Aplicación para realizar la compra a personas confinadas

Carlos Doblas Jodar

Resumen—En este proyecto se va a desarrollar una aplicación web para poder hacer la compra a personas que no pueden llevarla a cabo por sí mismas. La web permitirá al usuario realizar una cesta de la compra. Creando y añadiendo artículos para que otro usuario pueda irlos a comprar. Los dos usuarios podrán comunicarse mediante un chat de texto relacionado con la cesta. Los usuarios deberán obtener una cuenta asociada al portal mediante un formulario de registro. Una vez dispongan de cuenta, podrán iniciar sesión en el portal. Como objetivos secundarios se va a implementar: una interfaz totalmente ajustable al tamaño del dispositivo que lo utiliza, un sistema que informa al usuario cuando se disponga de una nueva versión de la web y un entorno de puesta en producción eficiente con contenedores para poder realizar una entrega continua del software.

Palabras clave—Front-end, Back-end, API REST, Object-relational mapping, Json Web Token, ServiceWorker, React.js, Node.js, Redux, Reducer, Action, bcrypt, Docker, framework.

Abstract—In this project, we are going to develop a web application that is able to make purchases to persons who cannot carry it out on their own. The web will allow the user to make a shopping cart. Creating and adding items so that another user can buy them. The two users will be able to communicate through a text chat related to the basket. Users must obtain an account associated with the portal through a registration form. Once they have an account, they will be able to log into the portal. As secondary objectives, we will implement: an interface that is fully adjustable to the size of the device that uses it, a system that informs the user when a new version of the web is available, and an efficient production start-up environment with containers to be able to deliver the software with a continuous method.

Index Terms—Front-end, Back-end, API REST, Object-relational mapping, Json Web Token, service worker, React.js, Node.js, Redux, Reducer, Action, bcrypt, Docker, framework.

1 Introducción

La posibilidad de contagio de coronavirus en los supermercados ha sido objeto de atención desde el inicio de la pandemia. Hacer la compra es indispensable y, la amenaza para la salud que ello conlleva, se convierte en una barrera para las personas de riesgo. La crisis de la COVID-19 ha cambiado completamente nuestra forma de vivir y de comportarnos. Muchos usos y costumbres se han visto alterados y transformados por la situación de confinamiento, entre ellos, el uso del E-commerce. Las compras online han experimentado un gran auge durante la pandemia. Al no poder salir de casa, las personas han realizado gran parte de sus compras por internet. Es una buena alternativa si se conoce el medio, pero supone un problema para gente, normalmente de mayor edad, que desconfía de los métodos de pago o no tiene las herramientas disponibles para realizar sus compras por internet. Irónicamente, durante la pandemia, este grupo de gente era el más vulnerable al contacto con otros individuos.

Entendiendo el problema expuesto, este proyecto intenta tender una mano hacia esas personas que necesitan realizar sus compras semanales y no pueden valerse por si mismas para llevarlas a cabo. Por lo tanto, el objetivo es crear una plataforma web que permitirá a los usuarios realizar una cesta de la compra. Creando y añadiendo artículos para que otros usuarios puedan, mediante un listado de cestas, irlas a comprar. Una vez la cesta se ha seleccionado por el usuario comprador. Los dos usuarios podrán comunicarse mediante un chat de texto relacionado con la cesta. Los usuarios deberán obtener una cuenta asociada al portal mediante un formulario de registro. Deberán introducir los datos de su perfil y, una vez dispongan de cuenta, podrán iniciar sesión en el portal. Como objetivos secundarios se va a implementar: un sistema de notificación de actualizaciones para, cuando se disponga de una nueva versión de la web, informar al usuario, una interfaz totalmente ajustable al tamaño del dispositivo que lo utiliza, un entorno de puesta en producción eficiente mediante contenedores y se seguirán patrones de diseño del software para que el código sea fácilmente ampliable y mantenible.

En los siguientes apartados del documento, podremos ver

- *E-mail de contacto: doblasjodar@gmail.com*
- *Mención realizada: Ingeniería del Software*
- *Trabajo tutorizado por: Cristina Romero Tris (Área de Ciencias de la Computación e Inteligencia Artificial)*
- *Curso 2020/21*

los objetivos que se han definido para el proyecto, el estado del arte de aplicaciones similares, la metodología seguida, análisis, resultados y conclusiones.

2 Objetivos

Este proyecto está formado por los siguientes objetivos:

- La aplicación debe ser muy intuitiva y tener un uso tan sencillo como sea posible.
- Los usuarios deben poder crear un listado de la compra. Deben poder añadir o eliminar artículos de esta. Podrán añadir artículos que ellos mismos hayan creado.
- Una vez finalizada la cesta se podrá subir a un repositorio de cestas donde los demás usuarios de la aplicación podrán verla.
- Los usuarios pueden solicitar realizar la compra de las cestas del repositorio. Una vez realizada la solicitud, se abrirá un chat para permitir la comunicación entre los dos usuarios.
- Los usuarios deben rellenar un formulario de registro para crear una cuenta.
- Los usuarios podrán modificar su perfil y añadir una foto.
- Los usuarios deberán iniciar sesión en el portal para tener acceso a sus funcionalidades.
- Cuando la web disponga de una nueva versión se emitirá una notificación a los usuarios. Si aceptan la nueva versión, su página se actualizará. Si no la aceptan, la actualización se llevará a cabo automáticamente en unas horas o cuando se cierren todas las ventanas del navegador.
- La web se mostrará, aunque este en modo sin conexión.
- La parte back-end de la web se podrá convertir en un contenedor y su puesta en marcha será rápida y eficiente.
- La web adaptará su formato según el dispositivo en el que se muestre.
- El tratamiento de los datos de usuario contará con los estándares de seguridad actuales.

3 Estado del Arte

Principalmente se ha estudiado la aplicación Bring. Se trata de una aplicación muy sencilla que apuesta por lo visual para hacer más cómodo el proceso de compra. Y es que su diseño muestra las listas en forma de tarjetones con el dibujo del producto que hay que comprar para que apenas haya que leer o ponerse las gafas de cerca para conocer en detalle todo lo que hay poner en el carro. Con ello consigue que cualquier usuario disponga de listas completas, sencillas y útiles con sólo acceder a la aplicación. Tan sólo hay que fijarse en los productos en rojo para ver lo que hay que comprar, y añadir las sugerencias en verde para completar la lista. Todo ello de forma muy visual. Donde Bring también destaca es a la hora de compartir estas listas de la compra. Esto significa poder crear entre dos o más personas una misma lista, contando con todos los elementos actualizados para evitar así olvidarse de nada a la hora de comprar. Sólo hay que añadir el correo electrónico de un contacto y que éste cuente con la aplicación instalada. Lo bueno es que, además de sincronizar los nuevos productos entre los usuarios, Bring ofrece un funcionamiento sin conexión a Internet, sin que la lista se pierda al entrar a zonas sin cobertura. Pero lo que realmente llama la atención de esta aplicación son sus notificaciones. Se encuentran en la pantalla de la derecha y con ellas es posible alertar a los contactos con los que se comparte una lista que se va a salir a comprar. Una especie de aviso para añadir productos de última hora necesarios que haya que adquirir. Junto a ello también es posible ver, desde esta pantalla, los nuevos productos añadidos a una lista compartida.

4 Desarrollo

4.1 Single-page Application

Una Single-page application (SPA o aplicación de una sola página en español) es una aplicación web que interactúa con el usuario recargando dinámicamente la página en la que se encuentra actualmente, en vez de cargar completamente una nueva página del servidor. Este enfoque mejora la experiencia del usuario eliminando las interrupciones entre páginas sucesivas, haciendo que la aplicación se comporte de forma más parecida a una aplicación de escritorio. En la mayoría de los sitios web hay mucho contenido redundante. La mayora de él se mantiene igual haga lo que haga el usuario (cabeceras, pies de página, logos, etc.) y hay muchas capas y plantillas repetidas. Las SPA se ejecutan en una sola página. Una vez ha sido cargada la página inicial toda la navegación del proyecto será por medio de dicha página evitando de esta manera refrescar el browser. Se suele hacer uso de un framework o librerías JavaScript para lograr el cometido, ya que estas implementan todo lo necesario para crear un proyecto SPA y para consultar el contenido al servidor a través de AJAX o WEBSOCKETS. Por último, están son las ventajas que nos han hecho decantarnos por desarrollar una SPA:

- Las SPA son rápidas, ya que la mayoría de los recursos (HTML + CSS + Scripts) solo se cargan una vez durante la vida útil de la aplicación. Con el servidor solo se transmiten datos.
- El front-end y el back-end son dos proyectos totalmente separados uno de otro. Al desvincular esa lógica y datos, los desarrolladores pueden crear muchas formas de front-end diferentes para mostrar y usar ese servicio back-end. Con una configuración desacoplada, los desarrolladores pueden construir, implementar y experimentar con el front-end de forma completamente independiente de la tecnología de back-end subyacente.

4.2 SPA y React.js

React.js es un framework que los desarrolladores usan para crear SPA de manera eficiente. Permite crear una colección de componentes reutilizables, o utilizar componentes a los que han contribuido muchos desarrolladores. Por estas razones, React.js se usará para desarrollar la parte front-end de nuestro proyecto.

4.3 React Redux

Al trabajar con React.js muchas veces se requerirá que los componentes se comuniquen entre ellos como podemos ver en la Figura 1. React.js trabaja con el concepto de estado de un componente. Los componentes comparten sus estados entre ellos para comunicarse. La gestión del estado es esencialmente una forma de facilitar la comunicación y el intercambio de datos entre componentes. Crea una estructura de datos tangible para representar el estado de la aplicación desde la que puede leer y escribir. Funciona bien para aplicaciones con pocos componentes, pero a medida

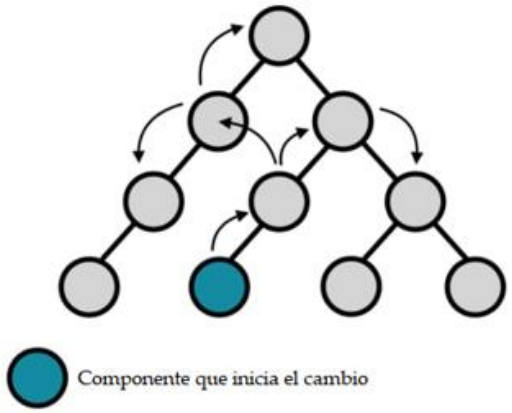

Figura 1: Comunicación React

que la aplicación crece, administrar los estados compartidos entre los componentes se convierte en una ardua tarea.

En una aplicación donde los datos se comparten entre componentes, puede resultar confuso saber dónde debería vivir un estado. Idealmente, los datos de un componente deberían residir en un solo componente, por lo que compartir datos entre componentes hermanos se vuelve difícil. Cuando un estado debe compartirse entre componentes que están muy separados en el árbol de componentes. El estado debe pasar de un componente a otro hasta que llegue a donde se necesita. Básicamente, el estado tendrá que elevarse al componente principal más cercano y al siguiente hasta que llegue a un ancestro que sea común a ambos componentes que necesitan el estado, y luego se transmite. Esto hace que el estado sea difícil de mantener y menos predecible. También significa pasar datos a componentes que no los necesitan. Está claro que la administración del estado se complica a medida que la aplicación se vuelve compleja. Es por eso por lo que usaremos una herramienta de administración de estados. Redux es una librería que nos permite trabajar con un estado general (llamado store) para centralizar la comunicación de todos los componentes. Ilustrado en la Figura 2.

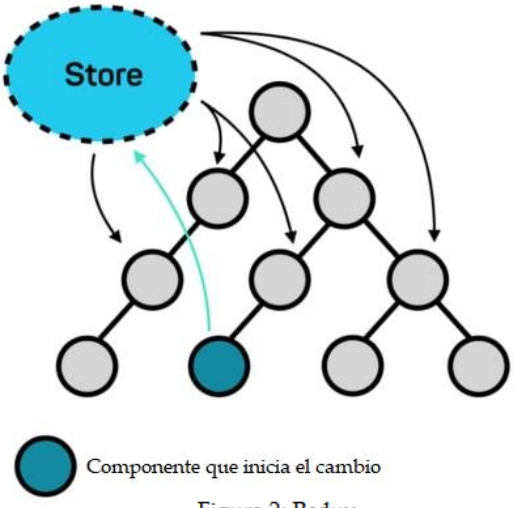

Figura 2: Redux

4.4 Service Worker

Otro concepto de desarrollo con el que trabajaremos para nuestro front-end es el Service Worker. Un Service Worker es un script que se ejecuta en segundo plano, en un hilo separado de la interfaz de usuario del navegador. La caché del Service Worker hace posible que un sitio web funcione sin conexión. Además, lo usaremos para detectar nuevas versiones de la aplicación web. Cuando realizamos cambios en nuestra aplicación y la desplegamos en producción generamos un nuevo Service Worker, que se mantiene a la espera. Los usuarios seguirán viendo contenido antiguo hasta que cierren (recargar no es suficiente) sus pestañas abiertas existentes. Con la ayuda del Service Worker podremos detectar los cambios y mandar una notificación al usuario para que, pulsando un botón, obtenga la nueva versión de la web.

4.5 API REST

Para comunicar nuestra parte front-end con el back-end usaremos API REST. Una API es una interfaz de programación de aplicaciones. Es un conjunto de reglas que permiten que los programas se comuniquen entre sí. El desarrollador crea la API en el servidor y permite que el cliente se comunique con ella. REST determina cómo se ve la API. Significa "Transferencia de Estado Representacional". Es un conjunto de reglas que los desarrolladores siguen cuando crean su API. Cada URL se denomina solicitud, mientras que los datos que se le envían se denominan respuesta.

4.7 Node.js y Express.js

Para implementar nuestra API usamos el entorno de ejecución basado en javascript Node.js. Para desarrollar con Node.js se ha usado Express.js. Express es el framework web más popular de Node.js. Proporciona mecanismos para:

- Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
- Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.

- Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la tubería de manejo de la petición.

4.8 Postgresql

Para cerrar el círculo de la parte servidor. Se ha trabajado con la herramienta de base de datos Postgresql. Postgresql es un sistema de código abierto de administración de bases de datos del tipo relacional, aunque también es posible ejecutar consultas que sean no relaciones. En este sistema, las consultas relacionales se basan en SQL, mientras que las no relacionales hacen uso de JSON. Como decíamos, se trata de un sistema de código abierto y además gratuito, y su desarrollo es llevado adelante por una gran comunidad de colaboradores de todo el mundo que día a día ponen su granito de arena para hacer de este sistema una de las opciones más sólidas a nivel de bases de datos. Interactuaremos con la base de datos por medio de la biblioteca ORM Sequelize. Una biblioteca ORM es una biblioteca que encapsula el código necesario para manipular los datos de la base de datos a la que se enlaza, por lo que ya no usa SQL; interactúas directamente con un objeto en el mismo idioma que estás usando para realizar cualquier acción en base de datos.

4.9 JSON Web Token

Autentificaremos los usuarios mediante JSON WEB TOKEN. Un JSON Web Token es un token de acceso estandarizado permite el intercambio seguro de datos entre dos partes. Contiene toda la información importante sobre una entidad, lo que implica que no hace falta consultar una base de datos ni que la sesión tenga que guardarse en el servidor (sesión sin estado).
Por este motivo, los JWT son especialmente populares en los procesos de autentificación. Con este estándar es posible cifrar mensajes cortos, dotarlos de información sobre el remitente y demostrar si este cuenta con los derechos de acceso requeridos. Los propios usuarios solo entran en contacto con el *token* de manera indirecta: por ejemplo, al introducir el nombre de usuario y la contraseña en una interfaz. La comunicación como tal entre las diferentes aplicaciones se lleva a cabo en el lado del front-end y del back-end.

Figura 3: JWT

4.10 Esquema back-end

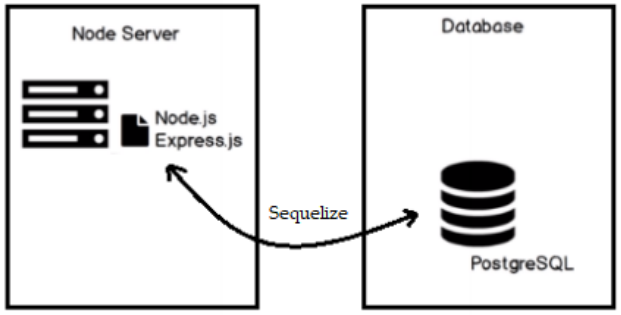

Figura 4: Back-end

4.12 Docker

Desde hace muchos años, el software empresarial se ha implementado normalmente en "bare metal" (es decir, instalado en un sistema operativo que tiene control total sobre el hardware subyacente) o en una máquina virtual (instalado en un sistema operativo que comparte el hardware con otros sistemas operativos "invitados"). Naturalmente, la instalación en "bare metal" hizo que el software fuera dolorosamente difícil de mover y de actualizar, dos limitaciones que dificultaban que respondiera con agilidad a los cambios en las necesidades comerciales.

Luego llegó la virtualización. Las plataformas de virtualización (también conocidas como "hipervisores") permitieron que varias máquinas virtuales compartieran un solo sistema físico, cada máquina virtual emulando el comportamiento de un sistema completo, con su propio sistema operativo, almacenamiento y E/S, de manera aislada. Así podría responder de manera más eficaz a los cambios en los requisitos comerciales, porque las máquinas virtuales se pueden clonar, copiar, migrar y activar o desactivar para satisfacer la demanda o conservar recursos.
También ayudaron a reducir los costes, porque se podrían consolidar más máquinas virtuales en menos máquinas físicas. Los sistemas heredados que ejecutan aplicaciones más antiguas se pueden convertir en máquinas virtuales y desmantelar físicamente para ahorrar aún más dinero.

Pero las máquinas virtuales todavía tienen algunos problemas. Son grandes (gigabytes) y cada una contiene un sistema operativo completo. El aprovisionamiento de una máquina virtual todavía lleva bastante tiempo. Finalmente, la portabilidad de las VM es limitada. Después de cierto punto, las máquinas virtuales no pueden ofrecer el tipo de velocidad, agilidad y ahorros que exigen las empresas. Los contenedores Docker funcionan un poco como las máquinas virtuales, pero de una manera mucho más específica y granular. Aíslan una sola aplicación y sus dependencias, todas las bibliotecas de software externas que la aplicación requiere para ejecutarse, tanto del sistema operativo subyacente como de otros contenedores. Todas las aplicaciones en contenedores comparten un único sistema operativo común (ya sea Linux o Windows), pero están

compartimentadas entre sí y del sistema en general.

Docker es una herramienta utilizada para aumentar la velocidad del proceso de desarrollo y despliegue del software. También ayuda a eliminar problemas específicos del entorno de programación, ya que permite replicar entornos de producción localmente. Eliminando el típico problema de "en mi pc funcionaba". Entender y utilizar esta tecnología era un paso importante en el proyecto y ha dado resultados muy satisfactorios. La hemos utilizado para emular una entrega continua del software ya que los contenedores de Docker facilitan la puesta en producción de nuevas versiones de software, y la reversión rápida a una versión anterior si es necesario. Por eso se ha incorporado a nuestro ciclo de vida del software y se ha utilizado para convertir en contenedor nuestra API REST junto con la base de datos que la acompaña. Así, cualquiera que disponga de una instalación de Docker en su ordenador puede poner en marcha la parte back-end del proyecto con la API REST, base de datos y todas sus relaciones listas para su funcionamiento.

5 METODOLOGÍA

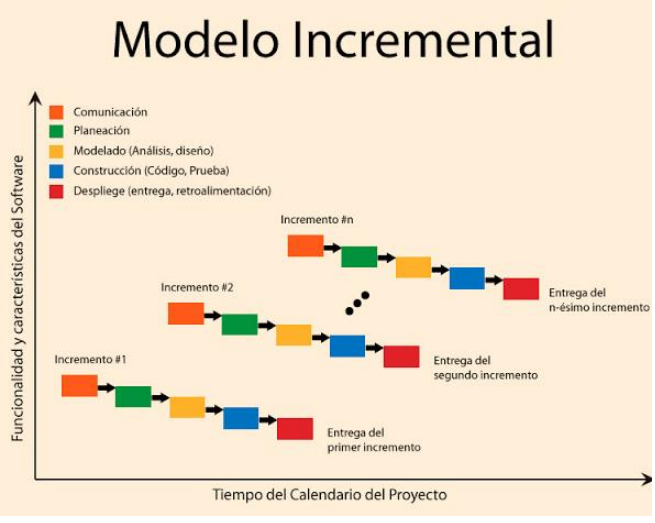

Figura 5: Modelo incremental

Se ha decidido trabajar con una implementación en modo iterativo e incremental para que el proyecto pueda adaptarse a posibles cambios en los requisitos y poder verificar que el trabajo se está llevando a cabo según los objetivos establecidos. En cada iteración se han realizado las actividades fundamentales de desarrollo del software: análisis de requerimientos, diseño, implementación y pruebas. En estas iteraciones de desarrollo se ha emulado la entrega de software al cliente. Trabajando con un sistema de entrega continua y entregas intermedias utilizando contenedores Docker.

5 PLANIFICACIÓN

La planificación del proyecto ha sido la siguiente:

- Semana 1: Reunión inicial con el tutor.
- Semana 2: Estudiar y analizar diferentes herramientas de desarrollo web.
- Semana 3: Estudiar y analizar diferentes herramientas de desarrollo web.
- Semana 4: Realización y entrega Informe inicial.
- Semana 5: Análisis de requisitos.
- Semana 6: Diseño de la aplicación (UML), formación sobre Docker en Node.js/ReactJS.
- Semana 7: Implementación de la aplicación.
- Semana 8: Implementación de la aplicación.
- Semana 9: Realización y entrega del Informe de progreso I e implementación de la aplicación.
- Semana 10: Implementación de la aplicación.
- Semana 11: Iteración Software.
- Semana 12: Iteración Software.
- Semana 13: Iteración Software.
- Semana 14: Realización y entrega Informe de progreso II. Iteración Software.
- Semana 15: Preparación presentación, realización Dosier e Informe Final. Iteración Software.
- Semana 16: Preparación presentación, realización Dosier e Informe Final.
- Semana 17: Preparación presentación, realización Dosier e Informe Final.
- Semana 18: Preparación presentación, realización Dosier e Informe Final.
- Semana 19: Entrega Dosier del TFG.
- Semana 20: Entrega Informe Final.
- Semana 21: Presentación y debate público del TFG.

6 ANÁLISIS DE REQUISITOS

6.1 Requisitos no funcionales

- La aplicación debe ser intuitiva y tener un uso tan sencillo como sea posible.
- Los colores, sonidos y animaciones deben ser suaves, agradables y tienen que parecerse a otras aplicaciones conocidas con funcionalidad parecida.
- La aplicación tiene que responder de forma rápida y fluida al uso del usuario.
- La aplicación debe ser responsive y se debe poder usar en dispositivos tablet, móvil y pc.
- Las imágenes e iconos deben tener unas medidas que faciliten su lectura.
- La aplicación permite trabajar en modo offline y cuando recupera conexión actualiza los datos.
- La aplicación informa al usuario de nuevas versiones disponibles.

6.2 Requisitos funcionales

- La aplicación permitirá crear una cuenta de usuario asociada a un correo electrónico.
- Sin una cuenta, no se podrá acceder a la aplicación.

- La aplicación dispondrá de un formulario de registro.
- Los formularios deben cumplir con los estándares de calidad actuales en cuanto al tratamiento de los datos.
- La aplicación tendrá un menú para poder navegar por las diferentes opciones.
- Los usuarios deben poder crear un listado de la compra.
- Los usuarios deben poder utilizar un buscador para seleccionar artículos.
- Los usuarios deben poder añadir o eliminar artículos de un listado de la compra.
- Los usuarios podrán crear sus propios artículos.
- Los usuarios no pueden modificar cestas que no sea de su propiedad.
- La cesta se podrá subir a un repositorio de cestas
- Los usuarios podrán utilizar un buscador para ver cestas subidas al repositorio de cestas.
- Los usuarios pueden solicitar realizar la compra de las cestas del repositorio.
- Se abrirá un chat para permitir la comunicación entre usuarios propietarios de una cesta y usuarios compradores de esta.

7 Resultados

A continuación, se describen los resultados obtenidos en el proyecto:

- El usuario que disponga de una cuenta podrá acceder al sistema mediante un formulario de acceso. Lo vemos en la Figura 6. Una vez conectado, la web guardará la sesión del usuario en el almacenamiento local, así como un token JSON WEB TOKEN que lo autenticará de forma única en el sistema y servirá para realizar comprobaciones de seguridad relacionadas con peticiones a la parte back-end.

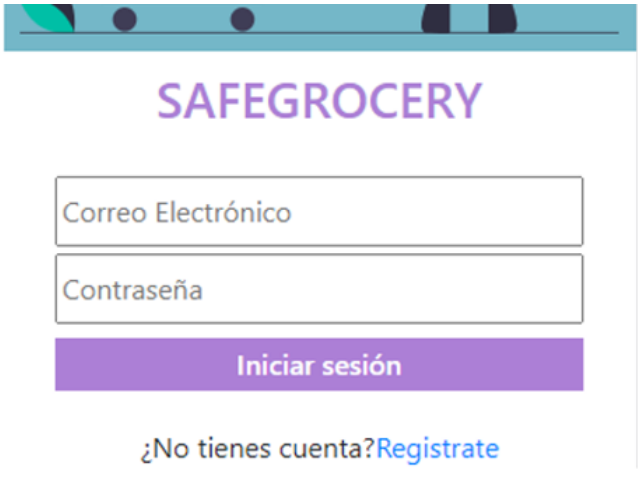

Figura 6: Inicio sesión

- Los usuarios pueden acceder al apartado "Mi Cesta", Figura 9, para añadir artículos escribiendo en el buscador. Si nunca habían creado ese artículo, lo tendrán que añadir pulsando la tarjeta con el signo de sumar. Se asigna una imagen según el nombre del artículo. Con sólo un click se añade el artículo a la cesta y se sigue el mismo proceso para eliminarlo. Una vez, la cesta tenga los artículos deseados, se puede enviar al listado de cestas a comprar. La web indicará que su cesta ha sido enviada y mostrará sus artículos.

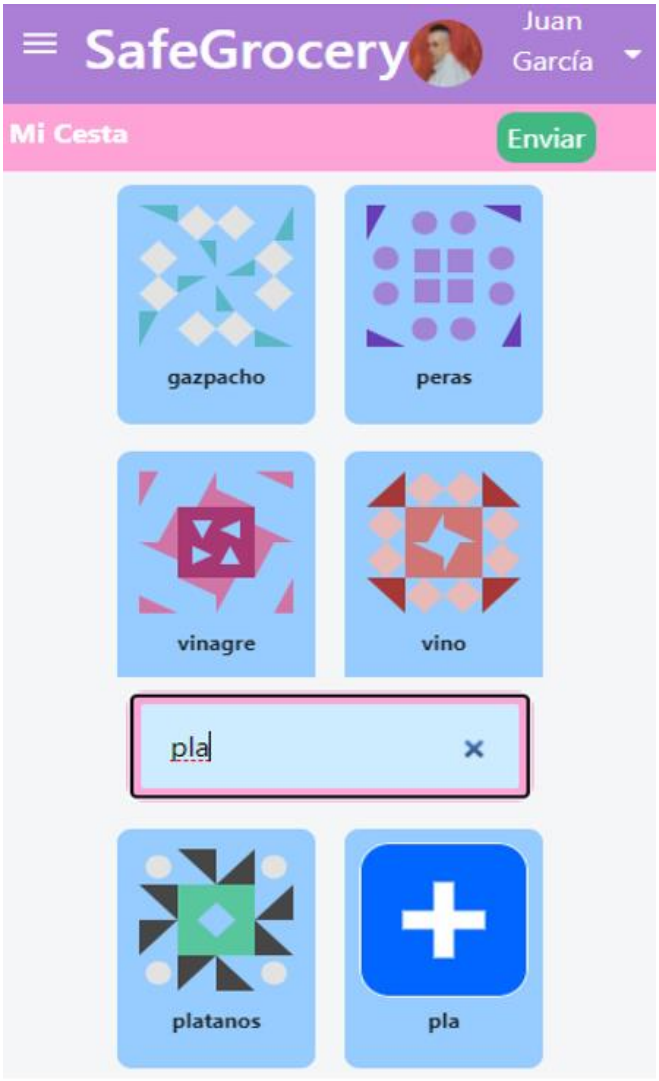

Figura 7: Mi cesta

- El chat permite compartir iconos e imágenes.
- La web dispone de menú lateral para moverte por sus diferentes opciones. También permite a los usuarios cerrar su sesión y las sesiones de usuario tienen un tiempo limitado.

- Un usuario podrá ver las cestas a comprar y sus productos.
- Pulsar el botón de compra le asignará automáticamente esa cesta y se abrirá un chat entre el usuario creador y el comprador. Sólo el usuario comprador puede finalizar la transacción pulsando el botón "Finalizar cesta".

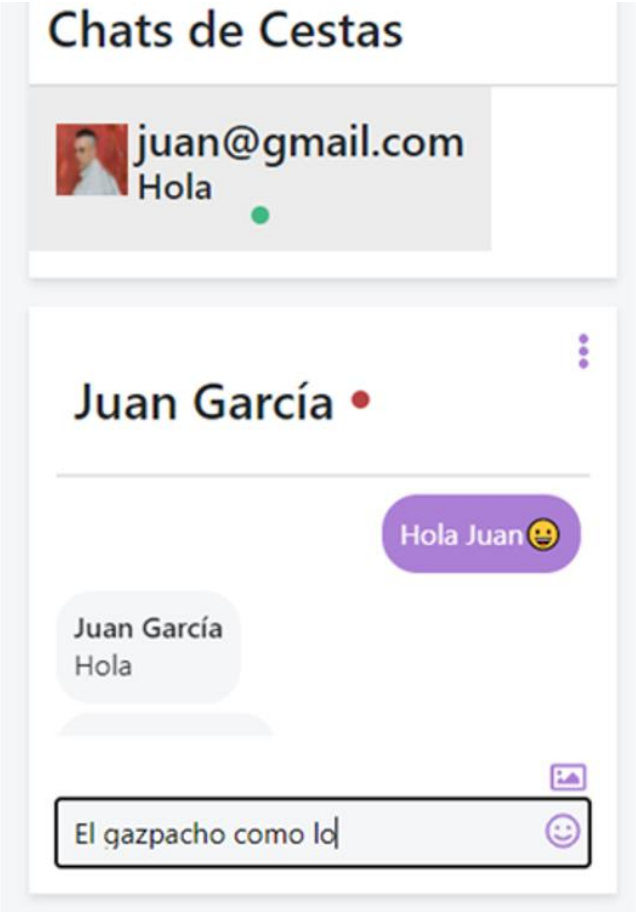

Figura 8: Chat

- La parte back-end se puede convertir en un contenedor usando Docker. Mantiene una estructura de API y base de datos mediante Docker Compose. La base de datos se genera automáticamente y solo se necesita de unos scripts de migración para crear todas las relaciones entre tablas.

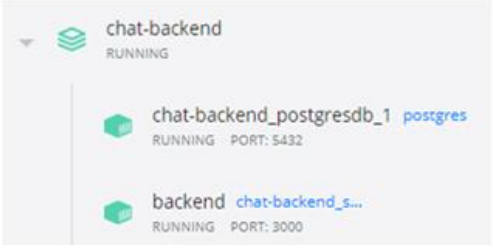

Figura 9: Docker Compose

- Los usuarios pueden listar las cestas enviadas y sus artículos disponibles.
- El chat utiliza sockets para enviar mensajes. Se envía información instantánea y se puede observar cuando un usuario está escribiendo.
- La web cuenta con una base de datos relacional utilizando PostgreSql. Utiliza el ORM Sequelize para realizar operaciones mediante código javascript. (*Apéndice A1*)
- La web adapta su formato a cualquier dispositivo en el cual se visualice.
- La web dispone de un formulario de registro para poder crear una cuenta y así acceder al sistema. El formulario cuenta con sistema de encriptación de contraseña que se lanzará cuando almacenamos los datos en la base de datos.

Figura 10: Registro de usuario

- Se permite a los usuarios modificar los datos de su perfil: cambiar su contraseña, nombre, imagen, etc.

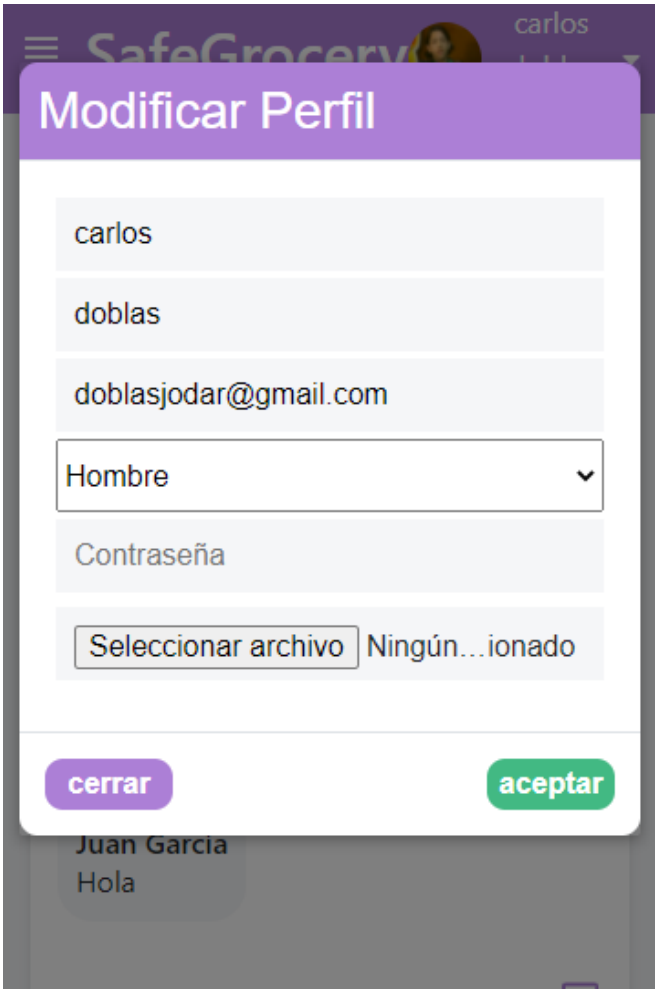

Figura 11: Modificar perfil

8 Conclusiones

Creo que el proyecto se ha resuelto de forma satisfactoria. Tanto la implementación de un control de acceso para los usuarios, la creación y gestión de cestas y un chat para comunicarse. Lo más complicado y lo que ha llevado más tiempo del proyecto ha sido la formación necesaria para desarrollar la web y la implantación de las bases del proyecto para poder empezar a trabajar en las funcionalidades más visibles para los usuarios. Configurar back-end con base de datos con migraciones, dockerización y el enlace con el front-end ha sido una tarea complicada. Esto ha llevado a retrasar las etapas de desarrollo puro y provocar desajustes de tiempo. Pero era necesaria para emular la entrega hacia cliente. La metodología de desarrollo incremental no ha sido la mejor opción ya que este proyecto tenía una gran carga de tiempo dedicada en establecer el marco de trabajo para poder utilizar esa metodología. Una vez terminado, tenemos un entorno que nos permite realizar desarrollo con entrega continua de forma eficiente. Valoro mucho los conocimientos obtenidos sobre las tecnologías trabajadas y espero que me sirvan en mi ámbito laboral. La intención secundaria de utilizar estas herramientas era obtener una base sólida de conocimiento de las mejores prácticas de programación web de la actualidad y creo que esa intención se ha cumplido con creces. He echado en falta trabajar con algún compañero en este proyecto. Creo que el trabajo se podría haber dividido en implementación de la aplicación y automatización de procesos de entrega. Ya que, al realizar todo el trabajo, la aplicación no es tan compleja como me hubiera gustado. Se podría haber ampliado mucho su funcionalidad y el trabajo en equipo le hubiera dado una capa extra de complejidad para integrar las partes y utilizar herramientas de gestión de la configuración. Como funcionalidades que me gustaría haber añadido y por las que daría continuidad al proyecto, cito las siguientes: aplicar testeo unitario utilizando la librería Jest, convertir la aplicación en una progressive web app, implantar la web en un servidor https, mejorar la usabilidad de la web y la información en formularios que reciben los usuarios, ampliar las funcionalidades(actualmente se puede tener una cesta de compra activa) y, por último, trabajar con un entorno serverless utilizando Amazon Web Services.

9 Agradecimientos

Quería agradecer a mi familia el apoyo durante todos estos años de mi carrera. A mi novia, que me ha ayudado en todo momento. Sin ellos esto no hubiera sido posible. Por último, agradecer la implicación de Cris Romero, mi tutora del TFG, por la predisposición en todo momento, la facilidad de transmitir y los consejos que me ha aportado durante todo el proyecto.

Bibliografía

1] React. (2021). [online] disponible en: https://reactjs.org/ [accedido 14 Mar. 2021].
[2] Node.js. (2021). [online] disponible en: https://nodejs.org/en/ [accedido 14 Mar. 2021].
[3] Docker. (2021). [online] disponible en: https://www.docker.com/ [accedido 14 Mar. 2021].
[4] Obey the Testing Goat!. (2021). [online] disponible en: https://www.obeythetestinggoat.com/ [accedido 14 Mar. 2021].
[5] Redux. (2021). [online] disponible en: https://redux.js.org/ [accedido 14 Mar. 2021].
[6] Postgresql. (2021). [online] disponible en: https://www.postgresql.org/ [accedido 10 Mar. 2021].
[7] Sequelize. (2021). [online] disponible en: https://sequelize.org/ [accedido 10 Abr. 2021].
[8] bcrypt. (2021). [online] disponible en: https://www.npmjs.com/package/bcrypt [accedido 13 Abr. 2021].
[9] Docker-Compose. (2021). [online] disponible en: https://docs.docker.com/compose/ [accedido 30 Abr. 2021].
[10] MySQL (2021). [online] disponible en: https://www.mysql.com/ [accedido 30 Abr. 2021].
[11] PWA. (2021). [online] disponible en: https://web.dev/progressive-web-apps/ [accedido 10 Abr. 2021].
[12] Jest. (2021). [online] disponible en: https://jestjs.io/ [accedido 2 Mar. 2021].

APÉNDICE

A1. UML BASE DE DATOS