



MÓDULO PROYECTO

CFGS Desarrollo de Aplicaciones

Multiplataforma

Informática y Comunicaciones

App en Ionic: BazarShop

Tutor individual: Silvia Pedrón Hermosa

Tutor colectivo: Cristina Silván Pardo

Año: 2024

Fecha de presentación: 28/05/2024

Nombre y Apellidos: Roberto Toquero Fernández

Email: roberto.toqfer@educa.jcyl.es

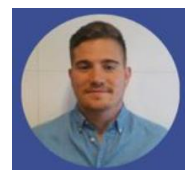


Tabla de contenido

1	Descripción general del proyecto.....	4
1.1	Objetivos	4
1.2	Cuestiones metodológicas	5
1.3	Entorno de trabajo (tecnologías de desarrollo y herramientas)	6
2	Descripción general del producto.....	12
2.1	Visión general del sistema: límites del sistema, funcionalidades básicas, usuarios y/o otros sistemas con los que pueda interactuar.....	12
2.2	Descripción breve de métodos, técnicas o arquitecturas(m/t/a) utilizadas.	15
2.3	Despliegue de la aplicación indicando plataforma tecnológica, instalación de la aplicación y puesta en marcha	20
3	Planificación y presupuesto	26
4	Documentación Técnica: análisis, diseño, implementación y pruebas.....	28
4.1	Especificación de requisitos	28
4.2	Análisis del sistema.....	29
4.3	Diseño del sistema:.....	30
4.3.1	Diseño de la Base de Datos.....	30
4.3.2	Diseño de la Interfaz de usuario.	35
4.3.3	Diseño de la Aplicación.....	40
4.4	Implementación:.....	41
4.4.1	Entorno de desarrollo.....	41
4.4.2	Estructura del código.	42

4.4.3 Cuestiones de diseño e implementación reseñables.	50
4.5 Pruebas.	55
5 Manuales de usuario	63
5.1 Manual de usuario	¡Error! Marcador no definido.
5.2 Manual de instalación	72
6 Conclusiones y posibles ampliaciones	63
7 Bibliografía	79
8 Anexos	80

1. Descripción general del proyecto

El proyecto consiste en el desarrollo y puesta en marcha de la aplicación BazarShop. Se trata de una aplicación multiplataforma desarrollada con el **framework Ionic** y **Angular**, diseñada para ayudar a los usuarios a gestionar sus productos. La aplicación permite almacenar y contar productos de manera eficiente, proporcionando una solución integral para la organización de inventarios personales o comerciales. Los datos se almacenan en **Firestore**, una base de datos en línea que ofrece sincronización en tiempo real y escalabilidad. BazarShop se enfoca en la facilidad de uso y la accesibilidad, asegurando que los usuarios puedan acceder a sus inventarios desde cualquier dispositivo.

1.1 Objetivos

Los objetivos principales que se buscan alcanzar con el proyecto son:

Aprender tecnologías no vistas en clase: Con este proyecto, quiero obtener experiencia práctica en el uso de tecnologías modernas que no hemos visto en clase, como Ionic y Angular para desarrollar aplicaciones multiplataforma, y Firestore para la gestión de bases de datos en tiempo real.

Desarrollar una app útil: Mi objetivo es crear una aplicación funcional que ayude a los usuarios a gestionar sus productos de manera eficiente y poder ofrecer herramientas para almacenar y contar inventarios, que puedan ser útiles tanto para uso personal como comercial.

Implementar mejores prácticas de desarrollo: Quiero aplicar principios de diseño y desarrollo de software, como la modularidad, la reutilización de código y la integración continua, para asegurarme de producir un código limpio, entendible y fácil de mantener.

Mejorar habilidades de gestión de proyectos: Con este proyecto, espero mejorar mis habilidades en la gestión del ciclo de vida del software. Aspectos importantes como planificar, diseñar, implementar y lanzar un proyecto completo, y desarrollar habilidades en la resolución de problemas.

Explorar el desarrollo multiplataforma: Aprovechar las capacidades de Ionic me permite desarrollar una única base de código que funcione en múltiples plataformas (iOS y Android), lo cual maximiza el alcance y la accesibilidad de la aplicación.

Preparación para el mercado laboral: Obtener experiencia práctica en un proyecto real donde mostrar mis habilidades en el desarrollo de aplicaciones móviles y web modernas.

Control de Versiones: Implementar y gestionar el control de versiones utilizando herramientas como Git y Github es clave, ya que me permitirá mantener un historial detallado de los cambios y poder realizarlo en varios equipos al mismo tiempo.

1.2 Cuestiones metodológicas

Para desarrollar la aplicación BazarShop, he seguido una metodología práctica y orientada a la experimentación, que combina la investigación en línea con el método de prueba y error. Pasos que se han seguido:

1. **Revisión de la Documentación:** Inicialmente, busqué documentación en línea y revisé múltiples fuentes para comprender las tecnologías involucradas en el proyecto. Esto incluyó la lectura de documentación oficial y artículos técnicos sobre Ionic, Angular y Firebase.
2. **Aprendizaje a través de Tutoriales:** Para familiarizarme con las funcionalidades y capacidades de las tecnologías seleccionadas, seguí varios tutoriales en línea. Estos tutoriales me proporcionaron una base sólida y ejemplos prácticos que pude adaptar a las necesidades específicas de mi proyecto.
3. **Método de Prueba y Error:** Implementé un enfoque de prueba y error para experimentar con los componentes y servicios que ofrece Ionic. Este método me permitió iterar rápidamente, identificar problemas, probar soluciones y refinar la implementación de las funcionalidades.

4. **Diseño de la Arquitectura del Sistema:** Diseñé la arquitectura de la aplicación, asegurándome de que fuese modular y escalable. Definí componentes clave, servicios y estructuras de datos, siguiendo principios de diseño de software modernos.
5. **Implementación Iterativa:** Adopté un enfoque iterativo para el desarrollo, implementando funcionalidades básicas primero y luego agregando características adicionales en ciclos sucesivos. Cada iteración incluyó fases de codificación, prueba y revisión.
6. **Control de Versiones:** Utilicé Git para el control de versiones, lo que me permitió mantener un historial detallado de los cambios, gestionar colaboraciones y asegurar la integridad del proyecto a lo largo del tiempo. Realicé commits frecuentes para documentar el progreso y facilitar la reversión de cambios en caso de errores.
7. **Pruebas y Validación:** Realicé pruebas continuas durante el desarrollo para identificar y corregir errores. Utilicé pruebas unitarias y de integración para asegurar que cada componente funcionara correctamente tanto de manera individual como en conjunto.
8. **Documentación del Proyecto:** Documenté cada etapa del desarrollo, incluyendo el diseño, la implementación y las pruebas. Esta documentación no solo sirve como referencia para el mantenimiento futuro, sino también como evidencia del proceso metodológico seguido.

1.3 Entorno de trabajo (tecnologías de desarrollo y herramientas)

Para el desarrollo del proyecto se han empleado las siguientes tecnologías:

IONIC

Ionic es un framework para la creación de aplicaciones móviles híbridas, compatible con iOS y Android. Ofrece un amplio catálogo de componentes prediseñados y profundamente probados, típicos en aplicaciones móviles híbridas. Además, proporciona librerías para la interacción nativa con el dispositivo, permitiendo acceso a funcionalidades como contactos, GPS y cámara. Ionic puede usar Angular, React o Vue como base.

¿Qué es una App híbrida?

Una aplicación híbrida combina herramientas web y nativas, permitiendo la creación de una aplicación móvil con un único código base. Esto proporciona flexibilidad para implementarla en diferentes sistemas operativos, ofreciendo un rendimiento similar al de las aplicaciones nativas y una experiencia de usuario fluida.

Ventajas de Ionic:

Curva de aprendizaje suave: Los desarrolladores familiarizados con lenguajes de desarrollo web encontrarán más fácil adaptarse a Ionic.

Coste económico: Comparado con el desarrollo nativo, Ionic es más rentable. Un solo equipo de desarrollo puede trabajar en aplicaciones para múltiples plataformas.

Desarrollo ágil: Los resultados se obtienen de manera rápida y eficiente.

Acceso a características nativas: Permite utilizar el hardware del dispositivo y acceder a herramientas nativas.

Desarrollo ligero: Puede utilizarse el navegador en lugar de múltiples emuladores, lo que es más ligero para dispositivos con menor memoria RAM.

¿Por qué escoger Ionic?

Ionic es ideal para aplicaciones que manejan una cantidad considerable de datos, se conectan a servicios externos y requieren una interacción ágil con el usuario.

La experiencia del equipo de desarrollo es importante. Si el equipo tiene experiencia en lenguajes nativos, puede resultar contraproducente migrar a Ionic. Sin embargo, para equipos familiarizados con Angular, React o Vue, Ionic es una elección excelente.

Además, permite formar un equipo multidisciplinar en lugar de mantener varios equipos específicos para diferentes plataformas. Para aplicaciones que no dependen del sistema operativo específico, Ionic es una herramienta esencial.

ANGULAR

Para el desarrollo de aplicaciones con Ionic, he escogido Angular como la base del

framework. Angular es una plataforma de desarrollo para aplicaciones web que facilita la creación de aplicaciones dinámicas y robustas. Desarrollado por Google, Angular ofrece un conjunto de herramientas y librerías que permiten a los desarrolladores construir aplicaciones de manera eficiente y estructurada.

¿Qué es Angular?

Angular es un framework de desarrollo web basado en **TypeScript** que proporciona una arquitectura robusta para crear aplicaciones de una sola página (SPA). Utiliza componentes reutilizables, inyección de dependencias y un sistema de enlace de datos bidireccional que permite mantener el modelo y la vista sincronizados automáticamente. Angular es conocido por su escalabilidad, modularidad y capacidad para manejar aplicaciones complejas.

¿Por qué Angular se combina bien con Ionic?

- **Compatibilidad nativa:** Ionic fue diseñado originalmente para trabajar con Angular, lo que significa que muchas de las funcionalidades y componentes de Ionic están optimizados para Angular. Esta compatibilidad asegura una integración suave y sin problemas.
- **Componentes reutilizables:** Angular y Ionic ambos promueven el uso de componentes modulares y reutilizables. Esto facilita la construcción de interfaces de usuario complejas y coherentes, ya que los componentes pueden ser creados una vez y utilizados en múltiples lugares.
- **Arquitectura estructurada:** Angular ofrece una arquitectura clara y bien definida que ayuda a mantener el código organizado y mantenible. Esto es especialmente útil en proyectos grandes y complejos, donde la estructura del código es crucial para el éxito del proyecto.
- **Herramientas y librerías:** Angular proporciona una amplia gama de herramientas y librerías que aceleran el desarrollo y mejoran la eficiencia. Estas herramientas incluyen Angular CLI (Command Line Interface) para generar y gestionar proyectos, y un ecosistema robusto de extensiones y módulos que complementan las capacidades de Ionic.
- **Rendimiento y eficiencia:** El enlace de datos bidireccional de Angular y su capacidad para manejar la lógica del lado del cliente de manera eficiente,

combinados con los componentes optimizados de Ionic, resultan en aplicaciones de alto rendimiento con una experiencia de usuario fluida.

- **Comunidad y soporte:** Angular cuenta con una gran comunidad de desarrolladores y un amplio soporte, lo que facilita encontrar soluciones a problemas y acceder a recursos educativos.

NODE.JS

Se requiere la instalación porque Ionic depende de Node.js y su gestor de paquetes, npm (Node Package Manager), para manejar sus dependencias y herramientas.

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite a los desarrolladores ejecutar código JavaScript fuera de un navegador. Se basa en el motor V8 de Google Chrome y proporciona una arquitectura orientada a eventos y no bloqueante, lo que lo hace ideal para aplicaciones que requieren alta escalabilidad y rendimiento, como servidores web y aplicaciones de tiempo real.

Ionic utiliza Node.js para gestionar sus herramientas y dependencias. Node.js viene con npm, un gestor de paquetes que permite instalar y gestionar bibliotecas y herramientas de JavaScript, incluyendo el propio Ionic. Al usar npm, puedes instalar, actualizar y desinstalar paquetes de manera fácil y eficiente, lo que es fundamental para el desarrollo moderno de aplicaciones.

FIREBASE

Para almacenar los datos del proyecto, he utilizado Firebase, una plataforma desarrollada por Google que proporciona una variedad de herramientas y servicios para el desarrollo de aplicaciones web y móviles. Firebase es útil para gestionar la base de datos, la autenticación de usuarios, el almacenamiento de archivos y la implementación de aplicaciones, entre otras funcionalidades.

Entre sus principales características se encuentran:

- Base de datos en tiempo real
- Base de datos no relacional (NoSQL)
- Autenticación de usuarios

- Almacenamiento de archivos
- Alojamiento de aplicaciones
- Mensajería en la nube
- Funciones en la nube

GIT Y GITHUB

Para almacenar el proyecto y gestionar sus cambios de manera eficiente, he utilizado Git y un repositorio en GitHub.

Git es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar y registrar los cambios en el código fuente a lo largo del tiempo. Facilita el seguimiento de las modificaciones, la colaboración en equipo y la reversión de cambios si es necesario.

Algunas de sus características son:

- **Control de versiones distribuido:** Cada desarrollador tiene una copia completa del historial del proyecto en su máquina local, lo que permite trabajar sin conexión y realizar operaciones rápidas.
- **Ramas y fusiones:** Git permite crear ramas para desarrollar nuevas características o realizar pruebas sin afectar el código principal. Estas ramas pueden fusionarse (merge) de nuevo a la rama principal (main o master) cuando estén listas.
- **Registro de cambios (commits):** Cada cambio en el código se registra como un "commit", que incluye un mensaje descriptivo y un identificador único. Esto permite un seguimiento detallado del historial del proyecto.
- **Reversión de cambios:** Si se introduce un error, Git permite revertir a versiones anteriores del código fácilmente.
- **Colaboración:** Facilita la colaboración entre desarrolladores y equipos, permitiendo que múltiples personas trabajen en el mismo proyecto simultáneamente.

GitHub es una plataforma de alojamiento de código que utiliza Git para el control de versiones. Es uno de los servicios más populares para el desarrollo de software colaborativo.

Características principales:

- **Repositorios:** Un repositorio (repo) es un espacio donde se almacena todo el historial del proyecto. Puede ser público (visible para todos) o privado (accesible solo para usuarios autorizados).
- **Interfaz web:** Ofrece una interfaz web fácil de usar para gestionar repositorios, visualizar el historial de commits, ramas, y realizar revisiones de código.
- **Colaboración y revisiones:** Facilita la colaboración mediante pull requests, que permiten a los desarrolladores proponer cambios y discutirlos antes de fusionarlos con la rama principal.
- **Issues y proyectos:** GitHub incluye herramientas para la gestión de proyectos, como issues (para seguimiento de tareas y bugs) y proyectos (tableros Kanban para organizar el trabajo).
- **Integraciones y automatización:** Permite integrarse con numerosas herramientas de desarrollo y CI/CD (Integración Continua y Despliegue Continuo) para automatizar pruebas, despliegues y otras tareas.

VISUAL STUDIO CODE

Para el desarrollo de la aplicación y la introducción de librerías he empleado este Entorno de Desarrollo y su línea de comandos integrada.

CAPACITOR

Capacitor es una plataforma de código abierto creada por Ionic para desarrollar aplicaciones web que funcionen de manera nativa en iOS, Android y la web desde una base de código única. Actúa como un puente entre el código web y las funcionalidades nativas del dispositivo, permitiendo el acceso a características específicas del hardware y el sistema operativo, como la cámara, el GPS y el almacenamiento local.

Juntos, Ionic y Capacitor permiten construir aplicaciones híbridas de alto rendimiento con una experiencia de usuario similar a la de las aplicaciones nativas.

Al combinar Ionic con Capacitor, se pueden utilizar componentes de interfaz de usuario preconstruidos como APIs y herramientas de Ionic para diseñar aplicaciones atractivas. Esto simplifica el desarrollo y la distribución de aplicaciones en múltiples plataformas, manteniendo un único código base.

2 Descripción general del producto

2.1 Visión general del sistema: límites del sistema, funcionalidades básicas, usuarios y/o otros sistemas con los que pueda interactuar.

Límites del Sistema:

BazarShop es una aplicación móvil multiplataforma que permite a los usuarios gestionar sus productos y mantener un inventario actualizado. Desarrollada con Ionic y Angular, la aplicación está diseñada para funcionar en dispositivos iOS y Android. Los datos se almacenan en Firebase, lo que garantiza la sincronización en tiempo real y la escalabilidad. La aplicación también integra una API de Capacitor para acceder a las funcionalidades nativas, en este caso la cámara.

Funcionalidades Básicas:

Gestión de Productos:

Agregar productos: Los usuarios pueden añadir nuevos productos al inventario con detalles como nombre, cantidad, precio.

Editar productos: Permite modificar la información de productos existentes.

Eliminar productos: Los usuarios pueden eliminar productos del inventario.

Visualización del inventario: Los usuarios pueden ver una lista completa de todos los productos almacenados de mayor a menor cantidad de unidades.

Obtención de ganancias: El sistema calcula automáticamente las ganancias potenciales de cada producto y el total que puede generar.

Autenticación y Gestión de Usuarios:

Login: La aplicación posee una pantalla de inicio de sesión donde los usuarios pueden ingresar con su correo electrónico y contraseña.

Registro: Los nuevos usuarios pueden registrarse proporcionando su nombre, correo electrónico y contraseña.

Recuperación de contraseña: Utilizando Firebase, los usuarios pueden recuperar su contraseña en caso de olvido a través de un proceso de restablecimiento de contraseña enviado a su correo electrónico.

Actualización y eliminación de usuarios: Los usuarios pueden actualizar sus datos personales y eliminar su cuenta desde la sección de perfil.

Ajustes de la Aplicación:

Modo oscuro/luminoso: Los usuarios tienen la opción de visualizar la aplicación en modo oscuro o luminoso, mejorando la accesibilidad y personalización según sus preferencias.

Interfaz de Usuario:

Pantalla de inicio (Landing Page) interactiva:

La aplicación cuenta con una pantalla de inicio atractiva e interactiva que da la bienvenida a los usuarios. Esta pantalla ofrece una vista general de las funcionalidades de la aplicación y facilita el acceso a la pantalla de login y registro.

La landing page puede incluir elementos visuales atractivos, botones de acceso rápido, y una breve descripción de las principales características y beneficios de BazarShop, para captar el interés de nuevos usuarios.

Sincronización en Tiempo Real:

Actualización en tiempo real: Los cambios realizados en el inventario se sincronizan automáticamente en todos los dispositivos conectados, gracias a Firebase.

Accesibilidad:

Multiplataforma: Disponible para dispositivos iOS y Android, permitiendo a los usuarios acceder a su inventario desde cualquier dispositivo.

Interfaz de usuario intuitiva: Diseñada para ser fácil de usar, con una navegación sencilla y clara.

Funcionalidades de API Capacitor:

Acceso a la cámara: Permite a los usuarios tomar fotos de los productos y añadirlas al inventario. Además de agregar una foto de perfil y modificarla cuando se desee.

Posibles usuarios del Sistema:

Personas que desean mantener un registro de sus pertenencias personales.
Usuarios que gestionan colecciones personales (libros, ropa, gadgets, etc.).

Propietarios de pequeñas tiendas que necesitan una herramienta sencilla para gestionar su inventario o comerciantes que requieren una solución asequible para controlar su stock y productos.

Departamentos dentro de empresas que manejan inventarios pequeños u organizaciones que necesitan una herramienta para gestionar suministros y equipos.

Interacciones con Otros Sistemas:

Firestore:

Autenticación: Integración con Firebase Authentication para gestionar el inicio de sesión y registro de usuarios.

Base de datos: Uso de Firebase Database o Firestore para almacenar y sincronizar datos en tiempo real.

API de Capacitor:

Cámara: Para capturar imágenes de los productos.

Plataformas iOS y Android:

Compatibilidad y despliegue: La aplicación está diseñada para funcionar de manera óptima en ambos sistemas operativos, aprovechando sus características nativas mediante Capacitor.

2.2 Descripción breve de métodos, técnicas o arquitecturas(m/t/a) utilizadas.

Arquitectura del Proyecto

Modelo-Vista-Controlador (MVC):

La aplicación sigue el patrón MVC, permitiendo una separación clara entre la lógica, la interfaz de usuario y el control de la aplicación.

Modelo: Gestiona los datos y la lógica del negocio. Se integra con Firebase para la gestión de datos en tiempo real, proporcionando una base de datos NoSQL y soporte para autenticación.

Vista: Se encarga de la presentación de la información al usuario, utilizando componentes de Ionic para crear una interfaz de usuario atractiva y reutilizable.

Controlador: Maneja la interacción del usuario, actualizando el modelo y la vista en consecuencia mediante servicios de Angular.

Métodos y Técnicas de Desarrollo:

Desarrollo Multiplataforma con Ionic y Angular:

Ionic Framework: Facilita la creación de la interfaz de usuario con componentes estilizados que ofrecen una experiencia nativa en iOS y Android. Ionic utiliza tecnologías web estándar (**HTML, CSS, JavaScript**) y se integra perfectamente con Angular.

Angular: Framework de desarrollo frontend utilizado para manejar la lógica de la aplicación y la manipulación eficiente a través de componentes, directivas y servicios.

Sincronización y Almacenamiento en Tiempo Real con Firebase:

Firestore Database: Proporciona una base de datos NoSQL en tiempo real que almacena los datos de los productos y usuarios, garantizando la sincronización inmediata entre dispositivos.

Firestore Authentication: Gestiona el registro, inicio de sesión y recuperación de contraseñas, facilitando una autenticación segura y sencilla para los usuarios.

Firestore Cloud Functions: Permite ejecutar funciones en la nube en respuesta a eventos específicos, en concreto el restablecimiento de contraseña.

Para ilustrar cómo se han implementado estas técnicas, a continuación, se presenta un ejemplo del servicio de Angular para la creación de productos desde Firestore y como lo llamo desde el componente para añadir productos:

```
import { Injectable, inject } from '@angular/core';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { createUserWithEmailAndPassword, getAuth, signInWithEmailAndPassword,
updateProfile, sendPasswordResetEmail, updateEmail, sendEmailVerification,
updatePassword } from 'firebase/auth';
import { User } from '../models/user.model';
import { AngularFireStore } from '@angular/fire/compat/firestore';
import { getFirestore, setDoc, doc, getDoc, addDoc, collection,
collectionData, query, updateDoc, deleteDoc, docData } from
'@angular/fire/firestore';
import { UtilsService } from '../utils.service';
import { AngularFireStorage } from '@angular/fire/compat/storage';
import { getStorage, uploadString, ref, getDownloadURL, deleteObject } from
'firebase/storage';

@Injectable({
  providedIn: 'root'
})
export class FirebaseService {

  auth = inject(AngularFireAuth);
  firestore = inject(AngularFirestore);
  storage = inject(AngularFireStorage);
```



```
utilsSve = inject(UtilsService);

// Añadir documento

addDocument(path: string, data: any) {
  return addDoc(collection(getFirestore(), path), data);
}
```

```
//CREAR PRODUCTO en componente que añade productos
async createProduct(){

  if (!this.user || !this.user.uid) {
    console.error('User is not defined or user UID is missing');
    return;
  }

  let path = `users/${this.user.uid}/productos`

  const loading=await this.utilsSvc.loading();

  await loading.present(); //llamando al loading cuando se inicia esta
funcion

  //Parámetros para subir la imagen y obtener la URL

  let dataUrl = this.form.value.imagen;
  let imagenPath = `${this.user.uid}/${Date.now()}`; //De este modo se
obtiene un path único para cada imagen

  //Obtner la URL de la imagen
  let imagenUrl = await this.firebaseSvc.uploadImage(imagenPath,dataUrl);
  this.form.controls.imagen.setValue(imagenUrl);

  delete this.form.value.id //Se borra el ID porque se realiza en la
función de actualizar pero no para la de agregar
```

```
this.firebaseSvc.addDocument(path,this.form.value). then(async res =>{

    this.utilsSvc.dismissModal({ success: true});

    this.utilsSvc.presentToast({
        message: "Producto creado correctamente",
        duration: 2500,
        color:'success',
        position:'middle',
        icon:'checkmark-circle-outline'
    })

    //Controlando errores
}).catch(error =>
    {console.log(error);

    }

    //SI TODO ESTA BIEN QUITAR LA FUNCION LOADING
).finally(()=>
    {loading.dismiss();
    })
}
```

Técnicas Adicionales

Control de Versiones con Git y GitHub:

Git para el control de versiones, permitiendo la gestión de cambios y la colaboración en equipo. GitHub aloja el repositorio del proyecto, facilitando su desarrollo en varios equipos.

Responsive Design:

Ionic Responsive Grid asegura que la aplicación sea accesible y usable en diferentes tamaños de pantalla y dispositivos.

Temas Oscuro y Luminoso:

Implementación de temas oscuros y luminosos para mejorar la accesibilidad y la experiencia del usuario.

Servicio de Guards en Ionic:

Los guards en Ionic se utilizan para controlar el acceso a determinadas páginas o funcionalidades de la aplicación, según ciertas condiciones, como el estado de autenticación del usuario.

Ejemplo del servicio de guards:

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  firebaseSvc = inject(FirebaseService);
  utilsSvc = inject(UtilsService);

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> |
    Promise<boolean | UrlTree> | boolean | UrlTree{
    let user = localStorage.getItem('user');
    return new Promise((resolve) => {
      this.firebaseSvc.getAuth().onAuthStateChanged((auth)=>{
        if(auth){
          if(user)
            resolve(true);
        }
        else{
          this.firebaseSvc.signOut();
          resolve(false);
        }
      })
    })
  }
}
```

```
});  
  
}  
  
}
```

Y como lo inyecto en el módulo que lo requiere de la aplicación:

```
path: 'home',  
  loadChildren: () => import('./paginas/home/home.module').then( m =>  
m.HomePageModule), canActivate:[AuthGuard]  
},
```

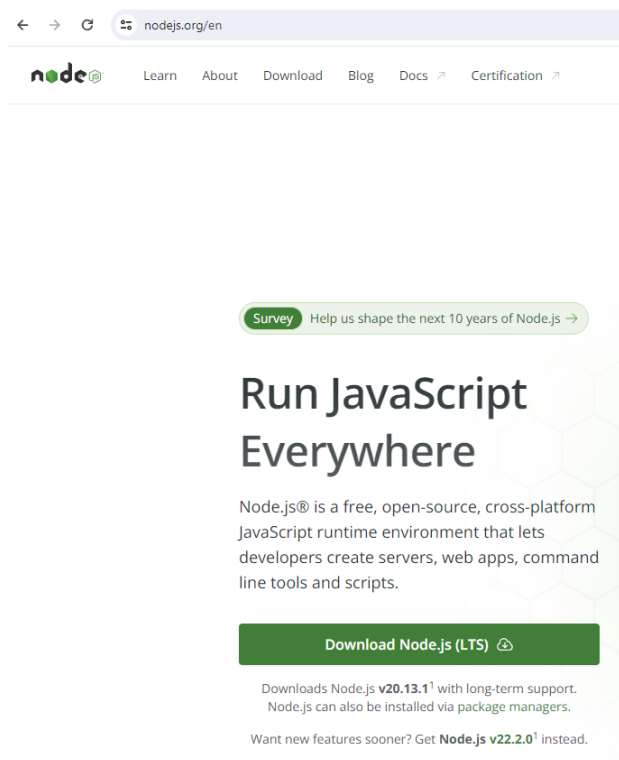
2.3 Despliegue de la aplicación indicando plataforma tecnológica, instalación de la aplicación y puesta en marcha

Para llevar a cabo la instalación del programa en un ordenador es necesario primeramente la descarga de NodeJS, también se recomienda tener instalado el IDE Visual Studio Code por si se desea comprobar los archivos que posee.

Una vez instalado con el proyecto al ser bajado del repositorio con un git clone es recomendable realizar un comando de:

npm install

Esto es por si ha quedado alguna dependencia o componente desactualizado o no se ha instalado correctamente, en caso de que no deje lo correcto es realizarlo desde un terminal abriéndolo como administrador.



Una vez instalado el programa en el equipo, a través del terminal o un IDE que permita introducir comandos lanzar el siguiente comando:

`npm install -g @ionic/cli`

Esto producirá la instalación de Ionic en el equipo. Puede comprobarse lanzando el comando:

`ionic -v` o `ionic --version`

Para comprobar la versión que se ha instalado o escribiendo:

`ionic start`

```
ionic CLI 7.2.0

Usage:
  $ ionic <command> [<args>] [--help] [--verbose] [--quiet] [--no-interactive] [--no-color] [--confirm] [options]

Global Commands:
  completion ..... (experimental) Enables tab-completion for Ionic CLI commands.
  config <subcommand> ..... Manage CLI and project config values (subcommands: get, set, unset)
  info ..... Print project, system, and environment information
  init ..... (beta) Initialize existing projects with Ionic
  live-update <subcommand> ..... (paid) Ionic Live Updates functionality (subcommands: manifest)
  login ..... Log in to Ionic
  logout ..... Log out of Ionic
  signup ..... Create an Ionic account
  ssh <subcommand> ..... (deprecated) Commands for configuring SSH keys (subcommands: add, delete, generate, list, setup, use)
  start ..... Create a new project

Project Commands:
  You are not in a project directory.
```

Una vez que se ha instalado correctamente, hay que posicionarse en la carpeta principal del proyecto a través de la terminal y lanzar el comando:

ionic serve

Esto generará un servidor que se lanzará en local en el navegador predeterminado, normalmente en localhost/8100. Al ser una versión para móvil se recomienda ir al **menú de más herramientas→ Opciones→ Herramientas para desarrolladores** o en el comando de teclado **Control + Mayús + I**.

Lo cual permitirá ver la terminal y la pantalla adaptada a la versión móvil, pudiendo escoger el tipo de dispositivo tanto Tablet como móvil.

Como la aplicación puede ser utilizada tantos en Android como en IOS se especificarán los pasos a seguir en cada caso:

Plataforma Android:

A través de la línea de comandos, una vez se hayan realizado todos los pasos previos:

Instalar el paquete @capacitor/Android con el comando:

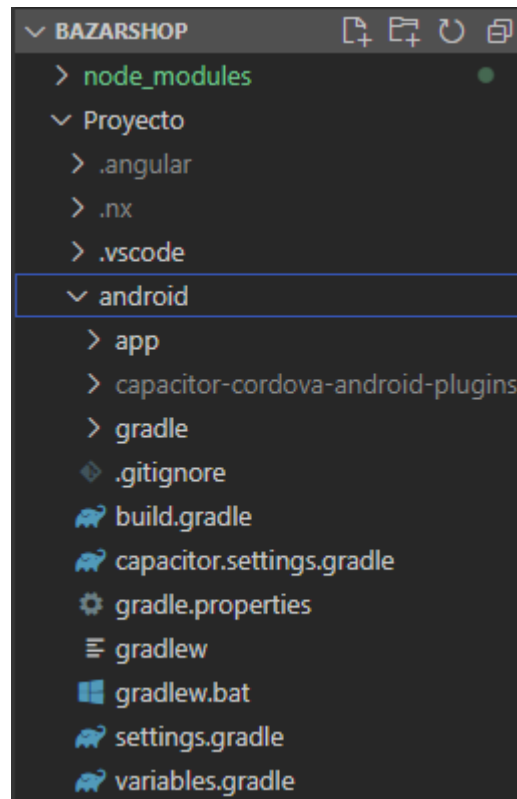
npm install @capacitor/Android

Realizar un **ionic build** en la línea de comandos.

Luego, agregar la plataforma Android con el siguiente comando:

`npx cap add Android`

Al realizar esto se genera la carpeta Android en el código de la aplicación.



Apertura del Proyecto en Android Studio

Para abrir el proyecto en Android Studio, ejecuta:

`npx cap open android`

Una vez que se lanza este comando te redirigirá automáticamente a Android Studio y se verá el código del proyecto. Se podrá ejecutar en un simulador, asegúrate de tener los SDK de Android en el programa. Puede que los últimos den fallo, por lo tanto, es recomendable utilizar una versión de Android anterior a la 14.

Una vez comprobado acudir a build → Generar APK.

Tras un tiempo de espera, el APK de la aplicación será generado y almacenado en el equipo. Acude a la carpeta donde se ha guardado pulsando en locate. Con un

USB, se puede pasar este archivo a tu dispositivo móvil o a través de un Cloud, por ejemplo.

En tu dispositivo:

Habilitar la depuración USB:

Ve a **Configuración > Acerca del teléfono**.

Toca el Número de compilación siete veces para activar las Opciones de desarrollador.

Vuelve a **Configuración > Opciones de desarrollador y habilita Depuración USB**.

Permitir la instalación de aplicaciones de fuentes desconocidas. Una vez realizados estos pasos, la aplicación quedará instalada en tu dispositivo.

Plataforma iOS:

Descargar el programa **Xcode** desde App Store.

Instalar CocoaPods a través del siguiente comando:

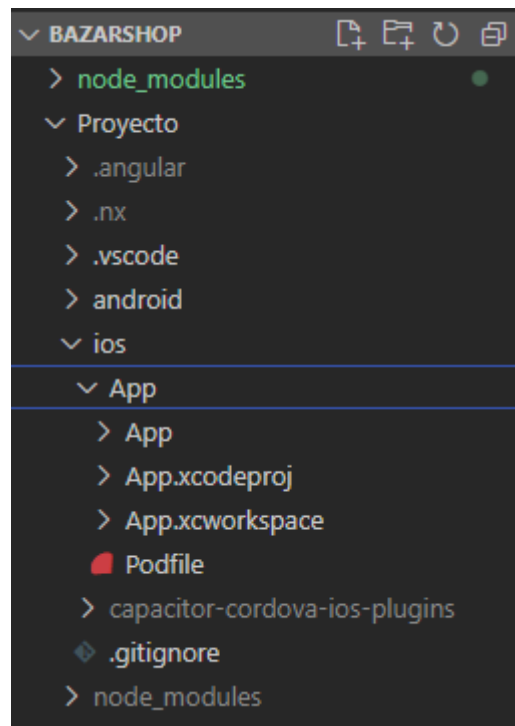
```
$ sudo gem install cocoapods
```

Instalación de **@Capacitor/ios** en la plataforma de iOS:

```
npm install @capacitor/ios
```

```
npx cap add ios
```

Esto genera la carpeta ios en el proyecto:



Después, abrir el proyecto en Xcode utilizando el comando:

`npx cap open ios`

En Xcode se pueden realizar ajustes adicionales si es necesario y se compilar la aplicación para su ejecución en un simulador de iOS o para su instalación en un dispositivo físico.

Para distribuir la aplicación en dispositivos iOS, se necesita un archivo IPA firmado correctamente que puede generarse desde Xcode.

Para descargar el **archivo IPA en dispositivo IOS:**

Cuenta de desarrollador de Apple: Necesitarás una cuenta de desarrollador de Apple si deseas instalar aplicaciones en dispositivos físicos.

Pasos:

En Xcode, ve a **Window > Devices and Simulators**.

En la ventana que se abre, asegúrate de que tu dispositivo iOS esté listado bajo la pestaña Devices. Si no aparece, asegúrate de que está desbloqueado y de confiar en la computadora en el dispositivo.

4. Preparar el archivo IPA

5. Arrastrar y soltar el IPA en Xcode

En la ventana de Devices and Simulators en Xcode, selecciona tu dispositivo en la lista de la izquierda. Arrastra y suelta el archivo IPA en la sección Installed Apps de tu dispositivo.

6. Xcode instalará la aplicación

3 Planificación y presupuesto

En cuanto a la planificación:

Fase 1: Definición y Diseño (2 semana)

Definir requisitos: Definición de los objetivos y características principales de la aplicación.

Diseño de la Interfaz de Usuario

Arquitectura del Proyecto: Elección de las tecnologías a utilizar y diseño de la estructura básica de la aplicación.

Documentación: Creación de una documentación mínima para guiar el desarrollo.

Fase 2: Desarrollo (3 semanas)

Implementación de la Interfaz: Desarrollo de los componentes de la interfaz de usuario según el diseño establecido.

Desarrollo de Funcionalidades: Implementación de las características principales de la aplicación, como la gestión de productos, autenticación de usuarios, etc.

Integración con Firebase: Configuración y conexión de la aplicación con Firebase para almacenamiento de datos y autenticación de usuarios.

Pruebas Continuas: Realización de pruebas durante el desarrollo para detectar y corregir errores.

Fase 3: Pruebas y Redacción de documentación (1 semana)

Pruebas Funcionales: Realización de pruebas exhaustivas para asegurar el correcto funcionamiento de la aplicación.

Redacción de documentación y manual de usuario.

Fase 4: Lanzamiento y Mantenimiento (indefinido)

Despliegue: Publicación de la aplicación en las tiendas de aplicaciones (si es necesario).

Mantenimiento Continuo: Actualizaciones periódicas para corregir errores, introducir nuevas funcionalidades y mejorar la experiencia del usuario.

En cuanto al presupuesto:

Teniendo en cuenta que lo ha desarrollado una persona sin experiencia y se han invertido aproximadamente 20 horas por semana, una duración de 6 semanas y estimando que un desarrollador junior puede cobrar unos 11 euros/hora aproximadamente.

Recursos humanos:

Desarrollador: 20 horas/semana * 6 semanas = 120 horas

120 horas * 11 euros/hora = 1320 euros.

Costos Adicionales:

Servicios de Firebase y mantenimiento: Variable dependiendo del uso, supongamos 200 euros para el período de desarrollo y futuro.

Total estimado del proyecto:

1520 euros.

Todo ello, contando con que el proyecto lo ha realizado tan sólo una persona y con funcionalidades iniciales que posteriormente podrán ampliarse bastante.

Por lo que si se realiza por un equipo de desarrolladores habría que pagarles a todos dependiendo de su rango y el tiempo de trabajo aumentaría. Por lo que el presupuesto podría verse aumentando considerablemente.

4 Documentación Técnica: análisis, diseño, implementación y pruebas.

4.1 Especificación de requisitos

Requisitos Funcionales:

1. Autenticación de Usuarios:

- Los usuarios deben poder registrarse e iniciar sesión en la aplicación.
- Para hacer el login se requiere un email y contraseña.
- Para registrarse se requiere un nombre, un email y una contraseña.
- El nombre debe tener mínimo dos caracteres, el email debe tener un formato adecuado y la contraseña debe tener mínimo 6 caracteres.
- Al escribir la contraseña debe existir la opción de visualizar sus caracteres y ocultarlos.
- Se debe proporcionar un mecanismo para recuperar la contraseña en caso de olvido.
- Los usuarios deben poder cerrar sesión de manera segura.
- Los usuarios podrán actualizar nombre y borrar la cuenta desde su perfil.
- Los usuarios deben ser almacenados en Firebase para su persistencia.

2. Gestión de Productos:

- Los usuarios autenticados deben poder agregar, ver, editar y eliminar productos.
- Cada producto debe tener un nombre, cantidad, precio y una imagen opcional.
- Los productos deben ser almacenados en Firebase para su persistencia.

3. Visualización de Inventario:

- Los usuarios autenticados deben poder ver una lista completa de todos los productos almacenados, ordenados por cantidad de unidades.

4. Cálculo de Ganancias Potenciales:

- La aplicación debe calcular automáticamente las ganancias potenciales de cada producto y el total que puede generar en base a la cantidad y precio de cada producto y mostrarlo en pantalla.

5. Interfaz de Usuario Intuitiva:

- La interfaz de usuario debe ser intuitiva y fácil de usar, con navegación clara y elementos de diseño coherentes.

Requisitos No Funcionales:

1. Rendimiento:

- La aplicación debe ser rápida y receptiva, con tiempos de carga mínimos.

2. Seguridad:

- Se deben implementar medidas de seguridad adecuadas para proteger los datos de los usuarios (guards).

3. Compatibilidad:

- La aplicación debe ser compatible con una amplia gama de dispositivos y navegadores web, incluidos dispositivos móviles con diferentes resoluciones de pantalla y sistemas operativos.

4. Escalabilidad:

- La arquitectura de la aplicación debe ser escalable para manejar un crecimiento futuro en la base de usuarios y la cantidad de datos almacenados.

5. Disponibilidad:

- La aplicación debe estar disponible y accesible para los usuarios en todo momento.

6. Acceso a la Cámara del dispositivo:

- Permisos para acceder a la cámara para que los usuarios puedan tomar fotos de los productos y de perfil.

4.2 Análisis del sistema

Este análisis proporciona una visión general de los elementos clave del sistema, excluyendo las funcionalidades detalladas previamente.

Tecnologías Utilizadas

Frontend:

- Ionic Framework: Utilizado para la interfaz de usuario con componentes reutilizables y estilizados. Algunos componentes utilizados incluyen IonHeader, IonContent, IonList, IonItem, y IonButton.
- Estructura: La estructura de la aplicación se basa en lenguaje HTML para el marcado, TypeScript para la lógica y CSS para los estilos.
- Angular: Utilizado para manejar la lógica de la aplicación y la manipulación de manera eficiente.

Backend:

- Firebase: Proporciona la base de datos en tiempo real y servicios de autenticación. Permite la sincronización instantánea de datos y gestión de usuarios.
- Firebase Authentication: Gestiona el registro, inicio de sesión y recuperación de contraseñas de los usuarios.
- Firebase Database: Proporciona una base de datos NoSQL en tiempo real para almacenar los datos de los productos y usuarios.

Acceso a Hardware:

- Capacitor: Utilizado para acceder a funcionalidades nativas del dispositivo.
- @capacitor/camera: Esta API permite capturar imágenes usando la cámara del dispositivo para ser usadas como imágenes de productos y perfil.

4.3 Diseño del sistema:

4.3.1 Diseño de la Base de Datos

Como se ha comentado previamente la base de datos se ha creado en Firebase que es una base de datos no relacional o NoSQL y que sirve para almacenar los datos en colecciones.

En el caso de esta aplicación, consta de dos colecciones, una principal que almacena los usuarios con sus respectivos datos y otra secundaria que almacena los productos asociados a ese usuario.

Además, tiene el servicio de autenticación con cada usuario, el servicio de envío de contraseña en casa de no recordarla y las reglas de almacenamiento.

Los campos que se almacenan de los usuarios son:

Id, nombre, email, contraseña y, de forma opcional una imagen.

BazarShop-app ▾

Authentication

Usuarios Método de acceso Plantillas Uso Configuración Extensions

Agregar usuario

Identificador	Proveedores	Fecha de creación ↓	Fecha de acceso	UID de usuario
nuria@gmail.com	✉	21 may 2024	21 may 2024	KTC3W1VSQzgAwn6F3TsMnC...
paquito@hotmail.com	✉	21 may 2024	21 may 2024	TgJjWafxmWcDwV6yo5J2G...
paco@hotmail.com	✉	21 may 2024	21 may 2024	vvq11dmqzuNejfNeYUXq2DIJ...
pepe@gmail.com	✉	21 may 2024	21 may 2024	mpHrAbCFB7eNNYH4R9NcYa...
roberto.toquero19@gm...	✉	20 may 2024	22 may 2024	yVeVtghNXDMFW1w4WIF9bf...
juan@gmail.com	✉	17 may 2024	21 may 2024	Sgjo5zJutDdRU3NA2sYSI89pj...
maria@gmail.com	✉	16 may 2024	21 may 2024	38Sed8y41RYev77wGr28cfmk...

Filas por página: 50 1 - 7 of 7

Imagen 5: Autenticación de los usuarios

🏠 > users > mpHrAbCFB7e...

Más funciones en Google Cloud ▾

(default)	users	mpHrAbCFB7eNNYH4R9NcYaY2zI72
<p>+ Iniciar colección</p> <p>users ></p>	<p>+ Agregar documento</p> <p><code>{id}</code></p> <p><code>38Sed8y41RYev77wGr28cfmkE12</code></p> <p><code>KTC3W1VSQzgAwn6F3TsMnCrLS0s1</code></p> <p><code>Sgjo5zJutDdRU3NA2sYSI89pjzt2</code></p> <p><code>TgJjWafxmWcDwV6yo5J2GVf6122</code></p> <p><code>mpHrAbCFB7eNNYH4R9NcYaY2zI72</code></p> <p><code>yVeVtghNXDMFW1w4WIF9bfoeAVK2</code></p>	<p>+ Iniciar colección</p> <p>+ Agregar campo</p> <p>email: "pepe@gmail.com"</p> <p>nombre: "pepe"</p> <p>password: "\$2a\$10\$vuuf99ZWddJ2.2ZC8MDUA.tWj8JfRyJNvOjqrnOQFmdX2jPAF/...</p> <p>uid: "mpHrAbCFB7eNNYH4R9NcYaY2zI72"</p>

Imagen 6: Almacenamiento de un usuario sin productos subidos a la app

(default)	users	38Sed8y41RYev77wGr28cfmkbEI2
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
users >	$\{id\}$ 38Sed8y41RYev77wGr28cfmkbEI2 > KTC3W1VSQzgAwn6F3TsMnCrLS0s1 Sgjo5zJutDdRU3NA2sYSI89pjzt2 TgJjWAFxmWcDwV6yo5J2GVf6122 mpHrAbCFB7eNNYH4R9NcYaY2zI72 yVeVtghNXDMFW1w4WIF9bfoeAVK2	productos + Agregar campo displayName: "Maria" email: "maria@gmail.com" imagen: "https://firebasestorage.googleapis.com/v0/b/bazarshop-app.appspot.com/o/38Sed8y41RYev77wGr28cfmkbEI2%2Fperfil?alt=media&token=21d31aa4-dd36-4485-bca9-219eecd6c80e" nombre: "Maria" uid: "38Sed8y41RYev77wGr28cfmkbEI2"

Imagen 7: Almacenamiento de usuario con imagen y productos.

Por lo tanto, si el usuario ha subido los productos, de forma automatizada, se le asocia la subcolección productos.

A la colección de productos se suben los datos:

Id, nombre, unidades, precio e imagen (de forma opcional).

+ Iniciar colección	+ Agregar documento	+ Iniciar colección
productos >	spveZ73wmPZzVPQuEEwn >	+ Agregar campo
+ Agregar campo		id: "spveZ73wmPZzVPQuEEwn" imagen: "https://firebasestorage.googleapis.com/v0/b/bazarshop-app.appspot.com/o/38Sed8y41RYev77wGr28cfmkbEI2%2F171588031487?alt=media&token=d6cceb79-3b8c-4891-be8e-28a40a580902" nombre: "Tijeras" precio: 25 unidades: 2
displayName: "Maria" email: "maria@gmail.com" imagen: "https://firebasestorage.googleapis.com/v0/b/bazarshop-app.appspot.com/o/38Sed8y41RYev77wGr28cfmkbEI2%2F171588031487?alt=media&token=21d31aa4-dd36-4485-bca9-219eecd6c80e" nombre: "Maria" uid: "38Sed8y41RYev77wGr28cfmkbEI2"		

Imagen 8: Colección de productos con su usuario asociado

En cuanto a las reglas, se almacenan en el propio firebase y son las siguientes:


```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    function isOwner(userId) {
      return request.auth.uid == userId;
    }
    //Regla para la colección de usuarios
    match /users/{userId}{
      allow read, write: if isOwner(userId);
    }

    // Regla para la colección de productos dentro de cada usuario
    match /users/{userId}/productos/{productoId} {
      allow read, write: if isOwner(userId);
    }
  }
}
```

Imagen 9: Reglas de la base de datos

Estas reglas aseguran que los datos en las colecciones "users" y "productos" dentro de cada usuario solo puedan ser leídos o escritos por el usuario autenticado que es el propietario de los datos. Esto protege la privacidad y seguridad de los datos de cada usuario en la aplicación BazarShop.

En cuanto a cómo se conecta al código de la aplicación, se instala a través de node y se adjunta el siguiente código:

Configuración del SDK

☒ npm ☐ CDN ☐ Config

Si ya usas [npm](#) y un agrupador de módulos como [Webpack](#) o [Rollup](#), puedes ejecutar el siguiente comando para instalar la versión más reciente del SDK ([más información](#)):

```
$ npm install firebase
```



Luego, inicializa Firebase y comienza a usar los SDK de los productos que quieres utilizar.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyDIpe-0bZpzV_BbchD9-p8DwsdHjQUkGWA",
  authDomain: "bazarshop-app.firebaseio.com",
  projectId: "bazarshop-app",
  storageBucket: "bazarshop-app.appspot.com",
  messagingSenderId: "807664708312",
  appId: "1:807664708312:web:ccf10044eb12ca4bcc85ad",
  measurementId: "G-HL3TVKK7W7"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```



Imagen 10: SDK de Firebase

Posteriormente en la estructura del código se crea un `firebase.service` desde donde se importarán todos los componentes necesarios para interactuar con los datos y se declararán los métodos requeridos por las funcionalidades de la aplicación.

Por último, el servicio de recuperación de contraseña también se configura desde firebase, al igual que otros varios como el de verificación de correo electrónico, por ejemplo.

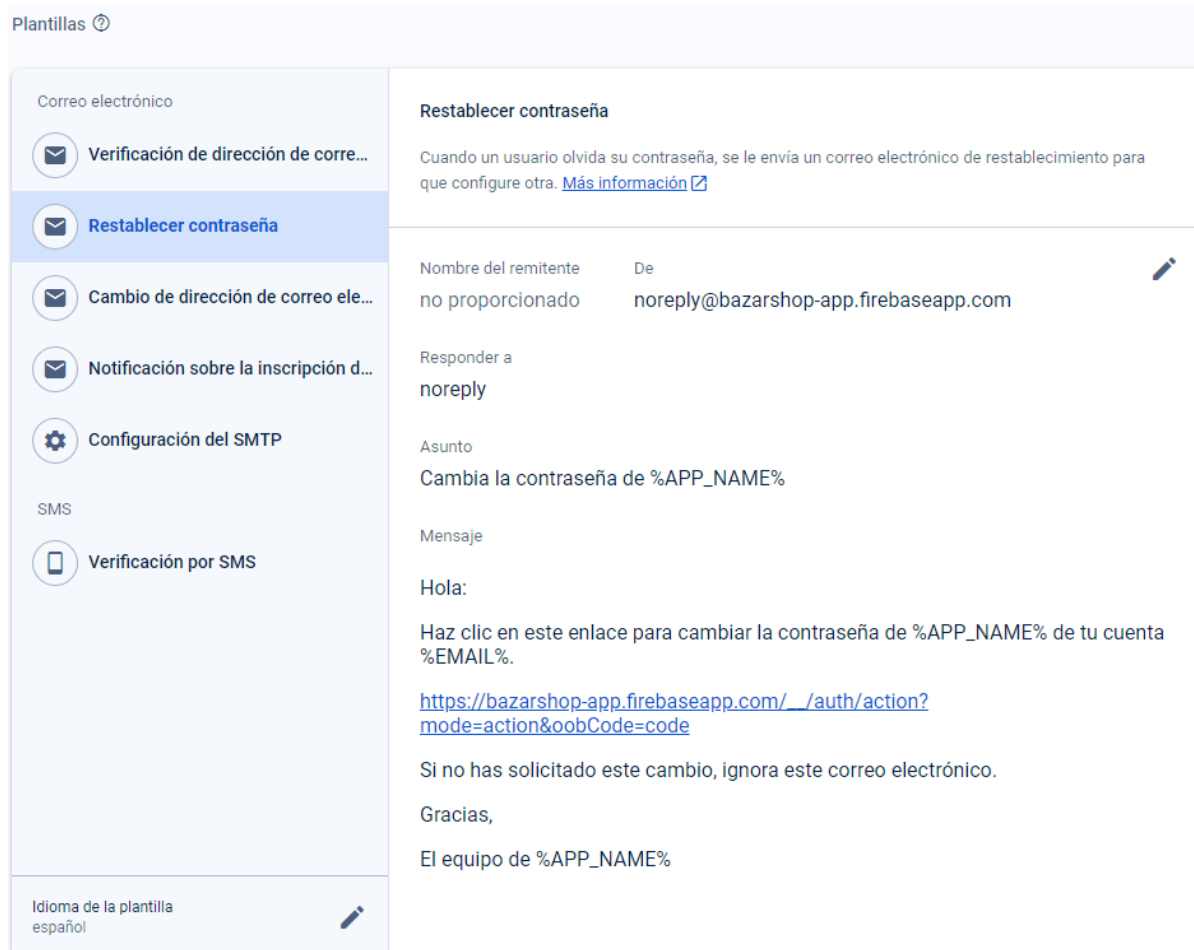


Imagen 11: Configuración del correo de recuperación de contraseña

4.3.2 Diseño de la Interfaz de usuario.

La aplicación es interactiva y consta de varios elementos que ofrece Ionic para hacer más sencilla e intuitiva la navegación.

Destacan:

Swiper-slide: Instalado desde una biblioteca externa a Ionic, sirve para crear pantallas deslizantes dentro de una misma página. Como mínimo deben crearse tres cada vez que se implementa y cada una de estas pantallas puede llevar los

elementos que se deseen como si fuera una página independiente. Al ser slide, se deslizan en bucle en de forma infinita de izquierda a derecha y de derecha a izquierda. Esto está implementado en el menú de inicio.

```
Proyecto > src > app > paginas > inicio > inicio.page.html > ion-content > swiper-container
1  <app-header title="Inicio"></app-header>
2
3  <ion-content>
4      <swiper-container (slidechange)="swiperSlideChanged($event)" [loop]
5          <!-- Pantalla 1 -->
6          <swiper-slide>
7              <div class="bg_animate">
8                  <app-logo></app-logo>
9                  <br><br>
10                 <div class="contenedor">
11                     <h1>Organiza tus productos</h1>
12                 </div>
13                 <section class="banner">
14                     <section class="banner_title">
15                         <h2>Crea tu <br> inventario</h2>
16                     </section>
17                     <div class="banner_img">
18                         
19                     </div>
20                 </section>
21                 <div class="burbujas">
22                     <div class="burbuja"></div>
23                     <div class="burbuja"></div>
24                     <div class="burbuja"></div>
25                     <div class="burbuja"></div>
26                     <div class="burbuja"></div>
27                     <div class="burbuja"></div>
28                     <div class="burbuja"></div>
29                     <div class="burbuja"></div>
30                     <div class="burbuja"></div>
31                     <div class="burbuja"></div>
32                 </div>
33             </div>
34         </swiper-slide>
35         <!-- Pantalla 2 -->
36         <swiper-slide>
37             <div class="contenedor2">
```

Imagen 12: Implementación de Swipper Slide en el código

Ion-icon: Se trata de los diferentes iconos que ofrece Ionic de forma automática, los cuales llamas de forma automática y puedes introducir el estilo que desees. Todos ellos están recogidos en:

<https://ionic.io/ionicons>

Ion-header: Configurado como un contenedor que va a crear el header de la aplicación, en mi caso lo he configurado una sola vez, en elementos compartidos (shared), para posteriormente llamarlo desde el resto de las pantallas.

```
<!-- Aquí se almacenan los componentes compartidos del header que son llamados desde el respectivo ht
<ion-header class="ion-no-border">
  <ion-toolbar color="tertiary">

    <ion-buttons slot="start">
      <ion-back-button *ngIf="backButton" [defaultHref]="backButton"></ion-back-button>
    </ion-buttons>

    <!-- Botón que solo aparece en la pantalla de añadir producto para cerrarla. Cuando es Modal -->
    <ion-buttons slot="start">
      <ion-button (click)="cerrarModal()" *ngIf="isModal">
        <ion-icon name="close-circle-outline"></ion-icon>
      </ion-button>

      <ion-menu-button *ngIf="showMenu" menu="menu-content"></ion-menu-button>
    </ion-buttons>

    <ion-title class="text-center">
      <strong>{{title}}</strong>
      <div class="admin">B A Z A R S H O P</div>
    </ion-title>
  </ion-toolbar>
</ion-header>
```

Imagen 13: Implementación del header

Ion-content: Este componente representa el área principal de contenido en una página de Ionic. Se utiliza para mostrar el contenido de la aplicación, como texto, imágenes, formularios, etc. Puedes aplicar estilos CSS para personalizar su apariencia y comportamiento.

Ion-button: Es un componente que representa un botón en Ionic. Se pueden aplicar atributos HTML como (click) para agregar funciones de clic, o incluso usar Angular para manejar eventos de clic.

Ion-input: Este componente proporciona un campo de entrada de texto en Ionic. También permite aplicar atributos HTML estándar como placeholder, disabled, etc., para personalizar su comportamiento y apariencia.

Formularios: Ionic ofrece soporte completo para la creación y validación de formularios. Permite utilizar los elementos ion-input, ion-select, ion-checkbox, etc., junto con HTML y CSS para diseñar y validar formularios.

Toast: Ion-toast es un componente utilizado para mostrar mensajes temporales o notificaciones al usuario. Puedes utilizar métodos de Angular o Javascript para controlar cuándo y cómo se muestra el toast en la aplicación.

Ion-card: Es un contenedor que muestra contenido en forma de tarjeta con un diseño visual agradable. También se aplican estilos CSS para personalizar su apariencia y contenido.

Ion-item: Representa un elemento de lista en Ionic, que puede contener texto, imágenes u otros elementos.

Ion-item-sliding: Proporciona una forma de deslizar los elementos de la lista para revelar acciones adicionales. Se puede aplicar eventos HTML o Angular para controlar las acciones que se desencadenan cuando el usuario desliza un elemento de la lista.

Ion-avatar: Es un componente utilizado para mostrar una imagen circular, típicamente asociada con un usuario o un elemento específico. Se puede personalizar con CSS.

Ion-list: Proporciona una lista de elementos en Ionic, que puede contener elementos ion-item, ion-avatar, etc. Puedes aplicar estilos CSS para personalizar el diseño y el aspecto de la lista y sus elementos.

Ion-label: Representa una etiqueta de texto en Ionic, que se utiliza para describir un campo de entrada u otro elemento. También es totalmente personalizable.

lon-footer: Es un componente que se coloca al final de una página en Ionic. Puedes aplicar estilos CSS para personalizar su apariencia y contenido, al igual que con otros elementos de Ionic.

lon-toggle: Proporciona un interruptor de palanca para cambiar entre dos estados, generalmente "encendido" y "apagado". Se aplican eventos HTML o Angular para controlar las acciones que se desencadenan cuando el usuario cambia el estado del interruptor.

lon-skeleton-text es un componente de Ionic utilizado para mostrar un esqueleto de texto mientras se carga el contenido real en una aplicación. Este componente es especialmente útil para mejorar la experiencia del usuario al proporcionar retroalimentación visual sobre la carga de datos. Se utiliza comúnmente en situaciones donde se espera una demora en la carga de contenido.

```
<!-- Productos -->
<ion-list *ngIf="!loading">
  <ion-item-sliding class="product-item" *ngFor="let p of productos">
    <ion-item>
      <ion-avatar slot="start">
        <img [src]="p.imagen" />
      </ion-avatar>
      <ion-label>
        <div class="name">
          {{p.nombre}}
        </div>
        <div class="data">
          <b>Precio: {{p.precio}} €</b>
          <br>
          <b>Unidades: {{ p.unidades }} </b>
        </div>
        <ion-chip color="terciary" mode="ios" outline="true">
          <ion-label> Ganancia Potencial: {{ p.precio * p.unidades }} €</ion-label>
        </ion-chip>
      </ion-label>
    </ion-item>
    <!-- OPCIONES -->
    <ion-item-options side="end">
      <!-- editar -->
      <ion-item-option (click)="addUpdateProduct(p)">
        <ion-icon class="option-btn" name="create-outline"></ion-icon>
      </ion-item-option>
      <!-- Eliminar -->
      <ion-item-option (click)="confirmDeleteProduct(p)" color="danger">
        <ion-icon class="option-btn" name="trash-outline"></ion-icon>
      </ion-item-option>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

Imagen 14: Introduciendo varios elementos comentados

4.3.3 Diseño de la Aplicación.

A lo largo de su interfaz, se despliegan diversas pantallas y funcionalidades diseñadas para facilitar la navegación y la gestión de inventario.

Tras la pantalla de inicio, la aplicación presenta un menú que se compone de varias opciones, incluyendo la posibilidad de iniciar sesión, registrarse o recuperar contraseña. Este menú inicial ofrece una entrada flexible a la aplicación, permitiendo a los usuarios acceder a las diferentes funciones según sus necesidades.

Una vez dentro de la aplicación, los usuarios pueden navegar a través de varias pantallas, como la pantalla principal donde se agregan y muestran los productos disponibles, representados en tarjetas deslizantes. Estas tarjetas permiten una visualización intuitiva de los productos, con la capacidad de actualizar o eliminar productos con un simple deslizamiento hacia la izquierda.

Además, la aplicación presenta un ion-card en la parte superior que proporciona información relevante, como el número total de productos y las ganancias generadas. También se incluye un botón para añadir nuevos productos de forma rápida y sencilla.

La interfaz de usuario se complementa con un menú lateral accesible mediante un deslizamiento hacia la izquierda, que ofrece opciones adicionales, como el acceso al perfil del usuario, ver la foto de perfil y el cierre de sesión.

En el perfil del usuario, se proporciona una vista detallada de la información personal, incluyendo el nombre, correo electrónico y foto de perfil. Desde aquí, los usuarios pueden actualizar su información, cambiar la foto de perfil o incluso eliminar su cuenta.

En resumen, la aplicación BazarShop ofrece una experiencia fluida y completa, con una interfaz de usuario intuitiva y funcionalidades diseñadas para satisfacer las necesidades de gestión de inventario de los usuarios.

4.4 Implementación

4.4.1 Entorno de desarrollo.

El entorno de desarrollo empleado ha sido Visual Studio Code ya que aporta grandes funcionalidades y complementos para el desarrollo en Ionic. Visual Studio Code es la elección más popular entre los desarrolladores de Ionic por varias razones. En primer lugar, su interfaz limpia y su diseño intuitivo hacen que sea fácil de usar, lo que es fundamental para mantener la productividad. Además, cuenta con una amplia gama de funcionalidades integradas y complementos que optimizan el flujo de trabajo.

Una de las características destacadas que he utilizado de Visual Studio Code es su terminal integrada, que permite ejecutar comandos directamente desde el editor. Esto es especialmente útil al trabajar con Ionic, ya que facilita la instalación de dependencias, la gestión de API y la interacción con la base de datos sin tener que cambiar de ventana o abrir una terminal por separado.

En cuanto al apartado de extensiones, he instalado las extensiones específicas para Ionic y Angular, como Angular Snippets e Ionic Snippets, son herramientas imprescindibles para simplificar la escritura de código. Estas extensiones proporcionan sugerencias de código, completado automático y correcciones rápidas, lo que agiliza el proceso de desarrollo y reduce la posibilidad de errores.

Además, la integración con Git ofrece una experiencia de desarrollo aún más eficiente al proporcionar sugerencias de código generadas automáticamente, basadas en el contexto del proyecto.

Para ver los cambios que se iban realizando he utilizado el navegador Google Chrome que se crea al lanzar el servidor de Ionic en vez de un emulador, por una cuestión de potencia en la RAM de mi ordenador ya que la creación de un emulador ralentiza mucho el proceso.

4.4.2 Estructura del código.

BazarShop se compone de una carpeta Proyecto y de un archivo ReadME.

En la carpeta proyecto se encuentran varias carpetas como las de Android e iOS mostradas anteriormente, la carpeta src que es donde se aloja el código y varios archivos de configuración de angular, ionic, capacitor en formato Json que se generan automáticamente con la creación del proyecto.

La carpeta src

Se encuentra las carpetas, app, assets, environments, theme y otros archivos relevantes como el global.css donde he pegado algunos elementos css que son generales para ciertos componentes de la aplicación. Pudiendo ser llamados desde cualquier parte de la misma.

La carpeta `environments` contiene la conexión con la base de datos Firebase, pegando el código facilitado en ella. Recordando que para instalarla correctamente hay que **quitar el modo estricto configurándolo en `strict: false` en el archivo `tsconfig.json`** en la raíz del proyecto para que no genere problemas con Angular.

La carpeta `assets` tiene los archivos físicos como las imágenes o los logos. En este caso solo he utilizado dos imágenes para la pantalla de inicio ya que el resto de las imágenes subidas por el usuario se almacenan en la base de datos.

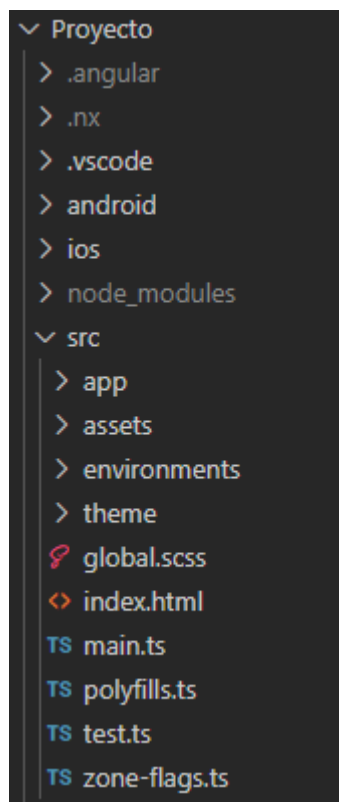


Imagen 15: carpeta `src`

La carpeta `app`

Es en esta carpeta donde se encuentra toda la lógica, estructura y estilos de la aplicación.

Cuenta con varias carpetas que son:

- Guards: Dos archivos donde se configuran los guards de Ionic para evitar que el usuario entre al menú o salga a inicio directamente si pone la ruta en el navegador.

- **Models:** Es donde se crean los objetos User y Producto que representan a las dos colecciones de la base de datos y que serán necesarios en muchas partes del código por lo que serán importadas posteriormente en la parte lógica de cada una de las páginas.
- **Paginas:** Donde se han creado tres páginas que almacenan cada una de las partes de la aplicación. Una para el inicio, otra para el login que a su vez contiene la pantalla de registro y la de recuperar contraseña. Y por último el home, que a su vez contiene la pantalla principal y el perfil del usuario.
- **Services:** Aquí se almacena el *util.service* con métodos importantes que serán utilizados en varias partes del código y el *firebase.service* que cuenta con los métodos relacionados con la interacción con la base de datos. Ambos serán inyectados en casi todas las páginas del código para utilizarlos.
- **Shared:** Se trata de una carpeta que contiene elementos que serán utilizados en diversas partes del código lo cual aligera las cosas porque es en ellos donde se declara todo, incluidos métodos, lógica o estructuras y estilos para después ser inyectados donde se les necesite.

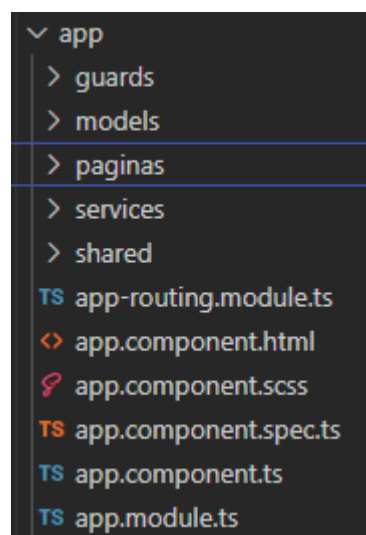


Imagen 16: carpeta app

Cuando se genera una página en Ionic empleando el comando:

ionic generate page nombre_de_la_pagina

Automáticamente se crea una carpeta con varios archivos, de los cuales destacan:

Routing.module: que sirve para establecer todas las rutas con las que estará enlazada esa página con otras páginas.

Module: Donde será necesario incluir los módulos de otras carpetas cuya lógica se desea implementar. Aquí se llama a las lógicas, estilos y estructuras generadas en la carpeta shared.

El html: Donde se escribe la parte del código html para dar la estructura y donde se declaran los elementos ionic mencionados anteriormente.

El css: Donde se escriben los estilos que se quiere dar a los elementos de la página.

El TypeScript: Es donde reside la lógica, los métodos, las inyecciones, importaciones y objetos requeridos para la página.

Por lo tanto, dentro de cada una de las páginas creadas para la aplicación existen estos archivos diferenciados que actúan de forma individual para cada página, pudiendo generar elementos comunes, lo cual facilita la redacción del código evitando repeticiones.

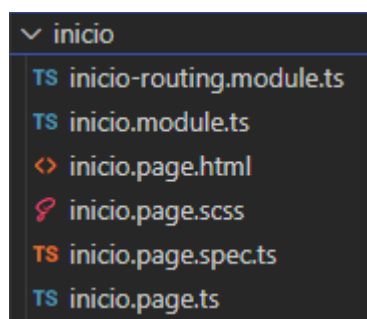


Imagen 17: Componentes de la carpeta inicio

```
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3
4 import { RegistroPage } from './registro.page';
5
6 const routes: Routes = [
7   {
8     path: '',
9     component: RegistroPage
10  }
11 ];
12
13 @NgModule({
14   imports: [RouterModule.forChild(routes)],
15   exports: [RouterModule],
16 })
17 export class RegistroPageRoutingModule {}
```

Imagen 18: Routing module de registro

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormsModule } from '@angular/forms';
4
5 import { IonicModule } from '@ionic/angular';
6
7 import { PrincipalPageRoutingModule } from './principal-routing.module';
8
9 import { PrincipalPage } from './principal.page';
10 import { SharedModule } from 'src/app/shared/shared.module';
11
12 @NgModule({
13   imports: [
14     CommonModule,
15     FormsModule,
16     IonicModule,
17     PrincipalPageRoutingModule,
18     SharedModule
19   ],
20   declarations: [PrincipalPage]
21 })
22 export class PrincipalPageModule {}
```

Imagen 19: Module de principal.page

En cuanto a los TypeScript constan de varias partes, el principio es donde se realizan todas las importaciones necesarias, luego cuando se da inicio al componente es

donde se realizan todas las inyecciones provenientes de otras carpetas que se requieren para ese código en particular y se declaran las variables necesarias o los formularios que se van a utilizar.

A continuación, el método `onInit()` que es el que configura qué elementos necesita tener la página al ser iniciada.

En algunos el método `submit()` que es para enviar el formulario y posteriormente el resto de métodos.

```
1 import { Component, OnInit, inject } from '@angular/core';
2 import { Router } from '@angular/router';
3 import { User } from 'src/app/models/user.model';
4 import { FirebaseService } from 'src/app/services/firebase.service';
5 import { UtilsService } from 'src/app/services/utils.service';
6
7 @Component({
8   selector: 'app-home',
9   templateUrl: './home.page.html',
10  styleUrls: ['./home.page.scss'],
11 })
12 export class HomePage implements OnInit {
13
14   firebaseSvc = inject(FirebaseService);
15   utilsSvc = inject(UtilsService);
16
17   usuario = {} as User;
18
19   paginas = [
20     {titulo: 'Home', url: '/home/principal', icon:'home-outline'},
21     {titulo: 'Perfil', url: '/home/perfil', icon:'person-outline'}
22   ]
23
24   router= inject(Router);
25   currentPath: string = '';
26
27   ngOnInit() {
28     this.router.events.subscribe((event: any) =>{
29       if(event?.url) this.currentPath = event.url;
30     })
31   }
32
33   user(): User{
34     return this.utilsSvc.getFromLocal('user');
35   }
36
37   //Cerrar Sesion
38
39   signOut(){
40     this.firebaseSvc.signOut();
41   }
42
43   ionViewWillEnter() {
44     this.getCurrentUser();
45   }
46
47   getCurrentUser() {
48     return this.utilsSvc.getFromLocal('user');
49   }
50
51 }
52
```

Imagen 20: TypeScript del home.

Por su parte las páginas html y css cuentan con los elementos propios de estos lenguajes, pero con elementos de ionic, en vez de los típicos componentes del html, aunque también puede utilizarse como los *div*, *p*, *image*, *a*, etc.

```
<ion-menu contentId="menu-content" menuId="menu-content" side="start">
  <app-header title="Menú"></app-header>

  <ion-content class="ion-text-center">

    <ion-avatar>
      <img *ngIf="user()?.imagen" [src]="user()?.imagen" />
      <ion-icon *ngIf="!user()?.imagen" class="empty-icon" name="person-circle-outline"></ion-icon>
    </ion-avatar>
    <h4>{{getCurrentUser().nombre}}</h4>
    <ion-menu-toggle auto-hide="false" *ngFor="let p of paginas">
      <ion-item [ngClass]="{'active': currentPath == p.url}" [routerLink]="p.url" routerDirection="root" detail>
        <ion-icon slot="start" [name]="p.icon"></ion-icon>
        <ion-label>{{p.titulo}}</ion-label>
      </ion-item>
    </ion-menu-toggle>
  </ion-content>

  <ion-footer class="ion-no-border safe-p-bottom">
    <ion-menu-toggle auto-hide="false">
      <ion-item (click)="signOut()" lines="none">
        <ion-icon slot="start" name="log-out-outline"></ion-icon>
        <ion-label>Cerrar sesión</ion-label>
      </ion-item>
    </ion-menu-toggle>
  </ion-footer>
</ion-menu>

<ion-router-outlet id="menu-content" main></ion-router-outlet>
```

Imagen 21: HTML del home

```
.container{
  width: 75%;
  margin: 50px auto;
}

.caja-grande{
  margin: auto auto 20px auto;
  background: linear-gradient(0deg, #FFFFFF, #FFFFFF);
  box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
  border-radius: 5px;
  width: 170px;
  padding: .75rem;
}

ion-item ion-label:nth-child(1){
  margin-left: 8px;
}

.letra{
  margin: 0;
  color: #f59e09;
}

.letra-peq{
  font-size: 14px;
  margin-bottom: 20px;
  white-space: nowrap;
}
```

Imagen 22: CSS del login

4.4.3 Cuestiones de diseño e implementación reseñables.

En primer lugar, destaco la **integración de la API de la cámara** la cual requiere de instalar el capacitor:

```
PS C:\Users\Usuario\Desktop\BazarShop\proyecto> npm install @capacitor/camera

added 1 package, and audited 1349 packages in 8s

206 packages are looking for funding
run `npm fund` for details
```

Link: <https://capacitorjs.com/docs/apis/camera>

También es necesario instalar PWA elements:

Link: <https://capacitorjs.com/docs/web/pwa-elements>

```
PS C:\Users\Usuario\Desktop\BazarShop\proyecto> npm install @ionic/pwa-elements
added 1 package, and audited 1350 packages in 6s

206 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Para Angular hacer la siguiente integración:

Angular

main.ts :

```
import { defineCustomElements } from '@ionic/pwa-elements/loader';
// Call the element loader before the bootstrapModule/bootstrapApplication call
defineCustomElements(window);
if (environment.production) {
  enableProdMode();
}
```

En *utils.service* que es donde se ubica el método que será llamado posteriormente desde otras partes del código:

```
import { Camera, CameraResultType, CameraSource } from '@capacitor/camera';
```

Con su función:

```
//FUNCION PARA RECOGER IMAGENES GRACIAS A LA API DE LA CAMARA
async tomarFoto(promptLabelHeader: string){
  return await Camera.getPhoto({
    quality: 90,
    allowEditing: true,
    resultType: CameraResultType.DataUrl,
    source: CameraSource.Prompt, //Permite al usuario elegir si quiere tomar
la imagen de la cámara o de la galeria
    promptLabelHeader, //Título que aparecerá cuando se use la cámara
    promptLabelPhoto:"Selecciona una imagen",
    promptLabelPicture:"Toma una foto"
  });
}
```

En segundo lugar, la creación de carpetas compartidas para integrar elementos comunes en varias pantallas como pueden ser los inputs o el logo:

Simplemente escribes `<app-header>` o `<app-logo>` en el HTML y recoge todas las estructuras realizadas en la carpeta compartida, ubicándolas en la página.

```
1 <ion-item class="custom-input" lines="none">
2   <ion-icon style="color: #27AE60;" *ngIf="icon" slot="start" [name]="icon"></ion-icon>
3   <!-- PARÁMETROS Y SUS ATRIBUTOS COMPARTIDOS PARA TODOS LAS CUSTOMIZACIONES DE LOS INPUT-->
4   <ion-input
5     [type]="type"
6     [autocomplete]="autocomplete"
7     [label]="label"
8     [formControl]="control"
9     label-placement="floating"
10  >
11 </ion-input>
12 <ion-button *ngIf="isPassword" (click)="mostrarPassword()" slot="end" fill="clear" shape="round">
13   <ion-icon style="color: #27AE60;" slot="icon-only" [name]="oculta?'eye-outline':'eye-off-outline'"></ion-icon>
14 </ion-button>
15
16 </ion-item>
```

Imagen 23: Html compartido de cualquier input de la app

En tercer lugar, el uso de formularios dentro de `ion-toast`. Que se despliegan al añadir, actualizar productos o actualizar el nombre de usuario.

Por último, los patrones de diseño del menú inicio con el `swiper slides` elementos deslizantes y los efectos usados en el `css`:

```
@keyframes burbujas{
  0%{
    bottom: 0;
    opacity: 0;
  }
  30%{
    transform: translateX(30px);
  }
  50%{
    opacity: .4;
  }
  100%{
    bottom: 100vh;
    opacity: 0;
  }
}

@keyframes movimiento{
  0%{
    transform: translateY(0);
  }
  50%{
    transform: translateY(30px);
  }
  100%{
    transform: translateY(0);
  }
}
```

Imagen 24: Css de Inicio con efecto para las burbujas



Imagen 25: Primer slide de Inicio

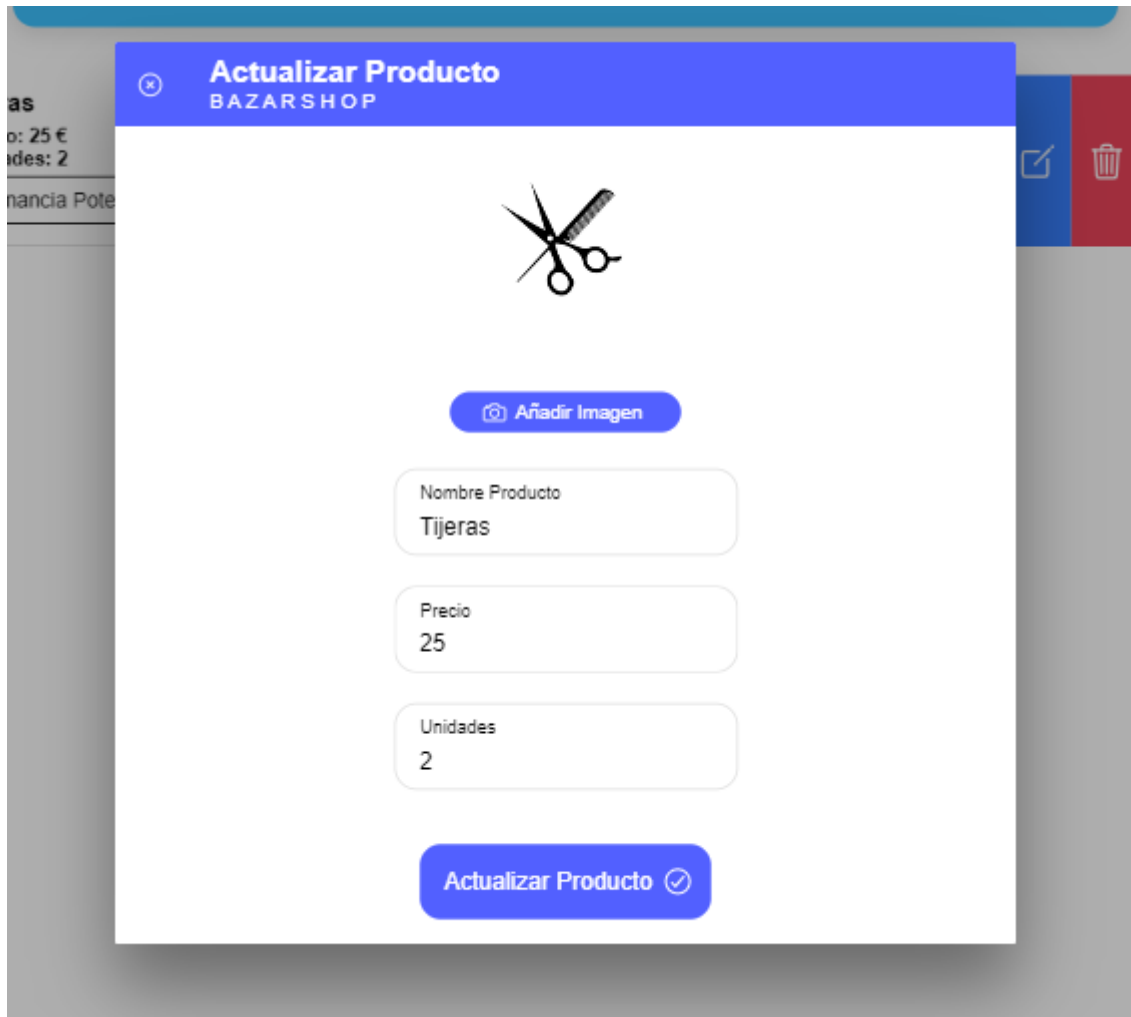


Imagen 26: Ion-Toast

4.5 Pruebas.

En cuanto al apartado de pruebas, se han realizado en varios aspectos claves de la aplicación. El primero en los guards que están declarados en el **app-routing.module.ts**.

```
path: 'login',
loadChildren: () => import('./paginas/login/login.module').then( m => m.LoginPageModule), canActivate:[NoAuthGuard]]
},
{
path: 'registro',
loadChildren: () => import('./paginas/login/registro/registro.module').then( m => m.RegistroPageModule)
},
{
path: 'recuperar-password',
loadChildren: () => import('./paginas/login/recuperar-password/recuperar-password.module').then( m => m.RecuperarPass
},
{
path: 'home',
loadChildren: () => import('./paginas/home/home.module').then( m => m.HomePageModule), canActivate:[AuthGuard]
}
```

Imagen 27: NoAuthGuard en login y AuthGuard en el home

Sirven para que un usuario no logado pueda acceder al home poniendo la ruta en el navegador, y un usuario que está en el home no pueda ir a inicio poniendo su ruta. Para ello tendrá que salir en cerrar sesión.

Es decir, si un usuario estando en home introduce esta ruta:
http://localhost:8100/login

No le dejará acceder y se quedará en el home.

Y si un usuario no logado pone esta ruta en el navegador:

http://localhost:8100/home/principal

No le permitirá acceder y permanecerá en el login. Lo mismo sucede desde la pantalla inicio, de registro y recuperación de contraseña.

Otra de las pruebas que se ha realizado es si el contenido se adapta correctamente a diferentes tamaños de dispositivos como dispositivos móviles y tablets tanto de Android como de Iphone, empleando el servidor que ofrece el navegador.

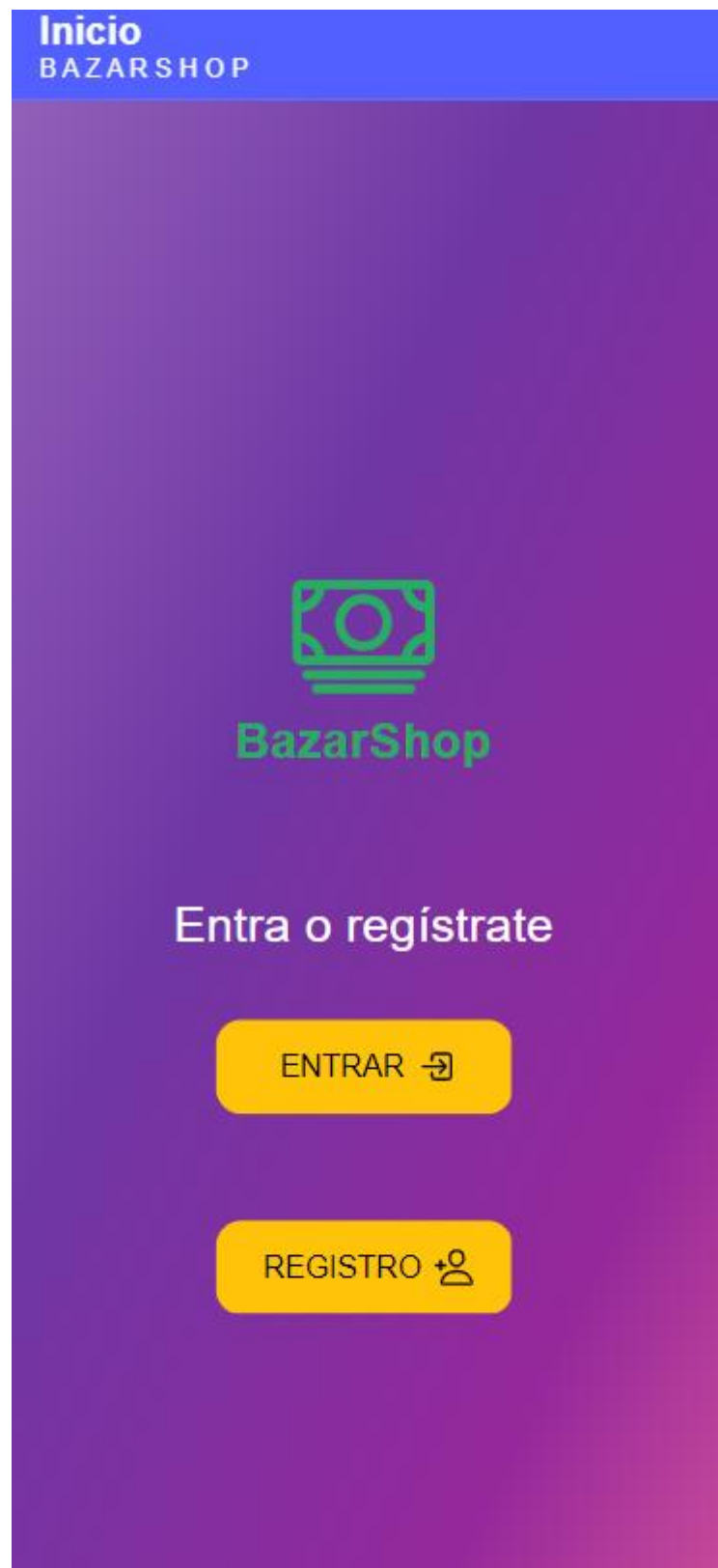


Imagen 28: Vista de inicio desde un Iphone12 Pro

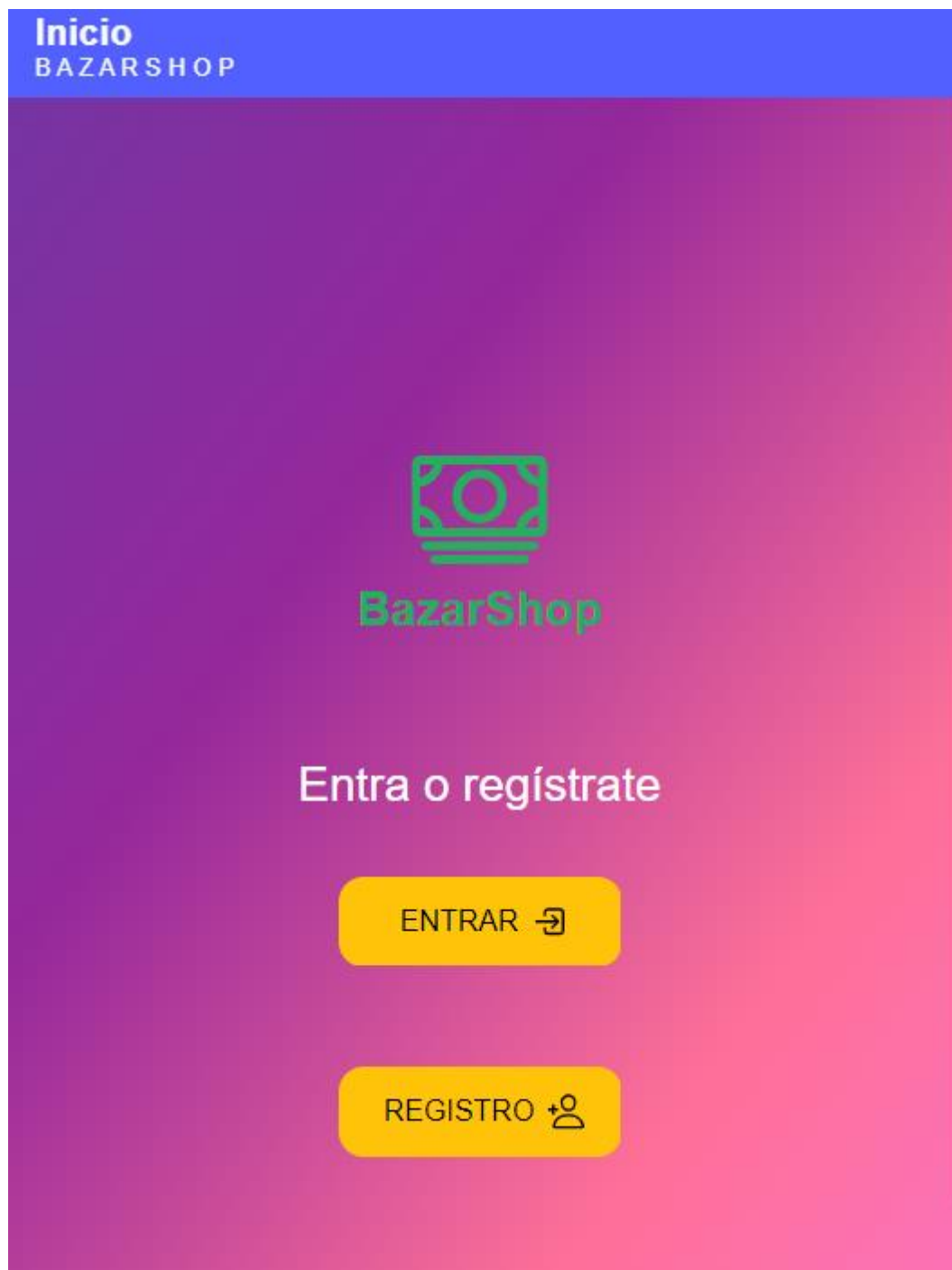
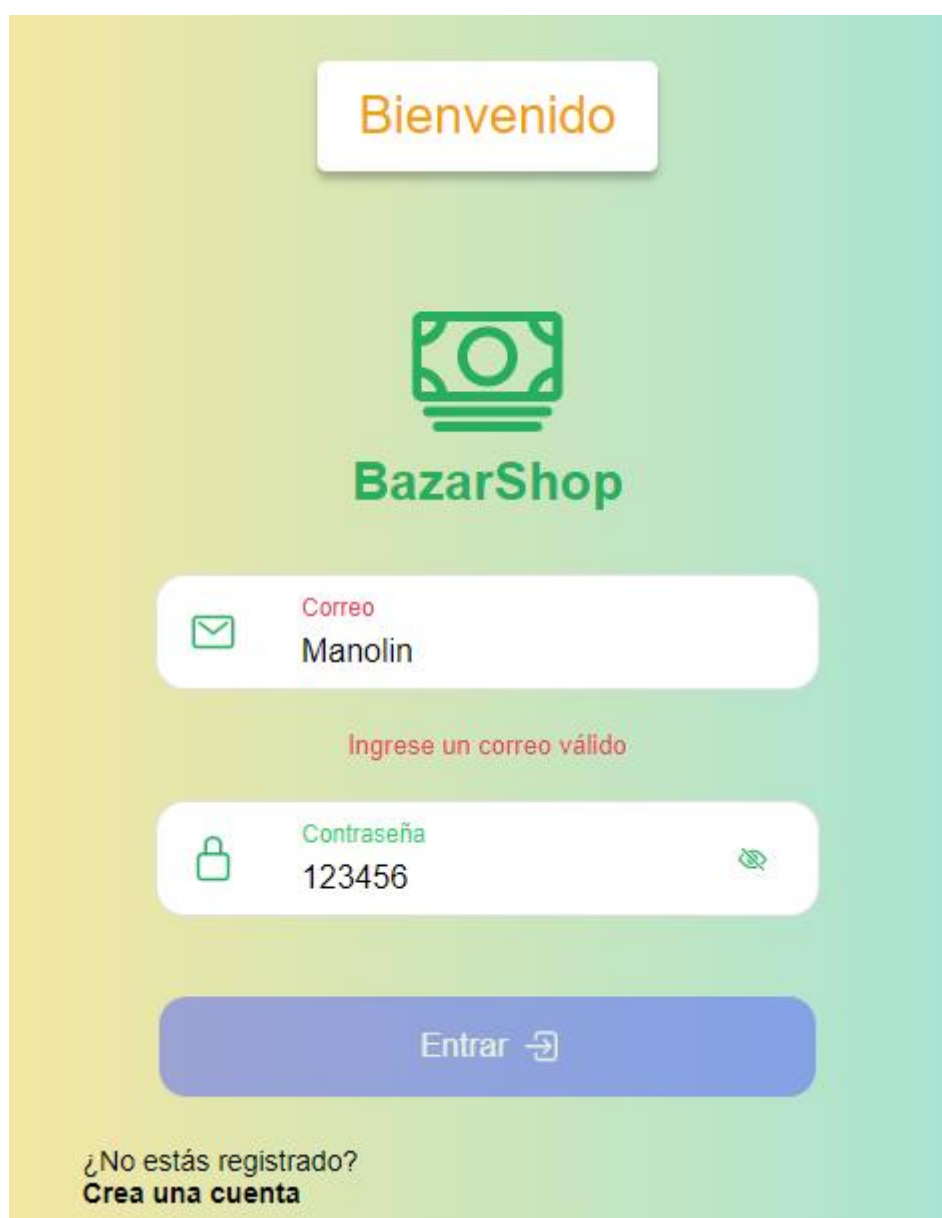



Imagen 29: Vista de inicio desde un Surface Duo


Otra de las pruebas ha sido durante el login de los usuarios comprobando que si un correo o contraseña se introduce erróneamente y no está registrado no te deje acceder. Los correos deben tener un formato válido o el botón del login o registro no estará disponible.





The image shows a login interface for 'BazarShop'. At the top, a white box with an orange border contains the word 'Bienvenido' in orange. Below this is a green icon of a banknote and the text 'BazarShop' in green. There are two input fields: the first is for 'Correo' (Email) with a green envelope icon and the text 'Manolin'; the second is for 'Contraseña' (Password) with a green padlock icon and the text '123456'. A red error message 'Ingresa un correo válido' is positioned below the email field. To the right of the password field is a green eye icon. At the bottom of the form is a blue button with the text 'Entrar' and a right-pointing arrow icon. Below the button, the text '¿No estás registrado? Crea una cuenta' is displayed.

Bienvenido


BazarShop

 Correo
Manolin

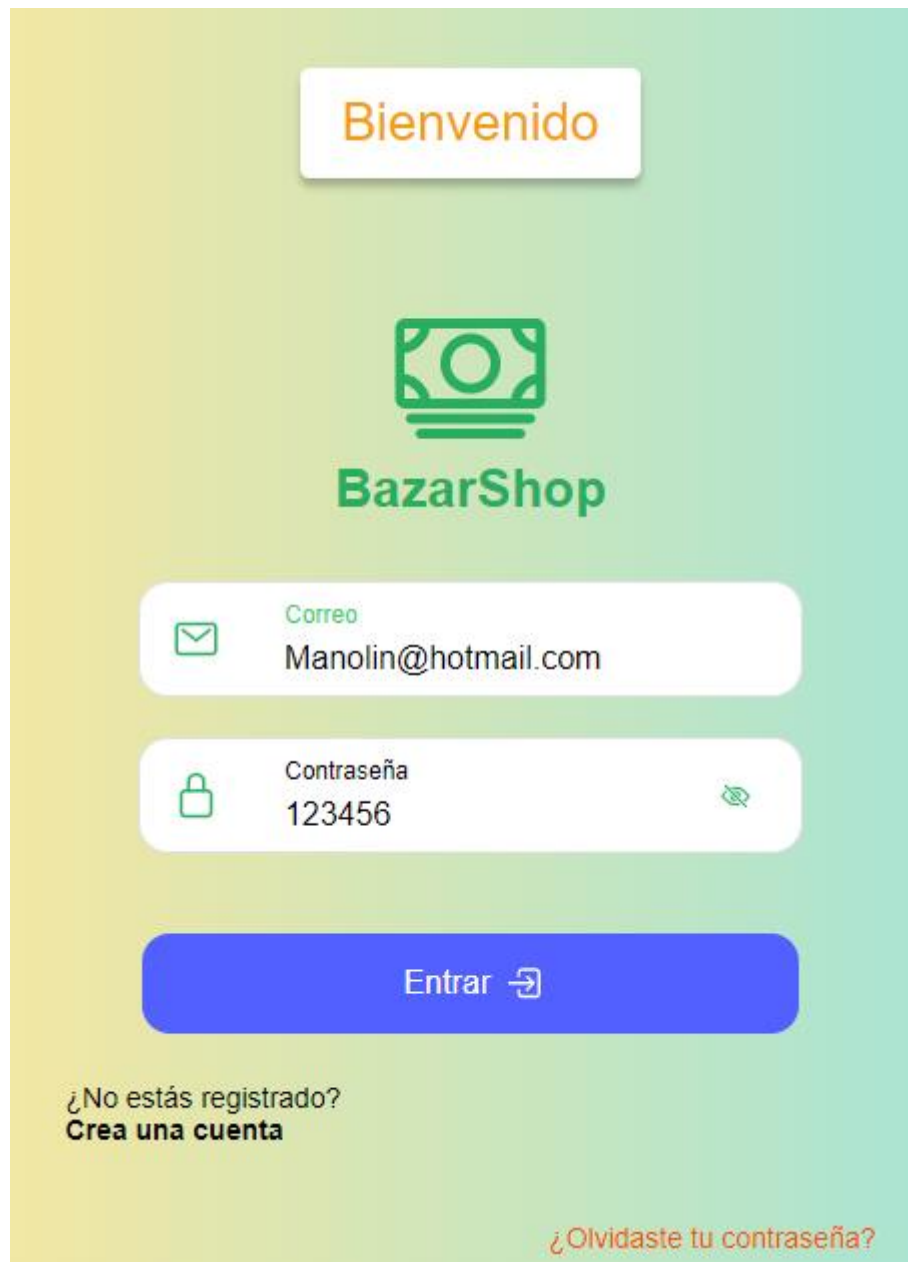
Ingresa un correo válido

 Contraseña
123456 

Entrar →

¿No estás registrado?
Crea una cuenta

Imagen 30: Botón de entrar deshabilitado al no introducir un correo válido



The image shows a login interface for 'BazarShop'. At the top, a white box with an orange border contains the word 'Bienvenido' in orange. Below this is a green icon of a computer monitor with a dollar sign on the screen. The text 'BazarShop' is displayed in a bold green font. There are two input fields: the first is labeled 'Correo' in green and contains the email 'Manolin@hotmail.com'; the second is labeled 'Contraseña' in green and contains the password '123456', with a green eye icon to its right. A large blue button with the text 'Entrar' and a right-pointing arrow is positioned below the fields. At the bottom left, the text '¿No estás registrado?' is followed by 'Crea una cuenta' in bold. At the bottom right, the text '¿Olvidaste tu contraseña?' is shown in orange.

Imagen 31: Botón del login habilitado al introducir un formato de correo válido.

También se ha comprobado con un login erróneo:

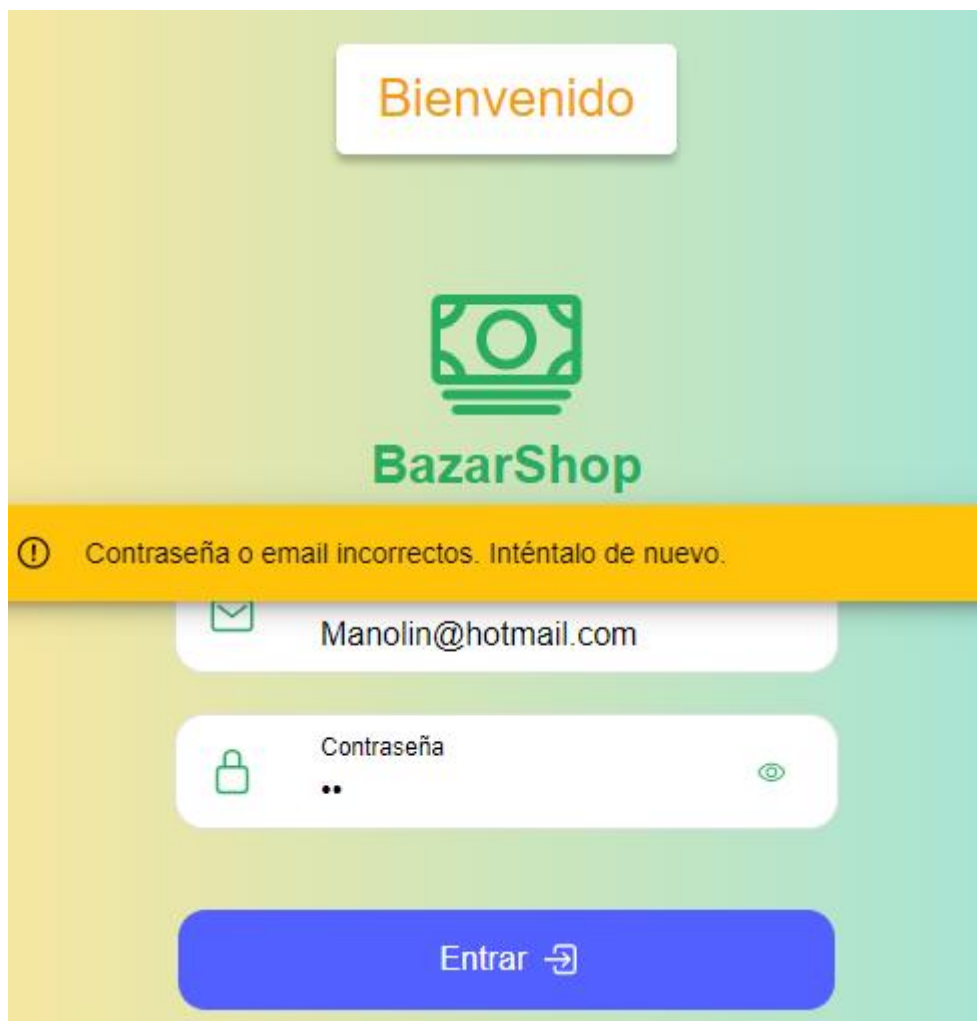
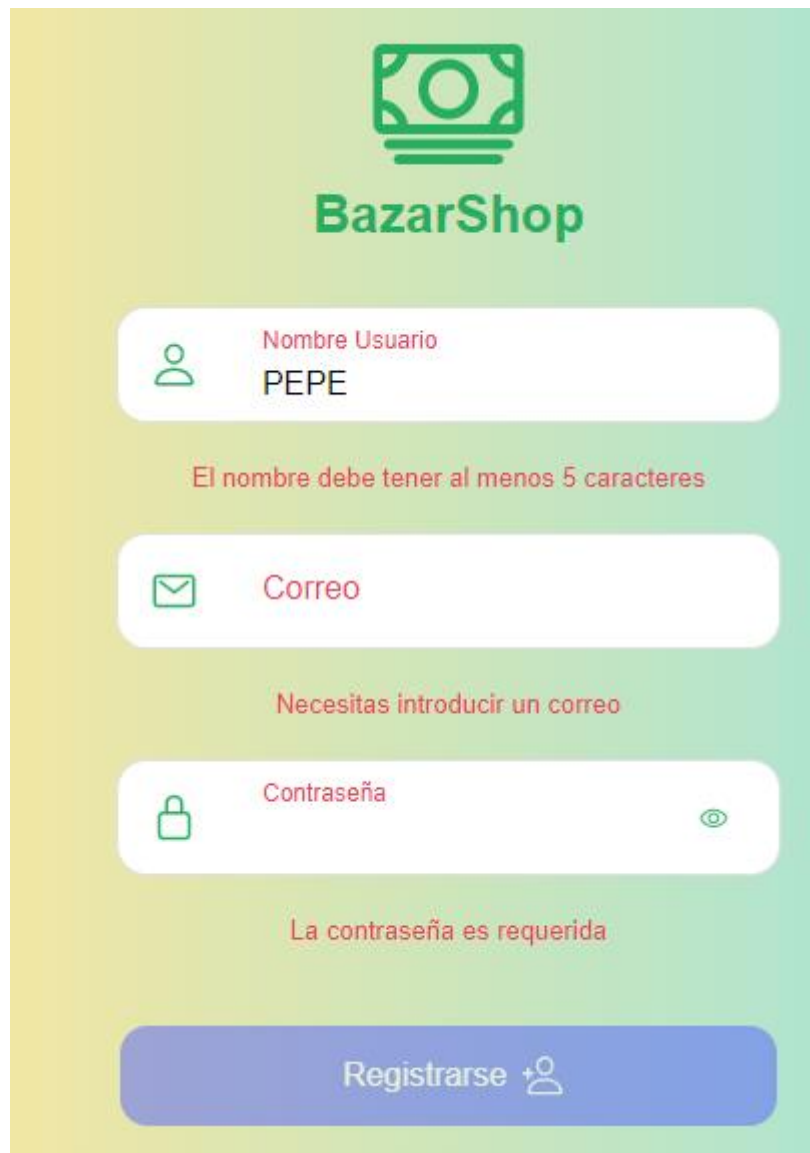


Imagen 32: Login con un usuario no registrado.

Además, se ha comprobado que el nombre en el registro tenga que tener como mínimo 5 caracteres y que ningún campo quede vacío.



The image shows a registration form for 'BazarShop'. At the top is a green icon of a cash register and the text 'BazarShop'. Below this are three input fields, each with a green icon on the left and a label in red text above the input area. The first field has a person icon, the label 'Nombre Usuario', and the text 'PEPE' entered. Below it is a red error message: 'El nombre debe tener al menos 5 caracteres'. The second field has an envelope icon, the label 'Correo', and is empty. Below it is a red error message: 'Necesitas introducir un correo'. The third field has a padlock icon, the label 'Contraseña', and is empty. To the right of the password field is a green eye icon. Below it is a red error message: 'La contraseña es requerida'. At the bottom is a blue button with the text 'Registrarse' and a small person icon.

Imagen 33: Comprobando el registro

Esto mismo se ha realizado con los campos de introducir un nuevo producto o actualizar uno existente.

También se ha comprobado el funcionamiento del modo oscuro y luminoso.

Lo que más problemas ha causado es la actualización de usuarios ya que tiene que cambiar el nombre del usuario en el perfil y en el menú lateral, el del menú lateral no cambia hasta que el usuario no cierre sesión y vuelva a entrar de nuevo.

5 Manuales de usuario

La aplicación BazarShop comienza con un menú inicio que consta de tres pantallas deslizables de forma infinita tanto de izquierda a derecha como de derecha a izquierda, el usuario puede arrastrar la pantalla para cambiar de una pantalla a otra. Las pantallas cuentan con imágenes y efectos para darle la bienvenida de una forma llamativa al usuario.

En la tercera pantalla, existen dos botones. Uno de ellos te da acceso al login y otro a la pantalla de registro.

Pantalla de login

En el Header de la aplicación hay una interacción que te hace regresar a la pantalla de inicio.

En su parte central consiste en dos inputs, uno para el email y otro para la contraseña. Ambos tienen que estar completos y con el formato adecuado en el correo para que el botón de entrar se deshabilite, que en un principio está deshabilitado.

En la parte inferior cuenta con dos interacciones más, una que te da acceso al registro y otra a la pantalla de restablecimiento de contraseña.

Si accedes al registro desde el login al volver atrás regresarás al mismo login. Sin embargo, si accedes a registro desde el botón disponible en inicio al volver atrás volverás a la pantalla de inicio.

Si los campos se introducen incorrectamente en el login la aplicación dará un error bajo cada input indicando que debes completarlo o introducirlo correctamente.

Una vez introducidos ambos campos y pulsar en entrar puede suceder que sea correcto y te dará acceso, tras una interfaz de carga, al home mostrando un mensaje de bienvenida con el nombre de usuario en tonalidad verde.

Si el login es erróneo lanzará un toast de color amarillo indicando que vuelvas a intentarlo.

Pantalla de registro

En esta ocasión, son tres campos a rellenar y ocurrirá lo mismo que en el login. Mostrando un error bajo cada campo si alguno de los campos no cumple las condiciones esperadas o se deja incompleto. Al completar los campos correctamente, el botón de registro se deshabilita y pueden suceder dos cosas:

Registro completo satisfactoriamente y te dará acceso al home tras una interfaz de carga y un mensaje de bienvenida registrándote en la base de datos.

Registro incorrecto porque el correo introducido ya estaba registrado previamente o alguno de los datos es erróneo.


Pantalla de restablecer contraseña

Si se pulsa en el botón situado en el header, volverás al inicio.

Consiste en un simple campo para que introduzcas tu correo en formato válido, una vez realizado el botón de envío se activará y al pulsarlo, enviará un correo automatizado al email introducido indicándolo con un mensaje y haciéndote regresar al login. Si se pulsa en el enlace de ese correo, te enviará a una pantalla para que cambies la contraseña. Si posteriormente intentas entrar a la aplicación con la contraseña anterior, esta fallará ya que se ha actualizado en la base de datos.



Cambia la contraseña de bazarshop-app ➤ Recibidos x

 noreply@bazarshop-app.firebaseio.com
para mí ▼

Hola:

Haz clic en este enlace para cambiar la contraseña de bazarshop-app de tu cuenta roberto.toquero19@gmail.com.

https://bazarshop-app.firebaseio.com/_/auth/action?mode=resetPassword&oobCode=Zm3xJrpwz4Ssi7iWvspy_HhymM05C6Q5glZCD5aqth8AA

Si no has solicitado este cambio, ignora este correo electrónico.

Gracias,

El equipo de bazarshop-app

Cambiar la contraseña

de **roberto.toquero19@gmail.com**

Nueva contraseña



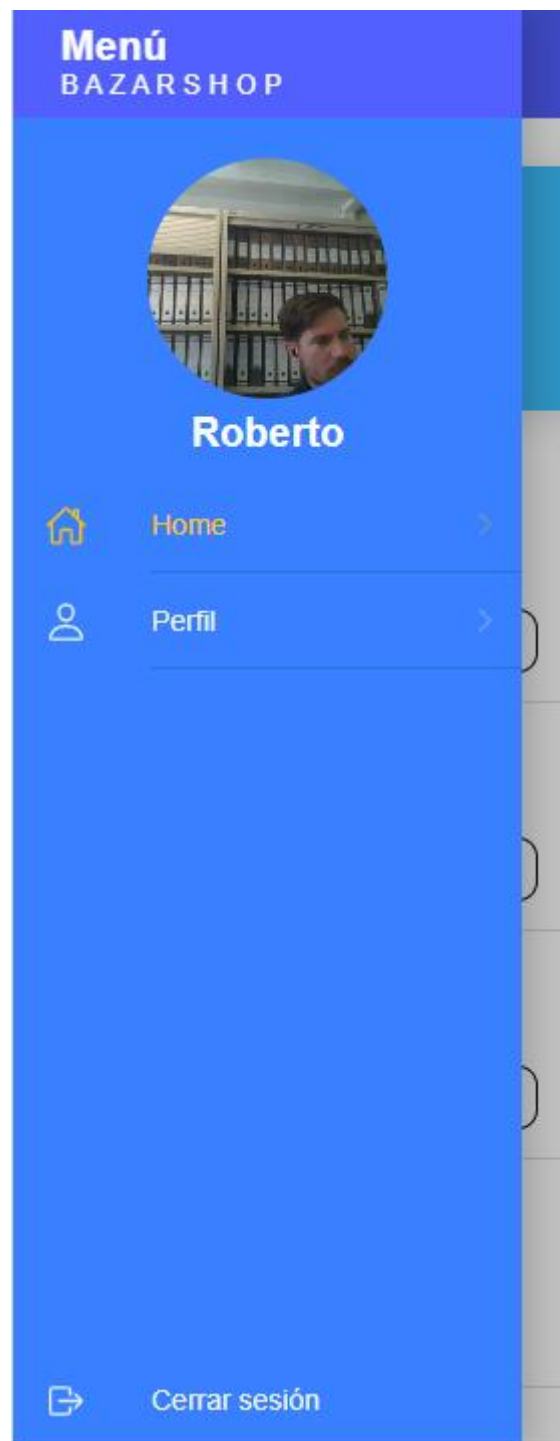
GUARDAR

Pantalla principal

Al entrar por primera vez no habrá ningún producto registrado y sólo estará el logo de la app. En el header hay un botón que despliega el menú lateral, también puede desplegarse deslizando hacia la izquierda.


En este menú se mostrará la foto de perfil, el nombre de usuario y remarcando en amarillo en que página te encuentras y el acceso a la página de perfil.

En la parte inferior estará el acceso al cierre de sesión, que te devolverá al login.



Fuera de este menú, en la pantalla principal en la parte inferior derecha está el botón de añadir productos, al pulsarlo desplegará un Ion-toast con un formulario donde habrá que introducir el campo nombre, unidades y precio de forma obligatoria o el botón para publicarlo no se deshabilitará. En la parte superior puedes cerrarlo pulsando en la X.



 Añadir Imagen

Nombre Producto

Precio

Unidades

Añadir Producto 

Adjuntar imágenes es opcional para subir productos, pero si se decide adjuntarla lanzará un submenú que pregunta si quieres tomar la foto con la cámara del dispositivo o si deseas subirla desde el almacenamiento.


Añadir Producto 

Imagen del producto

Selecciona una imagen

Toma una foto

Si decides seleccionar una imagen, te mandará al explorador de archivos y si decides tomar una foto lanzará la cámara.

Una vez se han completado los campos podrá añadirse el producto pulsando en el botón y esta pestaña se cerrará automáticamente.

De vuelta en el menú principal podrá verse una tarjeta con los datos introducidos en el producto y un indicativo en la parte superior que cuenta los productos y suma las ganancias potenciales.

A medida que se vayan añadiendo productos, estos se mostrarán de mayor número de unidades a menor.

6	895 €
Productos totales	Ganancias Potenciales



Coca Cola

Precio: 2 €
Unidades: 200

Ganancia Potencial: 400 €



Fanta Naranja

Precio: 2 €
Unidades: 100

Ganancia Potencial: 200 €



Kas limon

Precio: 2 €
Unidades: 50

Ganancia Potencial: 100 €



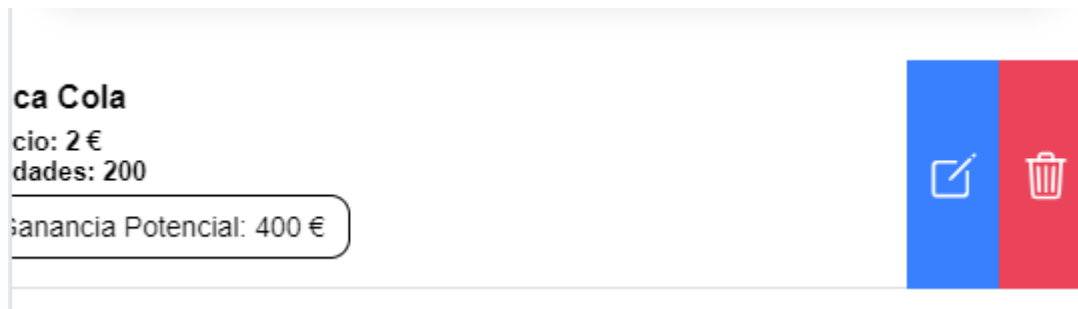
Nestea

Precio: 2 €
Unidades: 40

Ganancia Potencial: 80 €

Si se arrastra la tarjeta de cada producto hacia la izquierda aparecerán dos botones, el primero de ellos es el de actualizar el producto y lanzará de nuevo el toast para modificar los datos.

El segundo será el de eliminar producto y al pulsarlo lanzará un mensaje de confirmación antes de proceder con la eliminación.



Al pulsar en sí, se eliminará de la base de datos y no se mostrará más en pantalla y si se pulsa en no regresará al menú principal.

También se puede arrastrar de arriba hacia abajo para recargar la pantalla.

Pantalla perfil

La pantalla de perfil muestra los datos del usuario, su foto de perfil, un botón para modificarla que lanzará la misma metodología que sucede al subir un producto. Al cambiar la imagen esta se actualizará automáticamente tanto en el perfil como en el menú lateral.

Debajo de la foto se muestra el nombre y el correo del usuario y un menú seleccionable para actualizar perfil y eliminar la cuenta.

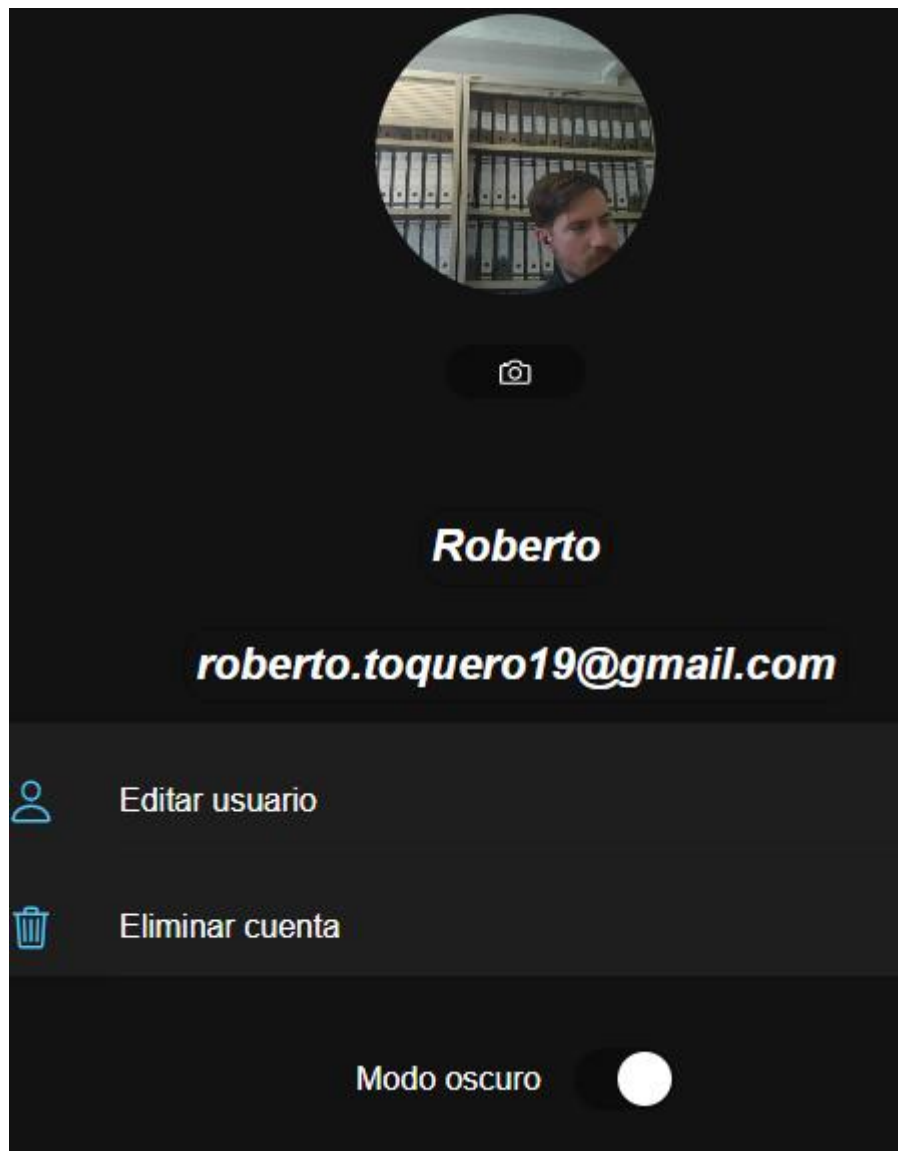
Si pulsas en actualizar perfil lanzará un toast para que puedas cambiar tu nombre. El cambio de email y contraseña aún no ha sido implementado.

Este cambio debe verse reflejado en los datos al arrastrar en pantalla de arriba hacia abajo para recargar la página.

Al pulsar en el botón eliminar cuenta lanzará un mensaje de confirmación como el visto previamente antes de realizar la acción. Si se decide eliminar la cuenta la

aplicación te redirigirá de nuevo al login borrando todos los datos del usuario y sus productos.

Por último, está el toggle que cambia entre modo oscuro y luminoso, al cambiar entre las dos opciones los colores de la interfaz cambiarán entre blanco y negro.



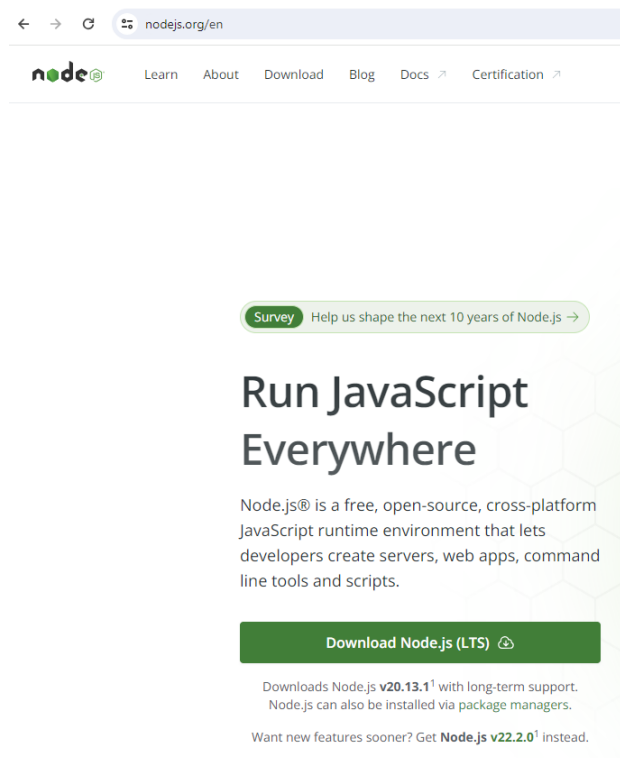
5.1 Manual de instalación

Para llevar a cabo la instalación del programa en un ordenador es necesario primeramente la descarga de NodeJS, también se recomienda tener instalado el IDE visual Studio Code por si se desea comprobar los archivos que posee.

Una vez instalado con el proyecto al ser bajado del repositorio con un git clone es recomendable realizar un comando de:

npm install

Esto es por si ha quedado alguna dependencia o componente desactualizado o no se ha instalado correctamente, en caso de que no deje lo correcto es realizarlo desde un terminal abriéndolo como administrador.



Una vez instalado el programa en el equipo, a través del terminal o un IDE que permita introducir comandos lanzar el siguiente comando:

npm install -g @ionic/cli

Esto producirá la instalación de Ionic en el equipo. Puede comprobarse lanzando el comando:

ionic -v

o

Ionic - - version

Para comprobar la versión que se ha instalado o escribiendo:

Ionic start

```
ionic CLI 7.2.0

Usage:
  $ ionic <command> [<args>] [--help] [--verbose] [--quiet] [--no-interactive] [--no-color] [--confirm] [options]

Global Commands:
  completion ..... (experimental) Enables tab-completion for Ionic CLI commands.
  config <subcommand> ..... Manage CLI and project config values (subcommands: get, set, unset)
  info ..... Print project, system, and environment information
  init ..... (beta) Initialize existing projects with Ionic
  live-update <subcommand> ..... (paid) Ionic Live Updates functionality (subcommands: manifest)
  login ..... Log in to Ionic
  logout ..... Log out of Ionic
  signup ..... Create an Ionic account
  ssh <subcommand> ..... (deprecated) Commands for configuring SSH keys (subcommands: add, delete, generate, list, setup, use)
  start ..... Create a new project

Project Commands:
  You are not in a project directory.
```

Una vez que se ha instalado correctamente, hay que posicionarse en la carpeta principal del proyecto a través de la terminal y lanzar el comando:

Ionic serve

Esto generará un servidor que se lanzará en local en el navegador predeterminado, normalmente en localhost/8100. Al ser una versión para móvil se recomienda ir al **menú de más herramientas→ Opciones→ Herramientas para desarrolladores** o en el comando de teclado **Control + Mayús + I**.

Lo cual permitirá ver la terminal y la pantalla adaptada a la versión móvil, pudiendo escoger el tipo de dispositivo tanto Tablet como móvil.

Como la aplicación puede ser utilizada tantos en Android como en IOS se especificarán los pasos a seguir en cada caso:

Plataforma Android:

A través de la línea de comandos, una vez se hayan realizado todos los pasos previos:

Instalar el paquete **@capacitor/Android** con el comando:

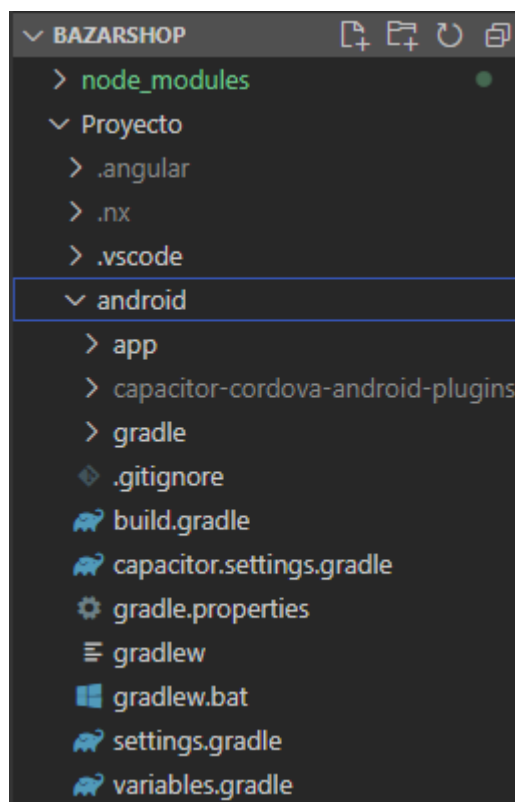
npm install @capacitor/Android

Realizar un **ionic build** en la línea de comandos.

Luego, agregar la plataforma Android con el siguiente comando:

npx cap add Android

Al realizar esto se genera la carpeta Android en el código de la aplicación.



Apertura del Proyecto en Android Studio

Para abrir el proyecto en Android Studio, ejecuta:

npx cap open android

Una vez que se lanza este comando te redirigirá automáticamente a Android Studio y se verá el código del proyecto. Se podrá ejecutar en un simulador, asegúrate de tener los SDK de Android en el programa. Puede que los últimos den fallo, por lo tanto, es recomendable utilizar una versión de Android anterior a la 14.

Una vez comprobado acudir a build → Generar APK.

Tras un tiempo de espera, el APK de la aplicación será generado y almacenado en el equipo. Acude a la carpeta donde se ha guardado pulsando en locate. Con un USB, se puede pasar este archivo a tu dispositivo móvil o a través de un Cloud, por ejemplo.

En tu dispositivo:

Habilitar la depuración USB:

Ve a **Configuración > Acerca del teléfono**.

Toca el Número de compilación siete veces para activar las Opciones de desarrollador.

Vuelve a **Configuración > Opciones de desarrollador y habilita Depuración USB**.

Permitir la instalación de aplicaciones de fuentes desconocidas. Una vez realizados estos pasos, la aplicación quedará instalada en tu dispositivo.

Plataforma iOS:

Descargar el programa **Xcode** desde App Store.

Instalar CocoaPods a través del siguiente comando:

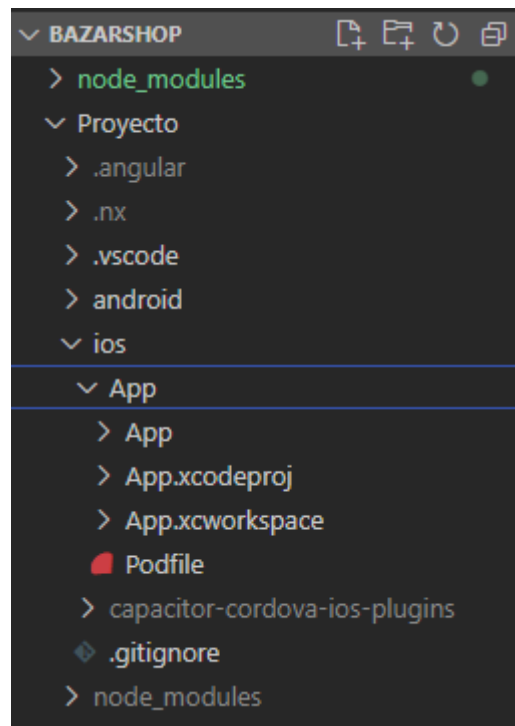
```
$ sudo gem install cocoapods
```

Instalación de **@Capacitor/ios** en la plataforma de iOS:

```
npm install @capacitor/ios
```

```
npx cap add ios
```

Esto genera la carpeta ios en el proyecto:



Después, abrir el proyecto en Xcode utilizando el comando:

`npx cap open ios`

En Xcode se pueden realizar ajustes adicionales si es necesario y se compilar la aplicación para su ejecución en un simulador de iOS o para su instalación en un dispositivo físico.

Para distribuir la aplicación en dispositivos iOS, se necesita un archivo IPA firmado correctamente que puede generarse desde Xcode.

Para descargar el **archivo IPA en dispositivo IOS:**

Cuenta de desarrollador de Apple: Necesitarás una cuenta de desarrollador de Apple si deseas instalar aplicaciones en dispositivos físicos.

Pasos:

En Xcode, ve a **Window > Devices and Simulators**.

En la ventana que se abre, asegúrate de que tu dispositivo iOS esté listado bajo la pestaña Devices. Si no aparece, asegúrate de que está desbloqueado y de confiar en la computadora en el dispositivo.

4. Preparar el archivo IPA

5. Arrastrar y soltar el IPA en Xcode

En la ventana de Devices and Simulators en Xcode, selecciona tu dispositivo en la lista de la izquierda. Arrastra y suelta el archivo IPA en la sección Installed Apps de tu dispositivo.

6. Xcode instalará la aplicación

6 Conclusiones y posibles ampliaciones

La aplicación utiliza tecnologías modernas para ofrecer una experiencia de usuario híbrida, lo que permite su ejecución tanto en dispositivos Android como iOS. Este enfoque facilita el desarrollo y mantenimiento, ya que el mismo código base puede ser utilizado para múltiples plataformas. Tras la realización y prueba de la aplicación veo a Ionic como una grandísima opción para el desarrollo de aplicaciones multiplataforma sobre todo para programadores experimentados en el frontend, con un amplio conocimiento de HTML, CSS y JavaScript poseen una gran ventaja al utilizar esta herramienta.

Una de las características clave de BazarShop es su capacidad para gestionar inventarios, optimizando así la presentación y administración de productos. La interfaz está diseñada para ser intuitiva y accesible, incorporando elementos visuales como el logo de la aplicación y otros elementos visuales típicos de apps para dispositivos móvil, esto aporta buena usabilidad y óptima experiencia de usuario.

Durante el desarrollo, se han abordado y resuelto varios desafíos técnicos, como la correcta configuración de rutas relativas para imágenes y la instalación de dependencias en entornos de desarrollo. También se ha trabajado en la generación y despliegue de aplicaciones para dispositivos móviles, utilizando Android Studio para Android, aunque no he probado en IOS al no contar con dispositivos con ese sistema.

De cara al futuro, hay varias ampliaciones posibles para BazarShop que podrían aumentar su funcionalidad y atractivo. Primero, la mejora de la interfaz de usuario y corrección de errores que seguro que los hay, sobre todo con la mejora de la

actualización de usuarios, pudiendo modificar correo electrónico y empleando la funcionalidad de Firebase que permite la verificación del mismo.

Lo segundo sería la optimización del código y mejora de varios métodos.

Otro aspecto importante que aporta Firestore es la integración de una base de datos en tiempo real que permitiría a los usuarios gestionar su inventario de manera más eficiente y precisa, sincronizando datos entre múltiples dispositivos.

Además, me gustaría incorporar nuevas funcionalidades y campos en la base de datos como filtrar por categorías de productos en un buscador o agregar la funcionalidad de agregar recordatorios o tareas cuando un producto esté cerca de terminarse. A esto ayudaría la incorporación de notificaciones push podría mantener a los usuarios informados sobre actualizaciones importantes, como cambios en el inventario o promociones especiales.

Otra ampliación significativa sería la implementación de características de comercio electrónico directamente en la aplicación, permitiendo a los usuarios no solo gestionar inventarios sino también realizar ventas y transacciones dentro de la misma plataforma. Esto podría incluir la integración de pasarelas de pago seguras y la gestión de pedidos. Haciendo que otros usuarios vean los productos que suben los demás para poder interactuar con ellos y poder comprarlos.

Finalmente, la aplicación podría beneficiarse de la inclusión de análisis y reportes avanzados, ofreciendo a los usuarios insights detallados sobre sus productos y ventas, lo que ayudaría en la toma de decisiones estratégicas. La implementación de inteligencia artificial y aprendizaje automático podría proporcionar recomendaciones personalizadas y predicciones basadas en los datos del usuario.

7 Bibliografía

Ionic Framework. (s.f.). Recuperado de <https://ionicframework.com/>.

CapacitorJS - Camera API. (s.f.). Recuperado de <https://capacitorjs.com/docs/apis/camera>.

CapacitorJS - Android. (s.f.). Recuperado de <https://capacitorjs.com/docs/android>.

CapacitorJS - iOS. (s.f.). Recuperado de <https://capacitorjs.com/docs/ios>.

HTML Color Codes. (s.f.). Recuperado de <https://htmlcolorcodes.com/es/>.

MDN Web Docs - CSS Basics. (s.f.). Recuperado de https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/CSS_basics.

OpenWebinars. (s.f.). Recuperado de <https://openwebinars.net/>.

YouTube - Ionic Framework Tutorial. (2021). Recuperado de <https://www.youtube.com/watch?v=3oQkFwK-mVw>.

Ionicons. (s.f.). Recuperado de <https://ionic.io/ionicons>.

8 Anexos

Enlace a repositorio GitHub del proyecto:

<https://github.com/RoberToquero/BazarShop.git>