en todas las páginas.

- **Login:** Componente encargado de iniciar sesión, formulario el cual pide correo y contraseña y botón para enviar el formulario. Este componente hace una petición al servidor de LogIn en la ruta **"(POST) /login"** y se queda a la espera de respuesta, hay dos opciones en este caso:
    - **Login incorrecto:** Se mostrará un mensaje de alerta: "Usuario o contraseña incorrectos".
    - **Login correcto:** Se guardará en cookies el token para poder mantener la sesión iniciada, después redireccionará al componente home.

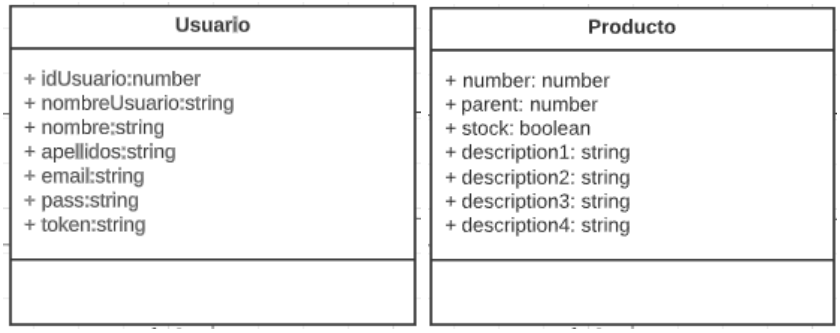
- **Register:** Se encarga de registrar a los usuarios, el formulario pide datos del cliente como nombre, apellidos...Pide repetir tanto el email como la contraseña para comprobar que el usuario no se ha equivocado al ponerlos, ya que son dos datos importantes a la hora de iniciar sesión posteriormente. El formulario valida todos los campos, mostrándose mensajes de error en el caso de que el usuario no los introduzca correctamente. Una vez introducidos los datos, el botón en la parte inferior manda una petición al servidor del Login con la ruta **"(POST) /registrar"**, como en login, hay dos opciones en la respuesta:
    - **Registro incorrecto:** Mostrará un mensaje de alerta con el mensaje correspondiente de por qué no se hay podido registrar al usuario
    - **Registro correcto:** Mostrará un mensaje de alerta con el mensaje "Registrado correctamente" y redireccionará al home.

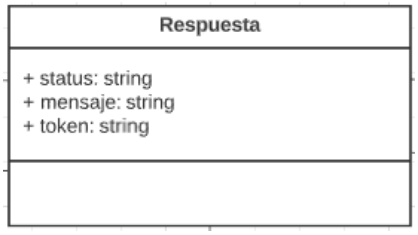
- **Home:** Página principal de la aplicación, esta hace las peticiones al servidor de stock proporcionando así información, tanto de los artículos con la llamada **"(GET)/pedidos"**, de los precios con la llamada **"(GET)/precios"** y de los pedidos con la llamada **"(GET)/pedidos"** al servidor en el puerto 3001. Al principio mostrará un botón para imprimir la carta, mientras cargan las peticiones al servidor. Cuando el usuario pulsa el botón se mostrarán en la parte izquierda todas las categorías que contiene la aplicación, cuando se pulse, se mostrará en el lado derecho los productos disponibles en el sistema. La información se muestra de dos maneras acorde a la situación:
    - **Usuario anónimo:** Se mostrarán los productos junto con sus precios.
    - **Usuario registrado:** Se mostrará si hay stock o no del producto (para mostrar la funcionalidad, se ha tomado con que si hay un pedido de ese artículo se muestre que no hay stock)

Estructura de Carpetas para Aplicación Cliente



Toda la parte importante de la aplicación se encuentra dentro de la carpeta ”src”, en esta se encuentra la carpeta **”componentes”** la cual contiene los componentes de **”Header”** y **”Footer”** junto con sus respectivos documentos, HTML y CSS para estilo y estructura, TypeScript para lógica de programación y spec.ts que controla el renderizado del componente. En la carpeta **”paginas”** se encuentran los siguientes componentes, **”Home”**, **”Login”** y **”Register”**. Todos ellos con sus respectivos documentos como en el **”Header“** y el ”Footer”. Dentro de la carpeta ”modelos” se encuentran los 3 modelos usados en la aplicación, necesarios para manejo de peticiones al servidor y poder así tener un mayor control de las respuestas.





- **Modelo Usuario:** útil para el control de las llamadas al servidor de Login.
- **Modelo Producto:** útil para el control de las llamadas al servidor de Stock

- **Modelo Respuesta:** útil para los dos servidores, para manejar las respuestas de status.

En la carpeta **"servicios"** se encuentran dos documentos importantes:

- **Login.service.ts:** manda peticiones al servidor de login, utiliza para ello la dependencia HttpClient de Angular y también utiliza el CookieService de Angular para establecer cookies en el buscador. Todos los métodos de petición al servidor devuelven un observable.
- **Stock.service.ts:** manda peticiones al servidor de stock, utiliza para ello, como en el anterior servicio, HttpClient de Angular. Todos los métodos devuelven un observable.

Los siguientes documentos que se encuentran (todos los que empiezan por **"app..."**) son configuraciones de la aplicación general, tanto las rutas de los componentes, marcando según que URL se introduzca, que componente se pintará en la página. También se encuentra un documento llamado **"app.module.ts"** en el que se incluyen todos los componentes y que providers se incluyen en la página. En este caso el único provider usado se llama **"CookieService"** el cual permite guardar en el buscador cookies, como el token usado.

Dentro de la carpeta **"assets"** se encuentran las imágenes o elementos estáticos que se pueden usar en toda la aplicación. El documento **"index.html"** es el punto de entrada a la aplicación, teniendo ahí el header, para poder incluir dependencias como Bootstrap, usada en esta aplicación.

Los siguientes documentos no se han mencionado ya que incluye sobre todo configuraciones de dependencias y del propio Angular.

Dependencias

Angular usa módulos para este apartado, así que se hablará de los módulos usados en la aplicación:

- **BrowserModule:** Exporta toda la estructura para una aplicación Angular
- **AppRoutingModule:** Sirve para configurar el routing de la aplicación
- **FormsModule:** Permite crear formularios y poder acceder a sus valores.
- **HttpClientModule:** Módulo para hacer peticiones al servidor.
- **ReactiveFormsModule:** Necesario para crear formularios reactivos y poder validar los campos introducidos.

5. Materiales y métodos:

Tecnologías Utilizadas

NODE.js

Node.js es un entorno de tiempo de ejecución del lado del servidor que se utiliza para desarrollar aplicaciones web y móviles en lenguaje JavaScript. Node.js permite a los desarrolladores crear aplicaciones escalables y de alta velocidad, ya que utiliza un modelo de entrada/salida sin bloqueo y orientado a eventos para manejar las solicitudes. Además, Node.js cuenta con una amplia variedad de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones web y móviles, incluyendo Express.js (Utilizado en los dos servidores). Node.js es una herramienta esencial en el desarrollo de aplicaciones modernas y escalables, y es una habilidad necesaria para cualquier desarrollador web o móvil.

Angular

Angular es la tecnología utilizada para este apartado, es un framework de JavaScript de código abierto escrito en TypeScript para el desarrollo web. Proporciona procesos de desarrollo web más rápido, las aplicaciones web son más ligeras, su objetivo es crear aplicaciones web de una sola página.

Bootstrap

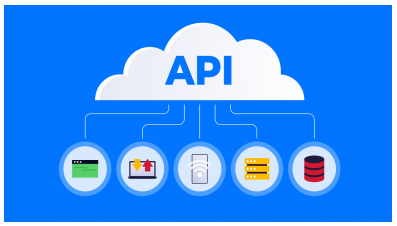
Bootstrap es un framework de desarrollo web gratuito y de código abierto que se utiliza para simplificar y acelerar el proceso de diseño y desarrollo de sitios web y aplicaciones móviles. Bootstrap incluye una serie de herramientas, como componentes HTML, CSS y JavaScript predefinidos, que permiten a los desarrolladores crear interfaces de usuario atractivas y responsivas con una gran consistencia y fácil mantenimiento. Además, Bootstrap se enfoca en la creación de sitios web "mobile-first", lo que significa que se prioriza el diseño y la creación de sitios web para dispositivos móviles antes que para los de escritorio.

CryptoJS

Es una biblioteca de cifrado y descifrado basada en estándares criptográficos reconocidos para JavaScript. Se utiliza para proporcionar una capa adicional de seguridad en aplicaciones web y móviles mediante la encriptación y descifrado de datos sensibles. CryptoJS ofrece una amplia variedad de algoritmos de cifrado como AES, DES, Triple DES, RC4, SHA1 y SHA256. Además, puede trabajar tanto con claves simétricas como asimétricas. Esta biblioteca es muy popular en el desarrollo de aplicaciones de seguridad, ya que permite la encriptación y descifrado de datos de manera segura y eficiente en ambientes JavaScript. Esta dependencia es utilizada en la parte de servidor, expresamente en el servidor login para encriptar la contraseña recibida por parte del

cliente y guardar en la base de datos la resultante de esta encriptación. Esto asegura que, si tiene acceso a la Base de Datos alguien no deseado, no pueda tener la contraseña y con ello no acceder a esta información delicada.

API



Interfaz de programación de aplicaciones, permite la interacción con los servicios web de RESTful. Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones, es un intermediario que permite la comunicación entre diferentes componentes de software, como programas, aplicaciones y sistemas. Las API permiten a los desarrolladores utilizar funciones y datos de una aplicación en otra, lo que reduce el tiempo y los recursos necesarios para desarrollar aplicaciones nuevas. Las API son fundamentales en el desarrollo de aplicaciones web y móviles, y existen diferentes tipos de API, como las API RESTful, que se utilizan para diseñar sistemas de software escalables y flexibles

JWT

Json Web Token, es una tecnología usada para la validación y autenticación de usuarios. Permite validar el usuario por un tiempo determinado, para crearlo desde el servidor de LogIn sería de esta manera:

```
const token = jwt.sign(
    { user_id: req.body.email },
    process.env.TOKEN_KEY,
    {
        expiresIn: "2h",
    }
);
```

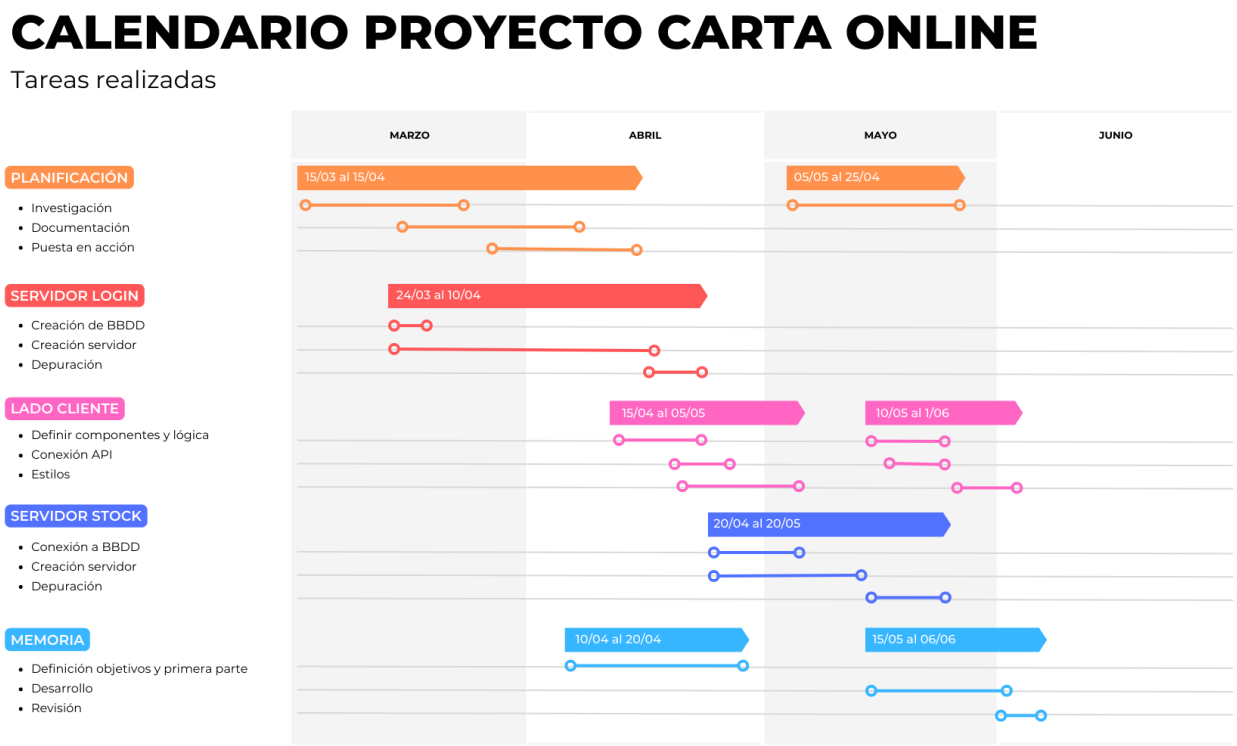
Por una parte, cogería como user_id el correo que se ha logueado correctamente, y por otro, cogería el "TOKEN_KEY" el cual se configura en el archivo ".env", junto con el tiempo en el que expira el token (en este caso 2 horas).

Esta tecnología se usa dentro del servidor dedicado al Login y se envía al cliente, el cual lo almacena como una cookie en el browser que utilicé. El cliente también manda peticiones para validar el token y poder mostrar la información correspondiente con el usuario.

Planificación del proyecto

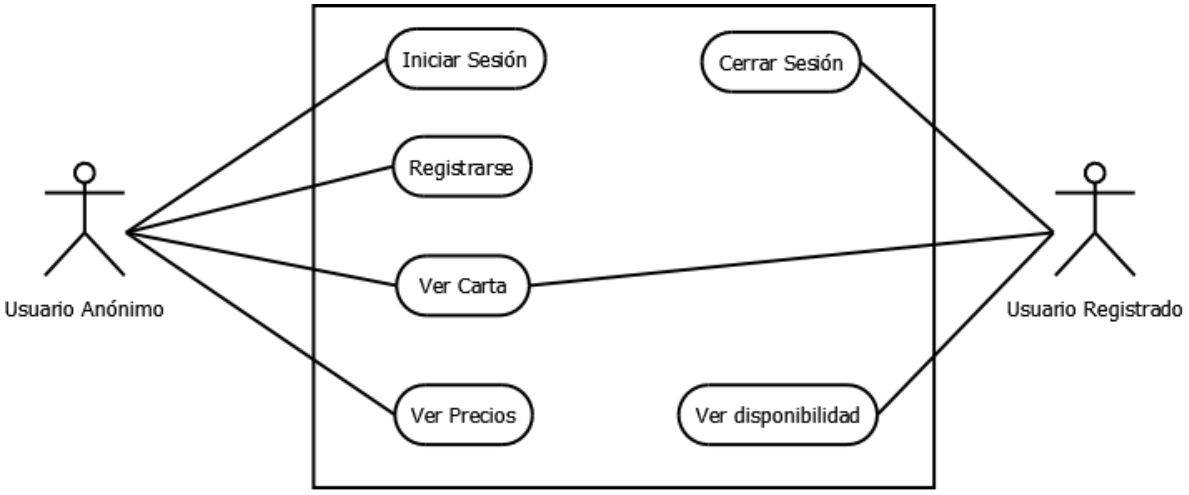
Se ha utilizado una planificación en cascada, este es un procedimiento lineal que se caracteriza por dividir los procesos de desarrollo en sucesivas fases de proyecto. En este caso se ha dividido por las distintas aplicaciones que conforman todo el proyecto. Incluido la investigación y la memoria.

Diagrama de Gantt



6. Resultados y análisis

Casos de Uso General



Caso de Uso para Inicio de sesión

Usuario anónimo	Programa
1. Entra en la página principal	Carga el componente Home
2. Click en botón “Inicio de Sesión”	Carga el componente LogIn
3. Introduce email y contraseña y click en botón “Inicio Sesión”	Comprueba la validez de los datos introducidos: **Opción 1:** Datos Correctos, manda petición a servidor para comprobar autenticidad

	Opción 2: Datos Incorrectos: Mensaje de alerta con el mensaje “No se han introducido correctamente los datos”
4. Espera la respuesta del Servidor	Acorde a la respuesta del servidor: **Opción 1:** LogIn correcto: Guarda token en cookies, redirige al componente Home y Header cambiar botones ya que detecta el token como cookie, imprime bienvenida de usuario y botón de cierre de sesión. **Opción 2:** LogIn incorrecto: Se muestra un mensaje de que el usuario no existe o que la contraseña o email son incorrectos.

Caso de uso para Registro de Usuario

Usuario Anónimo	Programa
1. Entra en la página principal	Carga el componente Home
2. Click en “Registrarse”	Carga el componente Register
3. Introduce campos y click en botón “registrar”	Comprueba la validez de los datos: **Opción 1:** Algún dato no es correcto, no cumple con los requisitos del campo que hay que introducir, input se vuelve rojo y no se envía la petición. **Opción 2:** Los dos emails introducidos no coinciden, sale un mensaje de alerta “Los emails no coinciden”

	<u>Opción 3:</u> Las dos contraseñas introducidas no coinciden, sale un mensaje de alerta “Las contraseñas no coinciden”. **<u>Opción 4:</u>** Todos los campos válidos. Se envía petición al servidor.
4. Espera respuesta del servidor	Recibe respuesta del Servidor **<u>Opción 1:</u>** Usuario ya registrado. Sale un mensaje de alerta “Usuario ya registrado” **<u>Opción 2:</u>** Registrado correctamente. Redirige a la página de Home.

Caso de uso para visualización de carta de un usuario Anónimo

Usuario Anónimo	Programa
1. Accede a la página	Carga el componente Home y hace peticiones al servidor de stock
2. Click en botón “Mostrar carta”	Imprime todas las categorías que existen en el programa de hostelería
3. Click en categoría	Imprime en el lado derecho de la página los productos correspondientes a la categoría seleccionada, tiene en cuenta el precio y si están o no disponibles.

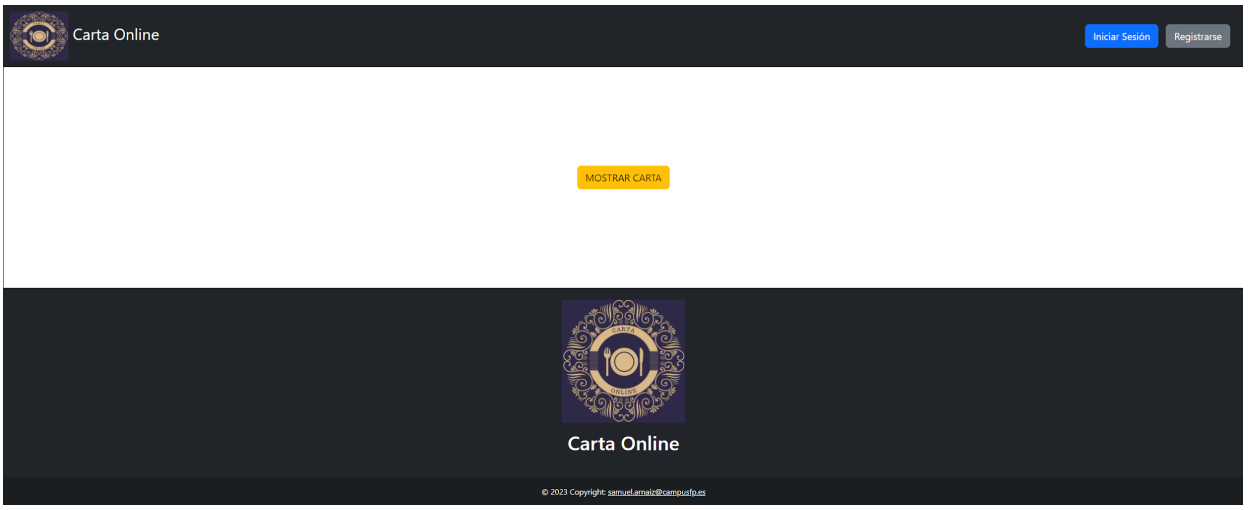
Caso de uso para visualización de carta de un Usuario Autenticado

Usuario Autenticado	Programa

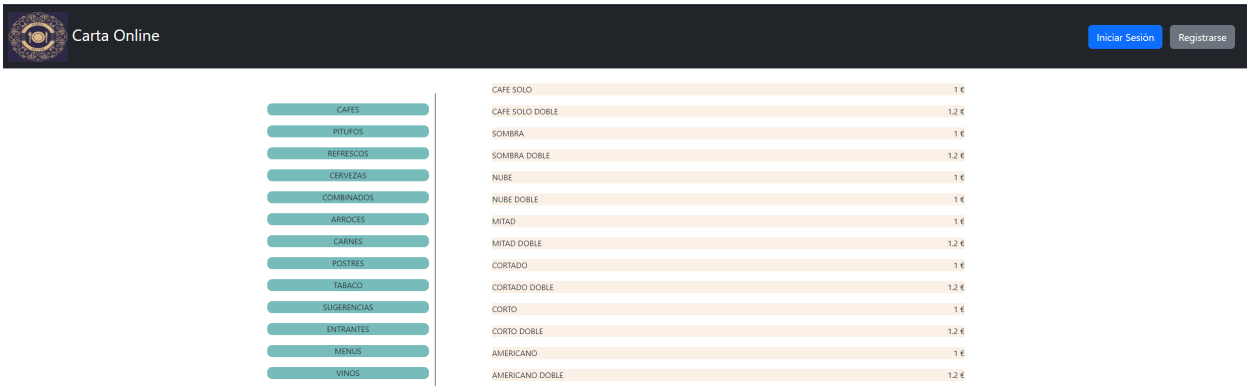
1. Accede a la página	Carga el componente Home y hace peticiones al servidor de stock
2. Click en botón “Mostrar carta”	Imprime todas las categorías que existen en el programa de hostelería
3. Click en categoría	Imprime en el lado derecho de la página los productos correspondientes a la categoría seleccionada, solo imprime el producto y si están o no disponibles con unos iconos de check o cruz.

Aplicación Cliente Apariencia

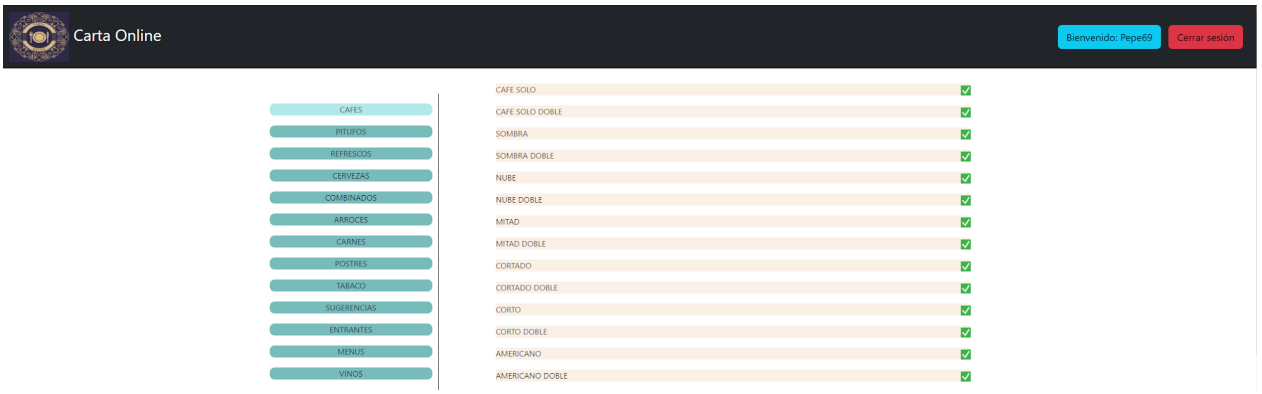
En cuanto el usuario inicia se encuentra con el Home como se ha comentado anteriormente, esta sería la apariencia:



Cuando se pulsa el botón y alguna categoría se mostraría el precio en el caso de usuario anónimo:



En el caso de estar logueado se vería si están o no disponibles (esta implementación es para mostrar la futura funcionalidad al conectarse al gestor de stock del programa):



7. CONCLUSIONES

El proyecto no ha cumplido con todos los objetivos a causa de las limitaciones del programa de hostelería prestado por parte de DSTnet, este era una demo y no incluía gestión de stock, implementación que el programa original incluye y todos los programas de hostelería también incorporan.

Esto se ha subsanado incluyendo como proyecto una demo de la futura funcionalidad en la parte de Usuario Registrado, teniendo en cuenta solo los pedidos que se hacen en la aplicación y no el stock en sí, figurando sin stock solo por haberse pedido y pagado una vez en la aplicación de hostelería.

El código de la parte cliente está configurado para solo tener que incluir la llamada al servidor de stock y poder imprimir o no, el producto. El servidor de stock solo tiene que hacer una llamada a la base de datos del programa, para poder luego enviar si existe o no productos en el comercio.

La parte de Registro se ha dejado accesible para el usuario anónimo, esto permite que cualquiera que pruebe este proyecto pueda ver también la funcionalidad demo del stock.

8. LÍNEAS DE INVESTIGACIÓN FUTURAS

- Integración con stock de programa para que funcione como
- Proteger componentes para que solo puedan acceder los usuarios que se deseen.
- Implementar criptografía en el lado cliente y enviar al servidor la contraseña encriptada directamente.
- Implementar los servidores en un servidor externo o en el propio servidor del programa.

9. BIBLIOGRAFÍA

Crear Servidor Nodejs para Autenticación:

https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/

UTILIDAD DE LOS MIDDLEWARE:
https://expressjs.com/en/guide/using-middleware.html

CREAR UN LOGIN EN ANGULAR:
https://codingpotions.com/angular-login-sesion

MODULO PARA CONFIGURAR COOKIES EN ANGULAR:
https://www.npmjs.com/package/ngx-cookie-service

CONTROL DE ERRORES:
https://michael-karen.medium.com/esperando-lo-inesperado-buenas-pr%C3%A1cticas-para-el-manejo-de-errores-en-angular-dc578da68ef9

LIBRERÍA CRYPTOJS:
https://cryptojs.gitbook.io/docs/

Dependencia Express:
https://kinsta.com/es/base-de-conocimiento/que-es-express/

Dependencia CORS:
https://www.npmjs.com/package/cors

Dependencia Nodemon:
https://www.npmjs.com/package/nodemon

Dependencia Dotenv:
https://www.npmjs.com/package/dotenv?activeTab=readme

Dependencia JWT:
https://www.npmjs.com/package/jsonwebtoken

FORMULARIOS REACTIVOS ANGULAR

https://www.youtube.com/watch?v=nVdCrD5Ehjo&list=PL_9MDdjVuFjEOZwlkucIUKeBILv-gLeN_&index=2

https://mailtrap.io/blog/angular-email-validation/#1-Apply-validation-rules

https://www.bezkoder.com/angular-15-form-validation/

CONECTAR CON SQL SERVER

https://www.tabnine.com/code/javascript/functions/mssql/Request/input

VER EL PUERTO EN SQL SERVER

https://www.sysadmit.com/2016/03/mssql-ver-puerto-de-una-instancia.html

API

https://www.redhat.com/es/topics/api/what-is-a-rest-api

SETEAR EL PUERTO SQL SERVE

https://github.com/tediousjs/tedious/issues/1120

https://stackoverflow.com/questions/35026/sql-server-convert-a-named-instance-to-default-instance

Bootstrap

https://www.hostinger.es/tutoriales/que-es-bootstrap

10. OTROS PUNTOS

Agradecimientos

A la Empresa DSTnet por prestar la demo y el acceso a la Base de Datos

A Rúben Álvarez Senovilla, testing del proyecto.