BEE BEER

ÍNDICE GENERAL

INTRODUCCIÓN

Para la realización del TFG hemos tenido en cuenta tanto el contenido del proyecto en sí como las diferentes tecnologías conocidas. Éste es un proyecto que busca potenciar las funcionalidades del sistema de tres capas, Frontend, Backend y Base de Datos. Dado que se trata de un proyecto de gestión de productos, en concreto de gestión de productos de cervezas. Al realizar el proyecto, aunque teníamos en mente como poder realizar un proyecto de estas características nos planteaba la necesidad de dejar entreabierta distintas opciones, ya que la definición propia del mismo podía cambiar por las propiedades del mismo.

JUSTIFICACIÓN

En el caso de tener una empresa intermediaria de cervezas, tendríamos que hacer frente a cambios en los atributos de los mismos, posibles actividades relacionadas con la cerveza. Además, Proyect Beer se crea para satisfacer la necesidad de gestión de no solo una empresa de cerveza, sino que busca satisfacer las necesidades de un elenco de empresas, algo que se realiza a través de un sistema API REST. Los métodos del Backend buscan satisfacer cada una de las necesidades específicas de las diferentes empresas. Por lo que vamos a trabajar a través de micro servicios, que proporcionan un soporte a cada una de las peticiones que realizan. A continuación, hemos creado un Frontend que proporcione una visión general del proyecto, que muestre de forma ágil las potencialidades del proyecto.

Debido a las necesidades de elasticidad y maleabilidad del proyecto hemos decidido realizar dicho proyecto con la tecnología MEAN Stack. Con una estructura de Base de Datos realizada en Mongo DB, debido a que los atributos deben ser maleables y su modelo pueda cambiar específicamente para alguna de las empresas, las posibles relaciones actuales y futuras de la aplicación. Además, deberemos tener en cuenta que todos los atributos de las distintas colecciones, así como las propias colecciones en sí podrían ser modificadas.

Utilizaremos una estructura integrada de Node, con Express y distintas librerías adscritas a las mismas, para satisfacer el servicio centrado en el Backend. Lo realizamos de esta manera debido a que éste deberá soportar distintas llamadas externas a la misma, así como desde la propia aplicación, llamadas dentro del propio Backend.

Finalmente, para crear una estructura visual del servicio web hemos utilizado Angular, principalmente con la librería Angular Module, para satisfacer distintas necesidades del proyecto. Utilizamos angular ya que tendremos en cuenta que, muchas empresas pequeñas y medianas suelen utilizar este framework.

Para ayudarnos en la creación del código como IDE hemos utilizado Visual Studio Code, debido a los conocimientos adquiridos durante el curso, nos ha facilitado mucho el trabajo.

MEAN STACK

OBJETIVOS

Nos hemos planteado lograr un proyecto que satisfaga todas las necesidades posibles. Haciendo hincapié en la integración del proyecto, pudiendo dar un servicio general a las distintas empresas. Algo que se puede comprobar fácilmente no solo generando un Frontend para mostrar los resultados. Sino que, además pueda hacer frente a distintas necesidades de las distintas empresas.

Éste proyecto tiene como principales finalidades, un servicio de gestión de clientes, un sistema para mostrar los productos disponibles, así como la compra de los mismos. Para la gestión de la compra se establece una estructura de carrito complejo. Así como un sistema que gestione el stock propio del mismo. Un servicio de cata online que proporcione una experiencia a los clientes de los distintos productos y los haga partícipes dando lugar a una mejora en la experiencia. Un juego para afianzar los conocimientos sobre cervezas, algo que repercute directamente en la satisfacción del cliente. Y finalmente un sistema de documentación para que los catadores o gestores de tienda puedan poner a disposición de sus clientes la documentación que requieran.

AGRADECIMIENTOS

Me gustaría agradecer la ayuda puntual que he conseguido con la experiencia en Kenjo. Ya que sin él y debido al reducido tiempo para realizar las prácticas he podido comprender en gran medida la funcionalidad de un sistema MEAN Stack.

COMPONENTES DEL EQUIPO Y APORTACIÓN REALIZADA POR CADA ESTUDIANTE

Pedro Murillo Arnau: he realizado el proyecto en MEAN Stack, que consta de una base de datos en Mongo DB, en principio de forma local, Node con Express para la creación de un Backend, que busca alocar cada una de las llamadas que se producen desde el Frontend. Respecto al Frontend he utilizado distintas herramientas de Angular así como librerías para implementar funciones determinadas, además he buscado como implementar otras herramientas que finalmente no he llegado a implementar por la falta de tiempo o por falta de conocimiento suficiente para poder implementarlo. A la hora de realizar el proyecto he utilizado las herramientas tanto de Github, como de Sourcetree para poder gestionar el proyecto y los distintos avances del mismo.

Además me he servido de conocimiento adquirido para poder implementar las diferentes partes del proyecto, tanto para el diseño y estructura del Frontend, como para el desarrollo y la correcta estructura del Backend.

Nagib Delgado Morales: se ha encargado de la parte del back-end, desarrollar el login de usuarios, dándole la seguridad necesaria con la ayuda de tokens. Un crud tanto para usuarios como para productos.

Para poder ejecutar el proyecto en un entorno local es necesario disponer de los scripts de la bbdd, los cuales descargamos desde el mismo repositorio del proyecto, abriendo la carpeta script y lanzándolos desde la interfaz gráfica de mongo compas.

Tanto el Frontend como el Backend se pueden descargar desde el repositorio: "https://github.com/PedroMurilloArnau/proyecto/tree/Pedro"

El proceso de ejecución sería la carga de los script de BBDD desde la interfaz gráfica de mongo, abrir tanto el Frontend y el Backend desde Visual Studio Code, y lanzaremos el comando ng serve -o y npm start para lanzar los respectivos proyectos

FASES DEL PROYECTO MEAN STACK

MODELO DE DATOS UTILIZADO

Para el modelo de datos hemos utilizado un modelo de datos no relacional y para ello hemos utilizado Mongo DB. A la hora de gestionar los datos hemos utilizado la consola de Mongo DB, Mongo Compass, que es la GUI para la gestión de MongoDB.

Para estructurar los datos, ya que el proyecto ha sido creado sobre la marcha, hemos realizado estructuras de colecciones abiertas, que permiten una rápida y flexible gestión de datos frente a otras tecnologías. Para la creación de dichas colecciones hemos utilizado una estructura de datos con el tipo de cada dato, generando un modelo específico y único para cada colección. Posteriormente hemos realizado una llamada a cada colección para generar por primera vez cada colección, lo que ha permitido controlar los posibles fallos generados en la etapa de desarrollo. Es destacable la importancia de generar Arrays de Objetos en las colecciones para mejorar el control y el alcance de determinados datos.

En total el proyecto consta de siete colecciones donde quedarán alojados todos los datos relativos a los apartados fundamentales, Usuarios, Cervezas, Pedidos, Catas y Juegos.

En Usuarios hemos establecido 4 tipos distintos de usuarios, “Cliente”, “Gestores de tienda”, “Expertos catadores” y “Programadores”. En un principio se quería desarrollar una estructura más abierta y establecer una nueva colección con el tipo de usuarios que pudieran acceder a la tienda. Pero al establecer que finalmente queríamos separar lo máximo posible el Backend y el Frontend, se buscó cerrar la posibilidad de abrir el tipo de usuarios para prevenir posibles problemas de incompatibilidad a la hora de generar nuevas páginas que utilizan este modelo de datos.

Respecto a la gestión de la cerveza, tiene asociada 3 colecciones, “Cervezas con su stock” para la gestión general de la tienda, “Tipos de cervezas” para organizar cualquier llamada al registro de la tienda que quiera conocer los tipos de cerveza de la misma y “Notas de cata”, que nos proporcionarán las impresiones de las características de las distintas cervezas generadas por los usuarios.

Respecto a pedidos, para recoger el final de la solicitud del carrito de compra generado por los usuarios de tipo “Cliente”, hemos creado la colección de “Compra”, que alberga todas las cervezas compradas. Modificando el stock definitivamente de la colección de

“Cervezas” y alojando información fundamental de la compra como precio, id y número de factura y fecha.

En lo que concierne a la Cata, contamos con dos colecciones, “Catas” donde se alojarán las distintas catas, así como documentación que ayudará a la creación de la misma y dará información a los “Clientes” sobre información necesaria para las catas. Cabe destacar que para el uso de la disponibilidad de los “Clientes” al acceso a sus catas, así como el inicio de la misma la colección cuenta con una atributo de tipo boolean que abre las puertas a su uso, o deniega el acceso al mismo.

Finalmente, para gestionar los datos de juego hemos creado dos colecciones, “Juego” donde se irán guardando las diferentes partidas que realicen los “Clientes” así como “Preguntas” una colección donde se almacenarán las distintas preguntas, en ellas tendremos la respuesta correcta dentro del Array de Objetos de “preguntas” asignado con el atributo de “valor” correspondiente a true. En nuestro caso hemos utilizado un modelo de juego de preguntas desde el Frontend de entre 2 y 4 respuestas, pero podríamos tener casos de tener más respuestas por pregunta.

El acceso a los datos se realizará siempre a partir del Backend. Mediante llamadas a los distintos micro servicios. Para la conexión con el Backend hemos utilizado la librería de “Mongoose” versión 6.2.10.

ESTRUCTURA DEL BACKEND

El Backend está realizado en Node con el uso de la librería de Express. Realizamos uso de gestión del rooting mediante llamadas y redirección a distintas direcciones. Además hemos establecido que las llamadas se realizarán en una estructura de micro servicios, por lo tanto dentro de cada uno de los casos de uso se alojan tanto las direcciones en archivos de tipo ".routing.js" así como los métodos que se van a implementar en las diferentes llamadas de tipo ".microservice.js".

Para la correcta funcionalidad de nuestro Backend hemos utilizado las siguientes librerías. "mongoose" para la gestión de la base de datos y las llamadas de la misma desde nuestro Backend; "jsonwebtoken" para poder generar un token que utilizaremos en nuestra sesión; "cors" para la gestión de las páginas seguras y poder realizar llamadas de tipo Api/Res; "express" para realizar la conexión desde el Backend, una librería que nos facilita la gestión de rutas; "express-session" es la librería de express que utilizamos con el fin de crear sesiones; "express-validation" una librería de express que utilizamos para realizar diversas validaciones en nuestras rutas, a modo de middleware; "bycryptjs" utiliamos para realizar la encriptación de las contraseñas. Además, hemos trabajado, aunque no se ha incluido en el proyecto "http-status-code", "superagent" entre otros para poder solventar distintos tipos de llamadas que finalmente no se han llegado a implementar.

A la hora de arrancar el Backend, previamente tenemos que tener una configuración de base de datos, tener una dirección y tener claro qué tipo de estructura queremos realizar. La conexión se realiza desde nuestro documento "app.js" un documento que alberga los principales datos de la conexión. desde el tipo de conexión que deseamos realizar en lo referente al control de acceso HTTP CORS. En nuestro caso dejamos abierta esta conexión, aunque de forma sencilla se podría establecer una white list a través de la cual controlar la conexión que se realiza. Además de controlar de forma sencilla los diferentes ataques a nuestra estructura del Backend, IPs y rutas de acceso. Posteriormente declaramos las diferentes rutas de acceso y donde se encuentran las carpetas alojadas. Finalmente se establece el punto de acceso propiamente dicho.

Entrando en el archivo "app.js" de "malta-cloud" podemos ver que tenemos asociados 5 rutas principales, "beer/users" donde se alojan todas las llamadas al Backend correspondientes a la gestión de los usuarios; "beer/cataloge" que corresponde con las

rutas de la gestión de la cerveza; “beer/purchase” correspondiente a la gestión de los pedidos de cerveza; “beer/tasting” donde se alojan todas las peticiones de cata y todo lo referido a la cata; y “beer/game” que acoge todas direcciones de peticiones al backend correspondientes con el juego.

Otra de las estructuras fundamentales de este proyecto hace referencia a la seguridad. En nuestro caso esta seguridad la utilizamos creando una estructura de llamadas a través de un token de validación. Este token de validación lo generamos al inicio de la sesión mediante el método de “loginUsuario” que se encuentra en “auth.microservice.js”. En esta llamada recogemos los atributos de “email” y “password” desde el Frontend y esta llamada, una vez comprobado la existencia de un email que corresponde con un usuario alojado en la base de datos y comprobado la validez de la contraseña a través de la des encriptación de la misma utilizando el mismo modelo que utilizamos a la hora de realizar la encriptación. Generamos un token, a través de los atributos de nombre del usuario y su email y alojamos este token en una sesión. Asignándole a la misma un tiempo de latencia de un día. Esta duración la utilizaremos en el caso que dejemos la sesión abierta esta dejará de funcionar en un periodo de 24h.

Además, el Backend utiliza una estructura de middlewares que verifican características necesarias en diversas peticiones. Ya que, aunque se realizan comprobaciones en el Frontend, queremos sustentar nuestro Backend con un sistema de medidas de protección a la hora de la llegada de los datos. Por ejemplo, en la longitud del atributo “password” a la hora de dar de alta un nuevo usuario.

Las distintas respuestas se darán a través de una respuesta determinada el Frontend.

ESTRUCTURA DEL FRONTEND

La estructura del Frontend está realizada en Angular. Y realiza llamadas al Backend a través de rutas de acceso controladas por Guards.

Primeramente tenemos que destacar las librerías utilizadas para realizar el proyecto, estas son, “angular-material” para la gestión principal de los diferentes atributos de las páginas, así como otras librerías asociadas a angular tales como, “angular-CDK”, “angular-forms”,”angular-animations”,”angular-flex-layout” entre otras. Así como librerías para la mejora visual tales como “bootstrap”; “sweetalert2” para la generación de pop-ups.

La estructura del Frontend está construida utilizando principalmente estructuras del angular module para que la realización de la misma, tanto visualmente como funcionalmente tenga cierta consistencia. Por ello ya que tenemos la estructura de app, hemos creado un archivo “material.module.js” donde iré alojando los distintos módulos de angular module utilizados. Entre ellos los módulos de tablas, diálogos, cards de información… Estos módulos se incluyen en “app.module.js” junto con los distintos componentes y otros módulos utilizados.

Para el enrutamiento se crean una serie de rutas sencillas, asociadas a componentes determinados. Para acceder a alguna de las rutas se requiere seguridad para su acceso entre ellas “Catalogue” donde se aloja todo lo requerido para el catálogo de la cerveza y notas de cata; “Tasting” donde crearemos todo lo requerido para la cata, tanto documentación como la creación de catas así como la activación de las propias catas; “Game” donde crearemos nuevas preguntas y respuestas para creación de los juegos así como la creación como un juego que dé la oportunidad al usuario a poner a prueba sus conocimientos sobre la cerveza; ”Purchase” donde podremos gestionar nuestro carrito; y ”Calendar”, en otras no lo requiere como es el caso de “Signup” para darse de alta los usuarios; “login” así como la pantalla principal, que nos mostrará información de la propia página.

La estructura principal del enrutamiento pasa por “app.component.js”. Donde se irán alojando las estructuras de navegación tanto del sidenav como de la barra de menú. Y dentro de él tenemos las diferentes rutas.

Pare el uso y despliegue del sidenav utilizamos un tipo evento de tipo “EventEmiter”, que desde uno de los componentes podremos llamar a otro, a través de un decorador de tipo “Output”. Este será llamado por un método dentro de nuestro menú para poder desplegarse y dentro de nuestro propio sidenav para poder cerrarse. Facilitando el uso de estos dos componentes de forma síncrona. Además, tenemos estructura responsive en el que damos preferencia del uso del sidenav en el caso de que la pantalla sea de un tamaño menor de 600px, mediante el uso de “layout” lo definimos como “fxHide.gt-xs” y el despliegue de los distintos botones del menú con un tamaño mayor a 599px mediante la definición de “fxHide.xs”. Además de utilizar estructuras propias de alineamiento de angular material como “fxFlex fxLayout…” para definir la alineación de los elementos en el contenedor mediante una estructura flex.

Las rutas más complejas son las concernientes al “juego” y los referidos a la “cata”. Para ello en ambos casos se pueden crear tanto una cata “cata” como un “juego” utilizando para el primer caso un usuario de tipo “Expertos catadores” y para el segundo de tipo “Gestores de tienda ". En ambos casos consiste en un formulario en el que puede crearse los elementos de cada tipo y mediante una llamada al servicio correspondiente se crea una llamada de tipo Post a través de la cual se creará un nuevo objeto de tipo “Cata” o de tipo “Pregunta”. Cabe destacar que en el tipo cata los criterios son estrictos en las características referidas al tipo de cata. Una vez seleccionado el tipo de cata, se nos facilita una lista de cervezas disponibles dentro de los diferentes select de cervezas, en los cuales, a la hora de seleccionar un tipo diferente de cerveza, eliminamos ese elemento de la lista para que en los siguientes select no aparezca ese tipo de cerveza. En el caso que seleccionemos un tipo diferente diferente del seleccionado cambia la selección del mismo en las listas de los siguientes selects.

En concreto una cosa de la cual destaca la página es el uso de diálogos y feedback para buena parte del código. Destacando la importancia del mismo en la cata dentro del componente “current-tasting”, en él damos importancia a la relación cliente-tienda que se produce al facilitar que los clientes puedan crear comentarios concretos dentro de su actividad a través de la creación de notas de cata. Éste diálogo consta de 6 contenedores, el primero referido al arranque de la “cata” donde se proporciona información “Expertos catadores” así como un círculo de progreso, proporcionado por angular-material, así como el progreso de la misma. Los siguientes contenedores irán creándose dependiendo del tanto por ciento de la evolución del progreso de la cata”25%, 50% y 75%”,

mostrando en cada caso un formulario para poder rellenar la nota de cata. Una vez completada cada una de las notas de cata podemos enviar la nota de cata o no. Finalmente podemos finalizar la "cata" que estemos realizando, produciendo una modificación en la base de datos de la cata pasando el atributo de "estado" del alumno en cuestión inhabilitándolo para acceder a esta misma cata.

También se destaca tanto la "documentation", relativo a las catas en las que hemos implementado un formulario de tipo desplegable por fases. En él se puede desplegar cada una de las fases de la documentación una a una y una vez finalizada se mostrará en el componente de "all-documentation".

Otra parte destacable es la relativa al componente de "new-question". Éste componente se accede a él, a través de un usuario de tipo "Gestores de tienda". El cual puede dar de alta una nueva pregunta o ver todas las preguntas que hemos creado como usuario, a través de un menú de dos botones que llaman a dos métodos diferentes. En el primero tenemos un formulario con la primera respuesta como respuesta correcta, y el resto como respuestas erróneas, siendo obligatoria al menos una incorrecta. En el segundo caso tendremos una tabla con todas las preguntas y el acceso a las distintas respuestas con una llamada a un diálogo, en el que se transfiere la información de la pregunta en cuestión. En este segundo apartado el gestor tiene a su disposición un "input" de tipo autocompletado, por el cual podremos ver si tenemos preguntas iguales o de contenido similar.

DIAGRAMA DE CASOS DE USO

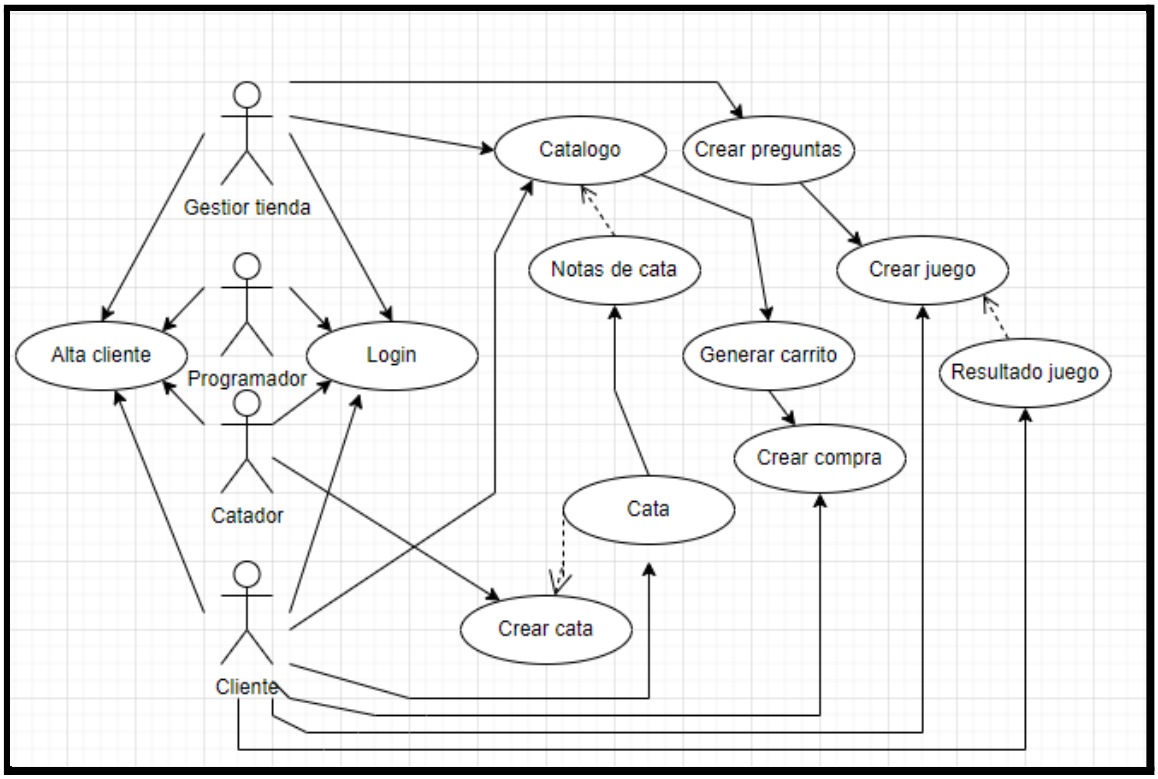
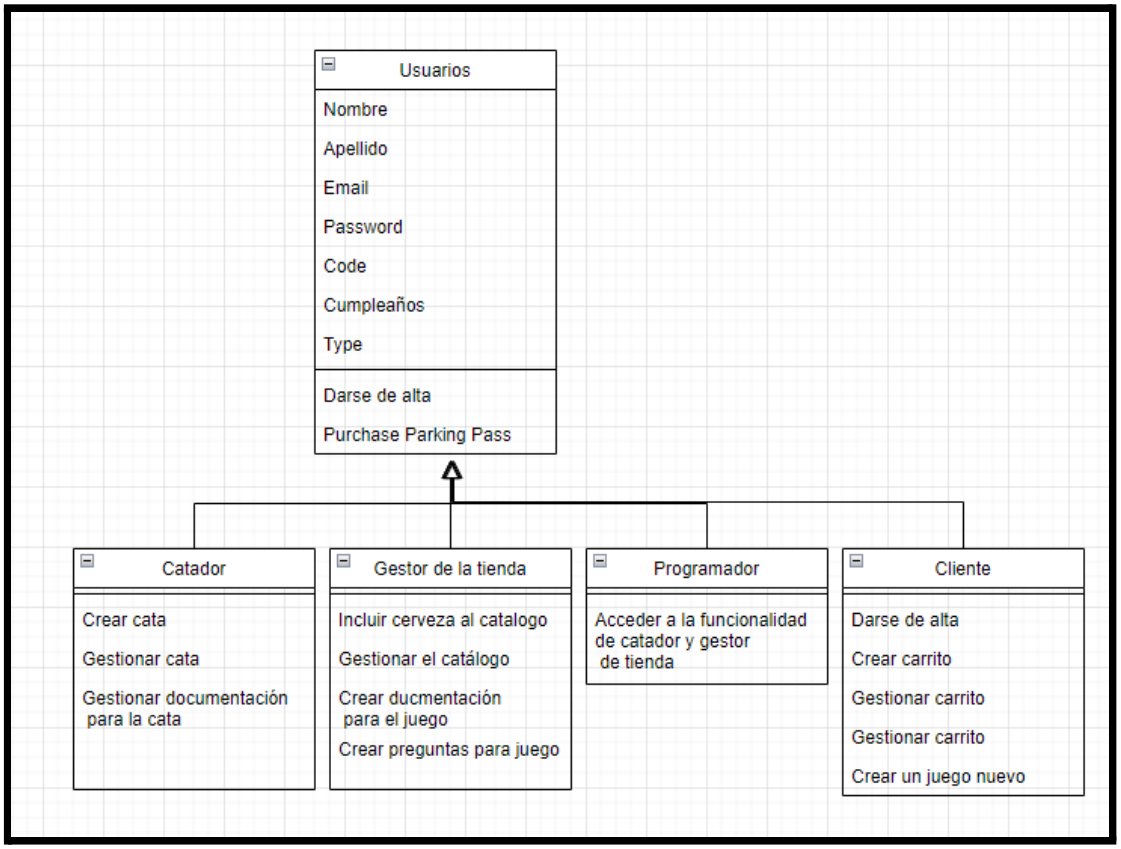


DIAGRAMA DE CLASES



CONCLUSIONES Y MEJORAS DEL PROYECTO

Respecto a los casos de uso de las catas nos habría gustado dedicar más tiempo al desarrollo de la misma ya que las posibilidades de crear catas de tipo diferente con diferentes tipos de cerveza, daría más posibilidades de crear catas más complejas además de poder dar mayor funcionalidad al diario para la creación de catas y una organización más clara de catas por fecha. Ya que en un primer momento se planteó tener las catas organizadas por un calendario y que ese calendario se pudiera ver tanto por "Expertos catadores", "Gestores de tienda" como por "Cliente", controlando su funcionalidad por los "Gestores de tienda". Además de que las catas tendrían que venir acompañadas de un link a una plataforma de streaming que pudiera facilitar la vista de las mismas. Además, que para las pruebas de cata el tiempo de la misma está falseado para realizar una prueba funcional, pudiendo ser este modificado con el tiempo de la cata con el atributo de "duration".

Otro de los aspectos de casos de uso que faltan por implementar es poder modificar las notas de cata, las catas y las preguntas del juego por los respectivos usuarios. Así que, si no estás de acuerdo con tu nota de cata, puedes modificarla, así como eliminarla si no estás de acuerdo con la misma. Lo mismo deberían poderse modificar tanto las preguntas realizadas por el "Gestor de tienda", pudiendo modificar las preguntas, así como cualquier aspecto de la misma. Igual que en la creación de la cata.

Respecto al Frontend son muchas las cosas que nos gustaría haber implementado. En lo relativo a las rutas se debería de haber establecido una estructura de rutas protegidas, que solo pudieran acceder a ellas a través de un guard, algo que facilita la propia estructura de la web, así como una notable mejora de la seguridad además de aumentando la modulación.

En el Frontend también falta por implementar un menú superior a la hora de seleccionar el componente de "all-documentation", dentro de "taste". Donde a través de unas pestañas de tipo "Chips" de información se desplegarán los contenedores correspondientes a los documentos relativos a la información requerida. Estos contenedores se desplegarían de una forma dinámica a través de observadores dentro de dicho menú.

En el apartado del Frontend relativo a la creación de nuevas preguntas el componente "new-question", debería de implementarse una tabla con los criterios de búsqueda del "input" de tipo autocompletado, de tipo dinámico. Y que accediendo a ella nos deje modificar la información de la pregunta en cuestión.

También se implementarían interfaces para el control de los atributos que se utilizan y la definición de los mismos erradicaría los errores que se producen por la incongruencia en los métodos de angular modules, así de métodos de ngFor.

Respecto al Modelo de datos, tendríamos que tener una estructura funcional que nos cree las distintas colecciones, ya que, aunque se puede iniciar de forma autónoma desde la propia página, para ciertas estructuras es importante lanzar los diferentes javaScripts de pruebas de todos ellos, actualmente se encuentran en la siguiente ruta "proyecto\malta-cloud\src\bbdd\test".

En el Backend falta modificar las llamadas de peticiones que redirija las mismas dependiendo de las rutas de direcciones, es decir que dependiendo de la llamada se cree una nueva base de datos generando las nuevas colecciones, que serán únicas dependiendo del tipo de tienda. Es decir que por cada tienda tengamos una única base de datos. Así podremos tener distintas llamadas al Backend completando así la estructura del Rest-Api que ofrezca multiservicios.

Éstas son algunas de las muchas cosas que nos gustaría implementar pero que debido a la falta de conocimiento o de tiempo no se ha llegado a completar.