
A Marketplace Application with User Authentication, Payment and Cloud Storage

Conor O'Reilly

B.Sc.(Hons) in Software Development

MAY 10, 2020

Final Year Project

Advised by: Daniel Cregg

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)



Contents

1	Introduction	6
1.1	Objectives	7
1.1.1	Mobile Application	8
1.1.2	Secure User Authentication	8
1.1.3	Separate User Functionality by Role	8
1.1.4	Payment System	9
1.1.5	NFC Functionality	9
1.1.6	Cloud Database	9
2	Methodology	11
2.1	Project Management	12
2.1.1	Agile	12
2.1.2	Scrum	12
2.1.3	Testing	13
3	Technology Review	14
3.1	Project Management Tools	14
3.1.1	Git	14
3.2	Testing Tools	14
3.2.1	Selenium	14
3.2.2	Kobiton	14
3.3	NFC Technology	15
3.3.1	NFC	15
3.3.2	NFC Tags	15
3.4	Developer Tools	15
3.4.1	Visual Studio Code	15
3.4.2	Android Studio	16
3.4.3	ASTRO	16
3.5	Application Development Tools	16
3.5.1	Typescript	16
3.5.2	Apache Cordova	17

3.5.3	Angular	17
3.5.4	Node Package Manager	17
3.5.5	Ionic Framework	17
3.5.6	Stripe	18
3.5.7	PhoneGap NFC Plugin	18
3.6	Database	18
3.6.1	Firebase	18
3.6.2	Firebase Authentication	18
3.6.3	Real-Time Database	19
3.6.4	Cloud Firestore	19
3.6.5	Cloud Storage	19
3.6.6	Firebase Functions	19
4	System Design	21
4.1	Database Design	21
4.1.1	Realtime Database	21
4.1.2	Customers Collection	22
4.1.3	User Collection	22
4.1.4	Product Collection	23
4.1.5	Orders Collection	23
4.1.6	Firebase Storage	23
4.1.7	Firebase Authentication	23
4.2	Application Design	24
4.2.1	Login	24
4.2.2	Register	25
4.2.3	Buyer List Page	26
4.2.4	Cart Modal Page	26
4.2.5	Checkout Page	27
4.2.6	NFC Page	27
4.2.7	Orders Page	27
4.2.8	Seller List Page	27
4.2.9	Create Product Page	28
5	System Evaluation	50
5.1	Reflection on Objectives	50
5.1.1	Secure Payment and Authentication	50
5.1.2	NFC	50
5.1.3	Design Elements	51
5.1.4	Testing	51
5.2	Further Development	51
5.2.1	Scalability	52

<i>CONTENTS</i>	4
6 Conclusion	53
7 Appendices	54

About this project

Abstract This project aims to blend the e-commerce world with local community marketplaces. It is a mobile application that allows merchants to register an account and create product listings in which consumers can then browse through and purchase. Authentication security and secure payments were considered and implemented in a modern approach. Authentication is handled through Firebase Authentication which uses industry standard security protocols. Secure payment is achieved via Stripe through card number encryption and HTTP Strict Transport Security (HSTS) which securely sends data over the internet. The use of NFC (Near Field Communication) technology which is becoming commonly found in smartphone devices is implemented and allows users to identify what product an NFC tag is associated with and gives them the ability to purchase this and other products through the application.

Author Conor O'Reilly

Chapter 1

Introduction

E-commerce is defined as transactions made via the internet. It is the buying and selling of goods and services online. Other online activities such as online banking, online ticketing and online auctions are also encompassed into the term e-commerce. The first transaction made on the internet was the purchasing of a book back in 1994. Since then, companies such as Amazon and Alibaba have changed the face of the retail industry. [1]

The Better Business Bureau identified some major reasons people do not buy online is their concern regarding online payment security, the reliability of companies and their privacy policies. [2]. People's trust and familiarity in an eBusiness also greatly influences whether they would be likely to purchase goods and services from such as business. [2]

According to studies done on e-commerce in Ireland, almost 3 billion euro will be spent in 2020 with 3.6 million people on the island purchasing products on the internet [3]. Across the globe, 4.2 trillion dollars are projected to be spent on e-commerce in 2020 with spending showing no signs of slowing down. By 2023, it is estimated that 6.55 trillion dollars will be spent worldwide. [4] Suffice to say, e-commerce is one of the most profitable business models in the modern world. Behemoth companies such as Amazon are testament to this.

In Ireland, weekend and seasonal markets such as the Galway Market and the Clifden Farmer's Market can be found across the country and are a major source of revenue for small businesses and sole traders. They are also an important part of Irish culture and a fabric of the local communities.

This project's aim is to blend the local community markets with the e-commerce world which would open extra revenue sources for the traders and to make the purchasing of goods easier for the customer. With the addition of NFC technology, customers who still wish to go to the market, can simply tap their phone against a tag which is connected to a product and purchase

it right then and there via the application. This replaces the hassle of cash-based transactions for both the customer and merchant.

When creating an account, users simply choose between the role of buyer and seller. Seller accounts can create new product listings for items they have to sell. A picture of the item, which can be taken through the application, as well as some information such as product name, category and price can then be added to the product listing. The unique identification number of the NFC tag is also added to the product listing by the seller simply pressing a button and tapping their phone against a tag. When a product listing is saved by a seller, it is stored in the Firebase real-time database and can be viewed by users logged in as buyer accounts for purchase. The seller can also delete the product listing if they choose to, or if they run out of stock. Product listing made by one seller, cannot be viewed or deleted by another seller.

When a user registers and logs in as a buyer account, every product listing made by all sellers are displayed as a list for buyers to browse through. This simulates a real marketplace where instead of walking by stalls and viewing everything for sale, they simply scroll down on their phone. Products can also be easily filtered by typing in the product name or choosing a category from a drop-down menu, making the buying experience simpler. If a user is at the physical marketplace and sees a product they wish to purchase, they can tap their phone against the NFC tag connected to the product and within a few clicks, have the product purchased and delivered to their home.

1.1 Objectives

These are the main features that we identified as crucial aspects of our project. Features such as secure user login and payment are important for consumer trust in an eBusiness and the familiarity of the layout of the application is also important. Data created in the application had to be stored somewhere and be readily accessible by users, so a cloud database was outlined as an important feature. NFC was also identified as a modern technology that would be interesting to implement into the project.

- Create a mobile application
- Add secure user authentication
- Create separate functionality of buyers and sellers
- Add payment system

- Add NFC functionality
- Implement a cloud database

1.1.1 Mobile Application

Creating a mobile application was extremely important for this project as it enabled us to use available mobile hardware such as camera and NFC functionality. When creating a product listing, the seller can take a picture through their phone of the product and can read the NFC tag's ID which will then be saved into the database. Having it as a mobile application as opposed to a website was also important as when visiting a market, consumers are already likely to have their phone with them and can then scan the tags and make purchases in this way.

The choice to concentrate on Android development rather than to build cross-platform and include iOS, Windows or Blackberry was made by studying the market share held by each operating system in the domain. Android holds a huge 88% of the market share for smartphone users [5] so this was a clear winner for us on this. The available hardware uses available in each device was also looked at. Until iOS 11, there was no API for third party development available on apple smartphone devices. [6] As a result of this, the decision to exclude iOS as a platform for our application was made. Being able to test the application on the device was another reason that Android was the target platform for the application, as an Android device with NFC capabilities was the only available hardware during development.

1.1.2 Secure User Authentication

This is an extremely important aspect of any e-commerce application and it was a major project decision to be made. We researched authentication methods such as AuthO, ServiceBot Auth and Firebase Authentication. We also discussed developing our own authentication system. When we decided that Firebase would be used for other data storage and the security protocols present in their software, we went with Firebase Authentication.

1.1.3 Separate User Functionality by Role

This was a central aspect of the project design. It was clear that buyers and sellers had to have different functionality depending on the role that they chose. Sellers had to be able to create and list products for buyers to browse through. During development, it also became clear that only the seller

who lists a product should have the ability to delete this product from the database. For the buyers, all products from every seller had to be listed and we also had to add a cart and checkout for them to purchase the products.

1.1.4 Payment System

A payment system was needed for the project. We looked at PayPal, Google Pay and Stripe as the main systems to be implemented. Initially, work was done adding PayPal to the project with the Cordova PayPal plugin. However, the plugin seemed to be somewhat outdated and we weren't fully happy with it, so we looked at the other options further. As Stripe was created by two Irish brothers and is currently being widely used, we went with this and found it to be a much better alternative to PayPal. The developer-first mindset that Stripe seems to take was great for our project as we could simulate real transactions with Stripe's test data.

1.1.5 NFC Functionality

We wanted to add a modern technology into the project and were interested in the idea of NFC. NFC can now be widely found in modern smartphones and people are generally familiar with it, so we set out to find a way to implement it into our project. We decided on using the tag as a product identifier which would be attached to the product. When the user scans the tag, the product which it matches to would be displayed on the user's screen and then purchased from there. The idea of adding some sort of locking mechanism to the NFC tag which would then be unlocked when the matching product was purchased was discussed but ultimately never implemented. This was because the applications usability would be decreased as the tag would have to be scanned multiple times. Once when they want to add the product to their cart, and again after the product is purchased. This was also a development hurdle as it was very difficult to prove to the app that the product was purchased and paid for so that the tag could be unlocked. As a result, we decided to leave this feature out.

1.1.6 Cloud Database

Implementing a cloud database was an important factor that was discussed prior to actual development. The two main databases that we looked at were MongoDB and Firebase Real Time Database. We decided on Firebase due to the extra functionality it provided as regards to hosting the application,

hosting functions, authentication and image storage. We also found that the free data limits would not be exceeded during project development.

Chapter 2

Methodology

Christian and I decided to collaborate on the project at the beginning of the academic year. We were assigned our supervisor and quickly got underway in brainstorming ideas. We knew from early on that we wanted to create an E-Business application that included common features that can be found such as user authentication, secure payment and data storage.

We also wanted to include some new form technology that isn't typically found in these types of applications. NFC technology piqued our interest as NFC capable smartphones are becoming more widely accessible at modest price points. We got underway in researching NFC technologies through articles and scientific papers and formed an understanding of how it works.

We liked the idea of having our application complementing a real-world marketplace, so customers could download the app and browse the market's available products and purchase them from the comfort of their own home. They could also attend the market in person and by simply tapping their phone against an NFC tag connected to a product, easily purchase this product virtually, eliminating the need for cash-based transactions and possibly even eliminating the need for vendors to be manning their stalls.

Upon agreeing to a general outline of the functionality of the application, we began discussing which technologies that would best fit the development process of the project. This included what platform to build the application on and what frameworks and languages to use to build the application.

The following are some decisions Christian and I considered in relation to how we are going to go about developing our ideas into a project:

- Application Architecture: What the architecture of the application will be and how will separate parts of the project interact with each other, such as database and mobile application.
- Technologies: What technologies are available to us and which ones will

best suit in implementing our ideas. What frameworks would be used in the project and different development environments will be needed.

- Project Management Tools: What project management tools and methodologies would best suit the project and would maximise communication between group members. How will group members share work between ourselves.
- Testing: How will the application be tested and what testing techniques can be used for different aspects of the project.

2.1 Project Management

2.1.1 Agile

Agile software development, as defined via “The Manifesto for Agile Software Development” comprises of a set of principles that accommodate changes to requirements at any stage of the development process. The principles are not a concrete definition of agility, but instead are guidelines used to deliver software in an agile manner. Agility is used to rapidly and flexibly respond to change in the development process. [7] It allows the software to be continually improved during development and enables early delivery of working prototypes.

We had learnt several types of methodologies for managing software development lifecycles during our time at GMIT and felt that agile was best suited to us as its principles state face-to-face communication as the best way to transfer information and for regular meetings to discuss and react to changing project requirements. Other project management approaches such as the Waterfall Model were considered but as this would require diligent planning, it was ruled out.

2.1.2 Scrum

Scrum is a framework that is used to control and manage the software development process. It increases the speed of production by splitting time into blocks called sprints, which are usually two weeks to a month long. Short daily meetings known as daily scrums are used to track progress on a sprint. Meetings are held at the end of a sprint to review the work that was or wasn’t completed and to present this work to the stakeholders. Discussions are then had with the stakeholders to outline work to be completed in the next sprint.

For our project, we had weeklong sprints in which Christian, and I were assigned a specific workload each week. Then, on the following Monday, we would meet with our project supervisor and discuss what work both of us accomplished during the previous week and set out work for the week ahead. Christian and I also stayed in daily communication in which we would discuss how we are both progressing on the current weeklong sprint and help each other out where needed.

2.1.3 Testing

Selenium was the IDE chosen to test the features of the application. This was done by loading the application on the Google Chrome web browser through the Ionic CLI development server with the command “ionic serve” and then testing features such as login/logout, adding products to cart and switching between pages in the application.

Testing on actual hardware was done through Kobiton. Automated tests were done to check recheck features that were testing on the webpage such as page navigation, login and adding products to cart. It was also used to test hardware specific features such as taking pictures with the phone camera and listening for Ndef events. Automated testing was somewhat limited through Kobiton as the free tier limits had to be adhered to.

Chapter 3

Technology Review

3.1 Project Management Tools

3.1.1 Git

Git is a version control system that is used to track changes in code during the software development process. It allows collaborators on a project to download the newest version of the software they are working on, make changes and upload the newest revision on the code.

Christian and I have used Git extensively during our time in college and always found it extremely reliable to use. Although we did briefly discuss other types of version control software, we decided to stay with Git as it is the best and most popular system to use.

3.2 Testing Tools

3.2.1 Selenium

Selenium is a framework for testing web applications that can run on Windows, Linux and MacOS. It provides a range of tools and libraries that enable functional tests through the Selenium IDE. It also supports tests written several in programming languages such as Java, Python and Ruby. These tests can then be run on all major web browsers. [8]

3.2.2 Kobiton

Kobiton is a testing framework designed for automated and manual tests on mobile devices. It can test hardware capabilities of the device such as camera

control, multi touch gestures and device connection management. It captures all commands made within an app which can then be used for performance benchmark reviews. [9]

3.3 NFC Technology

3.3.1 NFC

NFC (Near-field Communication) is a technology used for short-range contactless communication. It is designed for mobile phones that is mainly used for mobile payments, identification and mobile marketing. [10] It operates in a radius of roughly four centimetres and allows communication between two devices, with both devices being able to send and receive information. It doesn't rely on other digital communication types such as Wi-Fi, Bluetooth or LTE. [11] The technology is commonly found and widely used in the realm of contactless payment. Google Pay and Apple Pay [12] [13] both support contactless payment with NFC enabled devices. Banks such as AIB and Bank of Ireland also have NFC chips in their more recent debit and credit cards with contactless payment of up to EUR50 being possible. [14] [15]

3.3.2 NFC Tags

NFC tags are passive devices that are used to communicate with an active NFC device such as an NFC reader/writer or NFC capable smart phone. There are multiple types of tags that are each based on different standards such as ISO14443A or Sony FeliCa system. [16] Tags can generally be used to read and write information such as website URLs or perform a task such as executing a certain task on a smart phone like opening up a certain app.

3.4 Developer Tools

3.4.1 Visual Studio Code

Visual Studio Code is a light-weight code editor designed by Microsoft for use on Windows, Linux and MacOS. It comes with built-in support for JavaScript, TypeScript and Node.js.

The majority of our project's source code was written in this editor as it comes with pre-installed Git, syntax highlighting, code completion, code

refactoring and has multiple extensions that can be installed for various functionalities such as support for other languages including C++, C#, Java, Python, Go, among others. [17]

As of 2019, Visual Studio Code is the most widely used development environment with 50.7% of survey respondents reportedly using it [18]

3.4.2 Android Studio

Android Studio is the IDE for the Android operating system. It is built on JetBrains' IntelliJ IDEA and is designed to be used for Android development. Kotlin is the preferred language for Android development on this platform. However, Java and C++ are supported in this IDE also. Android Studio has a myriad of features such as Gradle build support, refactoring and Lint tools.

The Android SDK tools compile source code, data and resource files into an APK (Android package) which is then used to install the app on Android devices. [19]

3.4.3 ASTRO

ASTRO is a utility application for Android that is used for file management. It is used for navigating folders on Android devices and performing tasks on files such as Copy, Move and Delete. It supports zip and tgz files, making it possible to compress or extract archive files.

This application was used to circumvent a problem we experienced during the hardware testing phase in that our phones were not being recognised by our laptops, so we weren't able to target our devices while building the APK to test. Instead, the APK was built in android studio and then we would email it to ourselves and use ASTRO to extract the file onto our phones for testing.

3.5 Application Development Tools

3.5.1 Typescript

Typescript is a superset of JavaScript that provides functionality for static typing, classes and interfaces. It is designed for the development of large applications and compiles to JavaScript through the Typescript compiler. Bugs and errors are caught early in development as the Typescript compiler informs the IDE on its rich type information. Static typing enables tooling and IDE support and establishes a robust codebase. [20]

3.5.2 Apache Cordova

Apache Cordova is a web development framework that allows the use of standard web technologies for cross-platform development. Applications are executed in wrappers that target each platform and use API bindings to access device capabilities such as sensors, data and network status. [21]

Plugins are central to Cordova. A set of plugins, Core Plugins, are maintained by Apache and allows applications to access device capabilities such as battery, camera and NFC sensors. [21] Developers can also create their own plugins using JavaScript.

3.5.3 Angular

Angular is a TypeScript based app-design framework. [22] It is a complete rewrite of AngularJS that includes new features such as dynamic loading, asynchronous template compilations and iterative call-backs provided by RxJs.

3.5.4 Node Package Manager

Npm is a package manager that is used for the runtime environment Node.js. It has an online registry containing over 800,000 code packages in which developers share software. Organisations also use npm to manage private deployment. [23] The online registry is accessed via the npm command line client.

3.5.5 Ionic Framework

Ionic is an open-source user interface toolkit that is used for building mobile and desktop applications using standard web technologies HTML, CSS and JavaScript. [24] Originally built on top of Angular and Apache Cordova, the web components included in recent universally pair with JavaScript frameworks such as Angular, React and Vue. The components can also be used without any interface framework. Backend connections to Ionic apps are plentiful with options such as AWS, Azure and Firebase available.

Ionic is installed and updated through the Ionic CLI. It also comes with built-in debugging tools and a development server.

3.5.6 Stripe

Stripe is a payment processor that supports electronic transfer of money from a customer's bank to a merchant's bank. It accepts most payment types such as Visa, MasterCard and American Express, along with mobile wallets like Google and Apple Pay. It comes with a custom UI toolkit that allows merchants to create their own custom payment form for applications. It also has a pre-designed payment form to be imbedded in applications. Creating custom invoices and payment requests are also easily created with Stripe. [25]

3.5.7 PhoneGap NFC Plugin

This plugin is used to write and read NFC tags. It uses NDEF (NFC Data Exchange Format) which allows for maximum compatibility between NFC tags and devices. It is supported on most platforms such as Android, IOS, Windows and Blackberry, and can be installed with the use of Cordova.

Install command: `cordova plugin add phonegap-nfc` [26]

3.6 Database

3.6.1 Firebase

Firebase is a Backend-as-a-Service (Baas) on the Google Cloud Platform. It takes away the need for developers to manage their own servers and write APIs. [27] It provides a suite of tools and services for the development of mobile and web applications. The features that we used for the development of our application provided by Firebase are authentication, real-time database, hosting, cloud functions and storage.

3.6.2 Firebase Authentication

Firebase Authentication provides backend services to authenticate users to applications through ready-made UI libraries. It supports authentication using passwords, phone numbers and authentication APIs such as Google, Twitter and Facebook. [28]

Authentication credentials are collected from the user within the application, be it an email address and password or federated identity provider. These credentials are then passed to the Firebase Authentication SDK in which they are verified using the Firebase Authentication backend services and a response is returned to the client.

3.6.3 Real-Time Database

The Firebase Real-time Database is a NoSQL cloud-hosted database. Data is stored as JSON and is synced across all clients in real-time. The database remains responsive when offline as it persists data to disk. Upon re-establishing connectivity, data is synchronised server state and receives any changes it missed. [29]

3.6.4 Cloud Firestore

Like the real-time database, data is synced across client apps through real-time listeners and offline support is offered with the Cloud Firestore database. Data is stored in documents, ordered into collections. Documents can then contain nested objects and sub-collections. Queries can be performed on the database to retrieve specific documents or all documents in a collection. [30]

3.6.5 Cloud Storage

Cloud Storage is an object storage service used to store and serve user-generated content such as pictures and videos. [31]

3.6.6 Firebase Functions

Cloud Functions is a server-less framework lets applications to run backend code automatically in response to events triggered by HTTPS requests and Firebase Functions. [32] These Firebase Functions are deployed through the Ionic CLI (or whatever framework the app is being built in) and is stored in the Google Cloud. Various Firebase and Google Cloud features can be triggered through these functions such as authentication, storage and online payments.

Figure 3.1: Functions


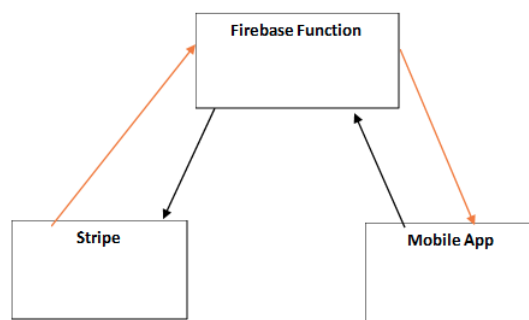
Function	Trigger	Region	Runtime	Memory	Timeout
createStripeCust...	 user.create	us-central1	Node.js 8	256 MB	60s
getCustomerOrders	HTTP Request https://us-central1-accountdb-d624e.cloudfunctions.net/getCustomerOrders	us-central1	Node.js 8	256 MB	60s
startPaymentInte...	HTTP Request https://us-central1-accountdb-d624e.cloudfunctions.net/startPaymentIntent	us-central1	Node.js 8	256 MB	60s
webhook	HTTP Request https://us-central1-accountdb-d624e.cloudfunctions.net/webhook	us-central1	Node.js 8	256 MB	60s

Figure 3.2: Functions



Chapter 4

System Design

4.1 Database Design

4.1.1 Realtime Database

Within the Firebase real time database, there are four collections. Two of the collections contain information that relate to user sign in and user storage data: user collection and customers collection. For the remaining two collections, there is the orders collection which contains data that pertains to when a user makes a purchase. The second remaining collection is products which contains the data that is stored from when a seller creates a new product entry

Figure 4.1: Environment

```
export const environment = {
  production: false,
  firebase: {
    apiKey: "AIzaSyDDusMHwYjCPVKz-UMGzt6AAYhxTKmj_PE",
    authDomain: "accountdb-d624e.firebaseio.com",
    databaseURL: "https://accountdb-d624e.firebaseio.com",
    projectId: "accountdb-d624e",
    storageBucket: "accountdb-d624e.appspot.com",
    messagingSenderId: "941414432769",
    appId: "1:941414432769:web:a363f0b3161fe1746ac000",
    measurementId: "G-B4NFVBF7NE"
  },
  stripe_key: "pk_test_BFC3tIr6KloVyMp0GvsMqLTw00YljY7B90"
};
```

4.1.2 Customers Collection

The customer's collection has data that relates to the user collection. The document has the unique customerID for every customer and the field has the relating unique ID created by firebase upon a customer registering a new account. This data is used to match orders to a customer.

Figure 4.2: Customer Collection

customers >	cus_GvI5xtpuY4xcrh >	+ Add field
orders	cus_GvWLWYDGKJ0jDG	user: "aEuaMGmJLcWC2PTS03epGCmcKLP2"
products	cus_GvzCuiKELuxdZD	

4.1.3 User Collection

The user's collection contains the data that is created upon a user registering a new account. It contains the time and date of when the user registered, the email and name of the user and the role of the user (buyer or seller) If the user created a buyer account, a unique customer ID is also created.

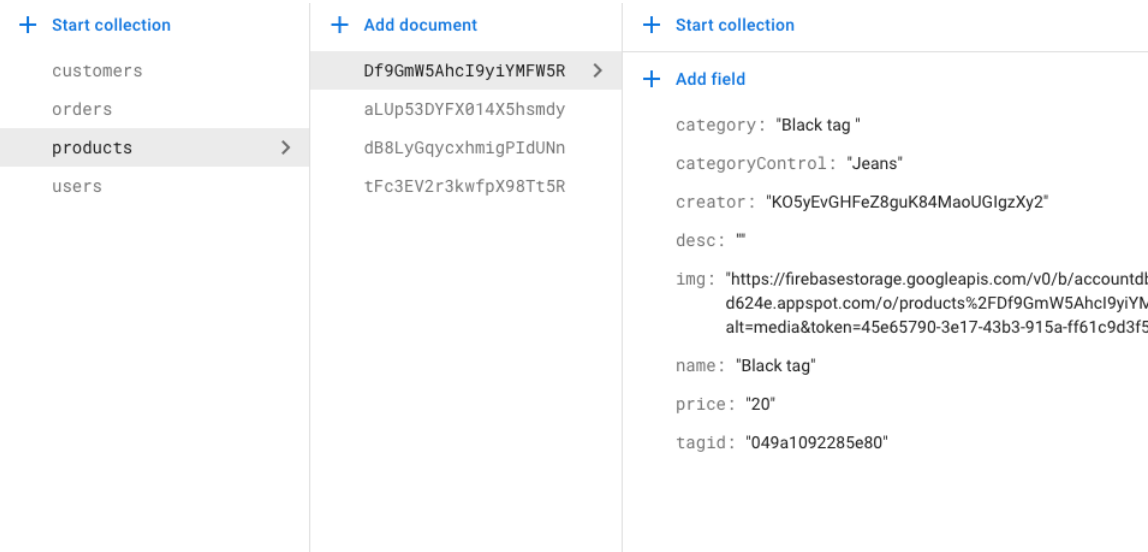
Figure 4.3: User Collection

+ Start collection	+ Add document	+ Start collection
customers	2A6RJ4FwPsZdxbZN2Sv8pTz	+ Add field
orders	4ey7ptSSQfb80W9K08Egdefl	created: April 1, 2020 at 5:15:36 PM UTC+1
products	6Sqod9TxxxRhGtHk4xbZ34d7	customerId: "cus_H1BbOoUgmujPg1"
users >	6iSNovGdbNSAchXGa1S6dk0L	email: "buyertest@test.com"
	AbJkgP040uSHWGCz7IH30Ds	name: "buyer"
	CtQgA1o8vSYHhFGWDFSSibM	role: "BUYER"
	K05yEvGHFeZ8guK84MaoUGIg	
	KgLLpjWgqUMMVe12Cs9HetRC	
	OstCprUA1sejBD0J1AJx2GMr	
	WqM8A4z5cuYP5pUMeeKqRqV3	
	aEuaMGmJLcWC2PTS03epGCmc	
	aTcLITSdw3UNEUTH1tJm0PkL	
	bLph4eocKQMBEr1abqnC7E9w	
	nF6ThnFYM7Fv0WnVSm4v4Nt	

4.1.4 Product Collection

The product collection contains documents that are created when a seller enters a new product into the app. Each document contains the creator’s unique ID. It also contains the name, description, price, category, NFC tag ID of the product and a link to the product’s image in Firebase Storage.

Figure 4.4: Products Collection



4.1.5 Orders Collection

The orders collection contains the order summary data from when the user makes a collection and it was verified by Stripe. It contains data on the cost amount of the order, the customer ID of the user who made the purchase, a link to the invoice generated by Stripe, the specific items ordered, payment status and the user ID of the user who made the purchase

4.1.6 Firebase Storage

Firebase storage is used to store product images from when the seller creates a new product. They are stored as a png file in the products folder.

4.1.7 Firebase Authentication

When a user registers a new account in the app, this authentication information gets stored here. As we used E-Mail as the identifiers, the identifier

Figure 4.5: Orders Collection

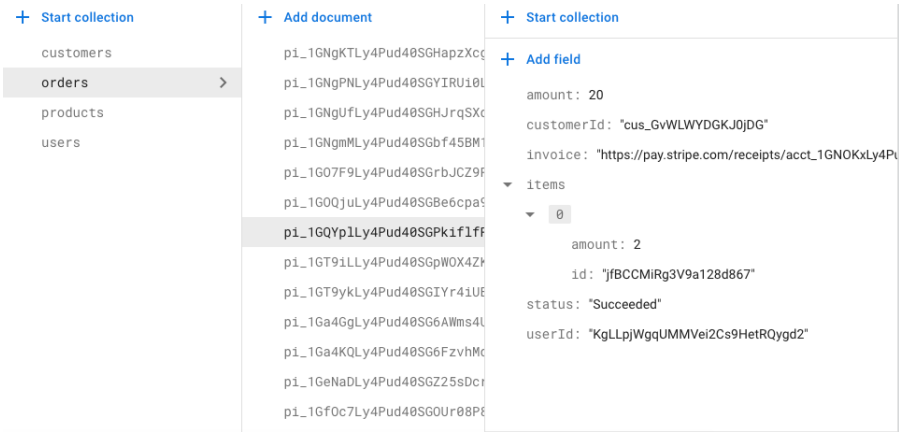


Figure 4.6: Storage



section is the E-Mail that the user enters upon registration. It also contains information of when the user created the account and last logged in and the user UID which can be found as the document ID for each user in the User Collection of the real time database.

Figure 4.7: Authentication

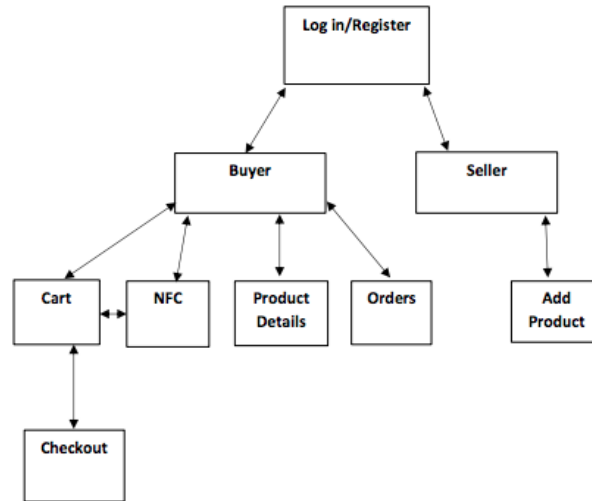


4.2 Application Design

4.2.1 Login

The login page is displayed to the user when the app is opened. A form that is stored in an Ionic container is presented to the user to enter their authentication information: E-Mail and password. The sign-in button is inactive until the user enters what the application recognises as a valid E-Mail format (Example@example.com) and a password that is at least six characters in length. At this point, the sign-in button becomes activated and the user can press it. If the E-Mail doesn't correspond to an E-Mail

Figure 4.8: Application Design



identifier in Firebase Authentication, an error message is returned detailing how no user with this E-Mail address can be found. If a registered E-Mail address is entered but with an incorrect corresponding password, an error message is returned detailing that the password enters does not match the E-Mail address. If both authentication form fields are correct, the user is logged in and they are loaded into a different page depending on their role: buyer or seller.

4.2.2 Register

From the login page the user can also register a new account by clicking the “create account” button at the bottom of the form. A flip effect using CSS and Ionic elements is used to create an animation that switches between the login and register forms. The register form prompts the user to create an account by entering their E-Mail, name and a password. It also has a select role option for a buyer or seller account. The E-Mail the user enters is checked against E-Mails stored in Firebase and returns an error message if this E-Mail is already in use. When the user enters in valid information and presses the register button, an account is created, and they are loaded into a different page which depends on if they chose a buyer or seller role. A Stripe account is also created for them here using Firebase Functions.

When a new account is registered, the sign up method and new Stripe customer function are invoked and sent to Firebase. An account is created in

Stripe and a new user is created in Firebase Authentication. A new document is also made in the user database collection and if a customer account is created, a document is created with data in the customer collection.

4.2.3 Buyer List Page

When a buyer account is logged in or registered, the Buyer List page is loaded. This page contains all the seller listed products for the buyer to choose from. Angular is used here to display the data from the Product's collection in the real time database in an ordered list. Each product's picture, name, category and price are displayed here. Angular's RouterLink is also used on each product so that if the user clicks on a product, they are brought to a new page which contains more information on the product and the ability to add and remove the product to and from their cart. The products unique ID is used to determine that the correct product is loaded when the user clicks on it.

On the Buyer List page, there are four buttons that each redirect to a different page: the NFC button which goes to the NFC page, the order button which goes to the orders page, the cart button that opens the cart module page and the logout button, which logs the user out and redirects back to the login page. It does this by removing authentication permission from the user. There is also a search box function and a sort by categories function. These both work in the same way in that there is a pipe implemented that filters the products. The search box function takes text input and converts it to lowercase. It then checks it against the categories and name fields of the products and return only those products with matching text. The filter by category option uses the same pipe as the search box but the user clicks a button and returns the products with the matching category that they chose.

4.2.4 Cart Modal Page

The cart modal page contains the products that the customer added to their cart. They can add or remove products to and from their cart from here. A simple formula that adds up their total purchase cost is implemented on this page.

On the bottom of this form is a checkout button that the user can click. This brings up the checkout page.

4.2.5 Checkout Page

The checkout page contains a form that the customer must enter to complete their purchase. The customer's name, street, city, eircode/zipcode, country and card details are entered into this form. When the customer clicks the buy now button after filling in their information, a payment intent is created by the payment intent function in Firebase Functions and is sent to the Stripe API. In other words, when the user makes a purchase, the application makes a call to Stripe through Firebase. Stripe then returns data to Firebase which in turn returns this data to the app.

4.2.6 NFC Page

The NFC page is accessed through a button on the Buyer List page. There is a button on this page that when clicked, checks if the phones NFC is enabled. If it is not, a message will be displayed prompting the user to turn on their NFC. If the devices NFC is turned on, the user will be prompted to tap their phone off an NFC tag. When the user taps their phone off an NFC tag, the NFC tag ID is read by the app and then checked against the tagid field in the real time database's product collection. If a matching tagid is found in the database, the product containing the tagid is displayed on the page. This functions similarly to if the user clicked on a product in the Buyer List page. The user can then add the product to their cart from here. If the tagid is not found in the database, an error message will be shown.

The cart button that can be found in the Buyer List page is also on the NFC page and functions the same way. The user can navigate back to the Buyer List page from here.

4.2.7 Orders Page

When an order is made by the customer, the orders page contains details of the transaction. This is done through the customer order function which requests data from the Stripe API. This includes the time and date of purchase, price of purchase, items ordered and payment completion information. There is also a button that the user can click which displays an invoice created by Stripe through the in-app browser.

4.2.8 Seller List Page

When a user with the seller role is logged in, the Seller List page is displayed. This page displays all the products that the seller with the UID logged in

has created. Similarly, to the Buyer List page, angular is used to display the products from the database but limits it to those with a certain UID. They can delete a product from here which is then removed from the real time database. At the bottom of this page, there is a button that when clicked, redirects to a new page that allows the user to create a new product.

4.2.9 Create Product Page

The create product page contains a form for the seller to fill out to create a new product. The form fields here include image, product name, price, category, description and NFC tag ID.

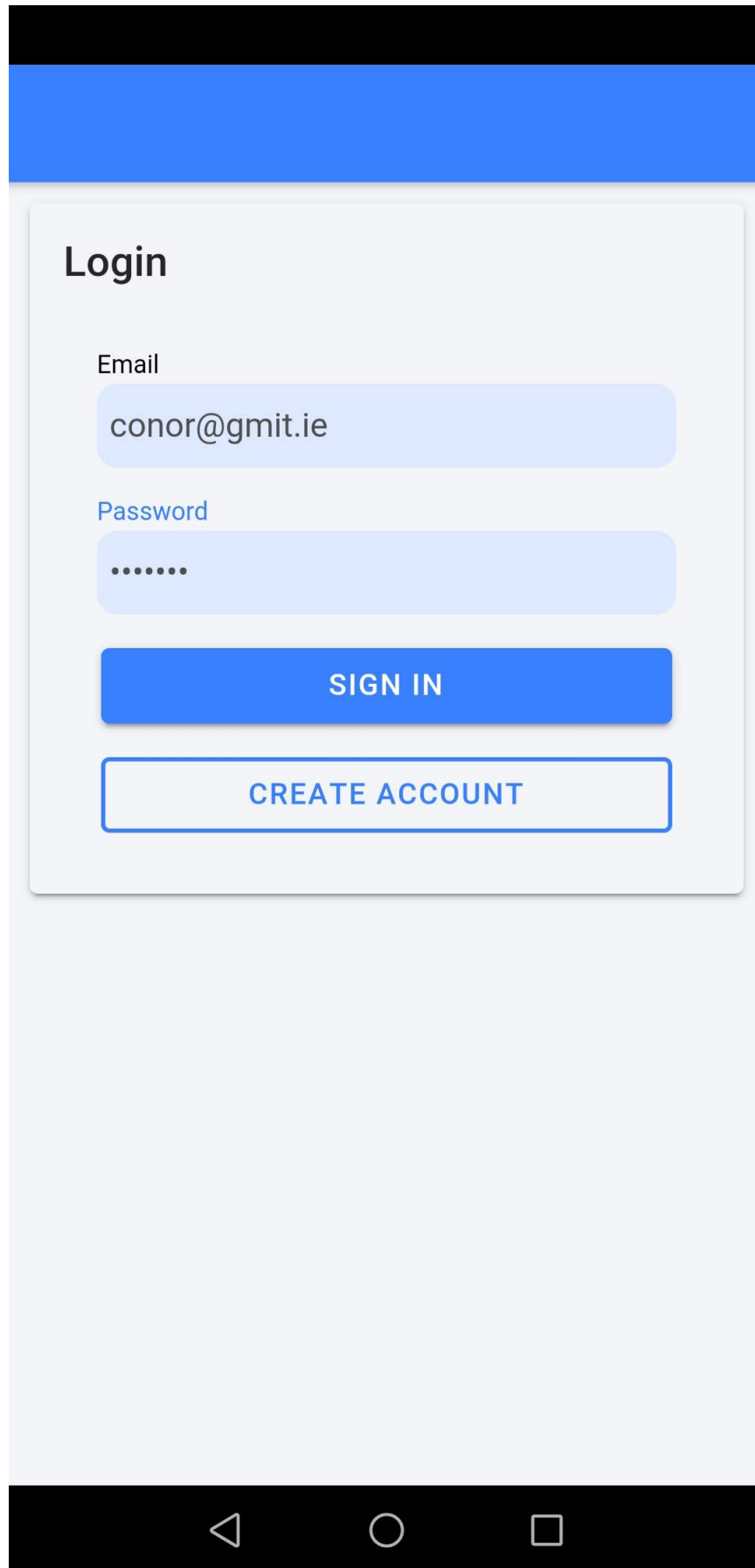
The Apache Cordova camera plugin is used which allows the user to take a picture of the product (device permission may have to be granted in the app). The image is then saved as base 64 in the products folder in Firestore.

The category uses a form control and displays a drop-down menu for the user to choose their category from.

The tagid field is filled by tapping the device off an NFC tag. This works similarly to reading an NFC tag id in the NFC page.

When all the necessary fields are filled, the user clicks the create product button and a new product is created in the real time database.

Figure 4.9: Login Page



The image shows a mobile application login screen. At the top, there is a black status bar. Below it is a blue header bar. The main content area is a light gray card with a white background. The card has a title "Login" in bold black text. Below the title, there are two input fields. The first is labeled "Email" and contains the text "conor@gmit.ie". The second is labeled "Password" and contains seven dots. Below the input fields, there are two buttons. The first is a solid blue button with the text "SIGN IN" in white. The second is a white button with a blue border and the text "CREATE ACCOUNT" in blue. At the bottom of the screen, there is a black navigation bar with three white icons: a triangle, a circle, and a square.

Login

Email

conor@gmit.ie

Password

.....

SIGN IN

CREATE ACCOUNT

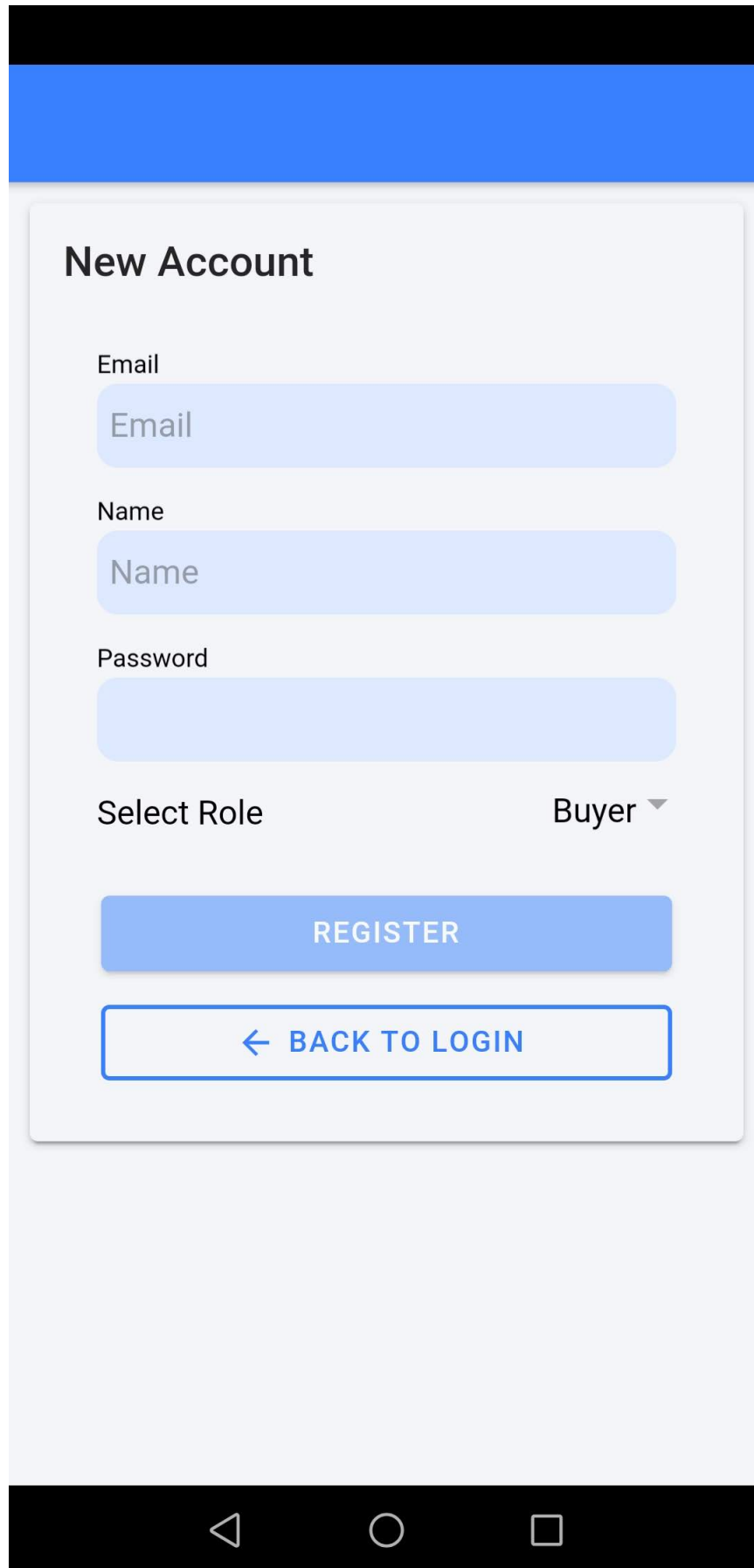
Figure 4.10: Login Form

```
this.loginForm = this.fb.group({  
  email: ['', [Validators.required, Validators.email]],  
  password: ['', [Validators.required, Validators.minLength(6)]]  
});
```

Figure 4.11: Login Method

```
// sign in method  
signIn(credentials): Observable<any> {  
  return from(this.afAuth.auth.signInWithEmailAndPassword(credentials.email, credentials.password)).pipe(  
    switchMap(user => {  
      console.log('real user: ', user);  
      if (user) {  
        return this.db.doc(`users/${user.user.uid}`).valueChanges();  
      } else {  
        return of(null);  
      }  
    })  
  )  
}
```

Figure 4.12: Register Page



The image shows a mobile application interface for a registration page. At the top, there is a black header bar, followed by a blue horizontal bar. Below this is a light gray card with a white background and a subtle drop shadow. The card is titled "New Account" in a bold, black, sans-serif font. Below the title, there are four input fields, each with a label above it: "Email", "Name", "Password", and "Select Role". The "Email" and "Name" fields have placeholder text "Email" and "Name" respectively. The "Password" field is empty. The "Select Role" field is a dropdown menu with the text "Buyer" and a downward arrow. Below the input fields, there are two buttons: a solid blue button labeled "REGISTER" and a white button with a blue border labeled "← BACK TO LOGIN". At the bottom of the screen, there is a black navigation bar with three white icons: a triangle, a circle, and a square.

New Account

Email
Email

Name
Name

Password

Select Role Buyer ▾

REGISTER

← BACK TO LOGIN

Figure 4.13: Register Form

```

this.registerForm = this.fb.group({
  email: ['', [Validators.required, Validators.email]],
  password: ['', [Validators.required, Validators.minLength(6)]],
  name: ['', Validators.required],
  role: ['BUYER', Validators.required]
});

```

Figure 4.14: Register Method

```

// sign up method
signUp(credentials) {
  return this.afAuth.auth.createUserWithEmailAndPassword(credentials.email, credentials.password).then(data => {
    console.log('after register: ', data);
    return this.db.doc(`users/${data.user.uid}`).set({
      name: credentials.name,
      email: credentials.email,
      role: credentials.role,
      created: firebase.firestore.FieldValue.serverTimestamp()
    });
  });
}

```

Figure 4.15: Create Stripe Account Function

```

export const createStripeCustomer = functions.auth.user().onCreate(async (snap, context) => {
  console.log('snap: ', snap);
  const customer = await stripe.customers.create({
    email: snap.email
  });

  const batch = admin.firestore().batch();

  const userRef = admin.firestore().collection('users').doc(snap.uid);
  const cusRef = admin.firestore().collection('customers').doc(customer.id);

  batch.update(userRef, { customerId: customer.id });
  batch.set(cusRef, { user: snap.uid });

  return batch.commit();
});

```

Figure 4.16: Navigate by Role Method

```

// function to go to the correct page depending on user role
navigateByRole(role) {
  if (role === 'BUYER') {
    this.router.navigateByUrl('/buyer');
  } else if (role === 'SELLER') {
    this.router.navigateByUrl('/seller');
  }
}

```


Figure 4.17: User Registration

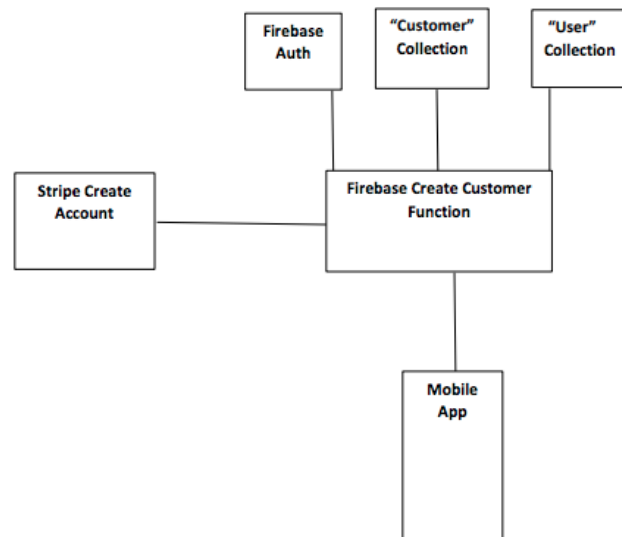


Figure 4.18: Buyer List Page

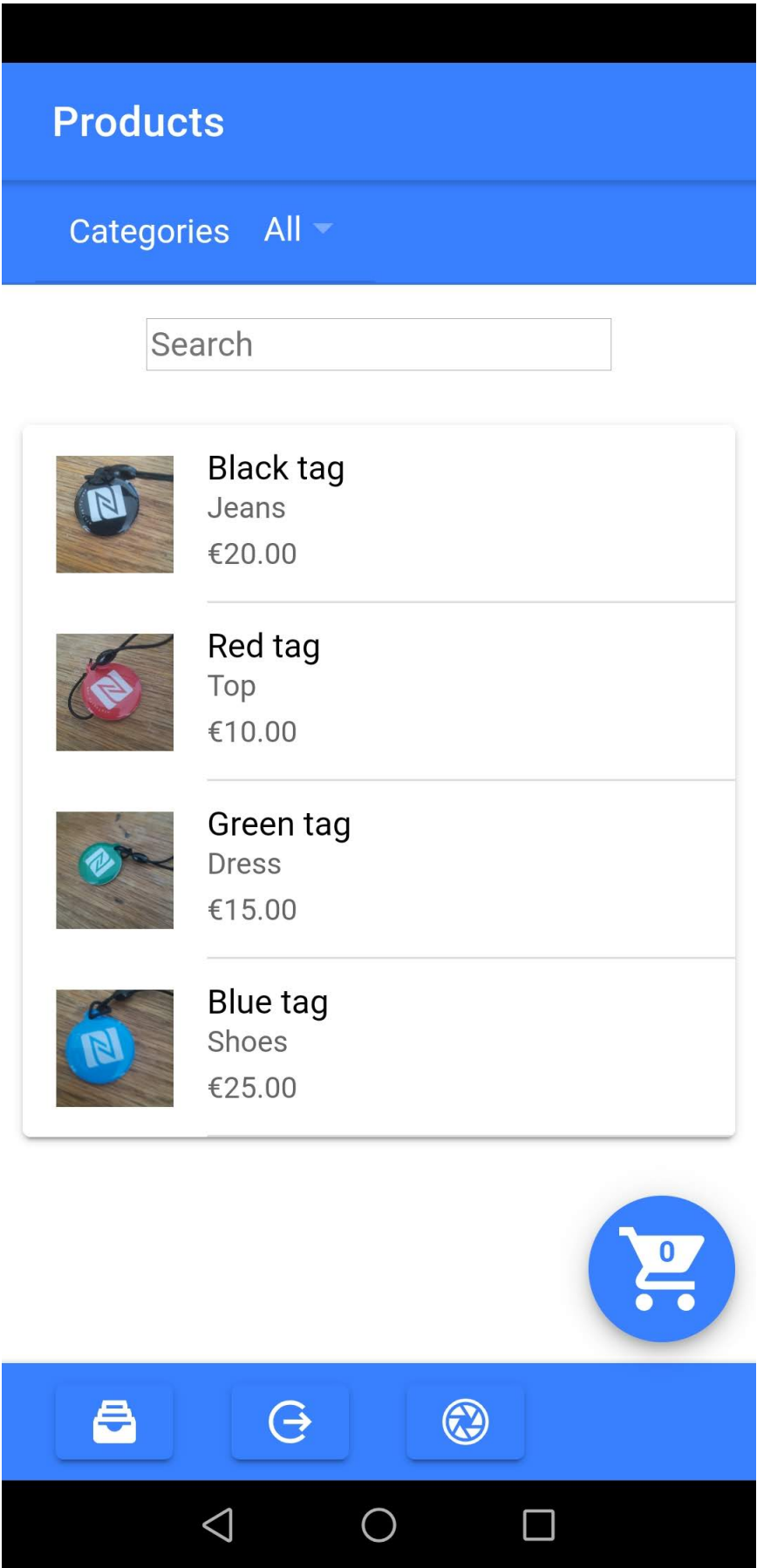


Figure 4.19: List all items

```
<ion-item *ngFor="let p of products | async | filter:filterProducts" [routerLink]="[p.id]">
```

Figure 4.20: Get all Products

```
getAllProducts() {  
  return this.db.collection('products').snapshotChanges().pipe(  
    map(actions => actions.map(a => {  
      const data = a.payload.doc.data();  
      const id = a.payload.doc.id;  
      return { id, ...data };  
    })))  
}
```

Figure 4.21: Pipe for Filtering

```
transform(value: any, searchText: string): any {  
  
  if(!value || !searchText){  
    console.log("Value 1", value)  
    return value;  
  }  
  
  return value.filter(product =>  
    product.categoryControl.toLowerCase().indexOf(searchText.toLowerCase()) !== -1 ||  
    product.name.toLowerCase().indexOf(searchText.toLowerCase()) !== -1);  
}
```

Figure 4.22: Cart Modal Page

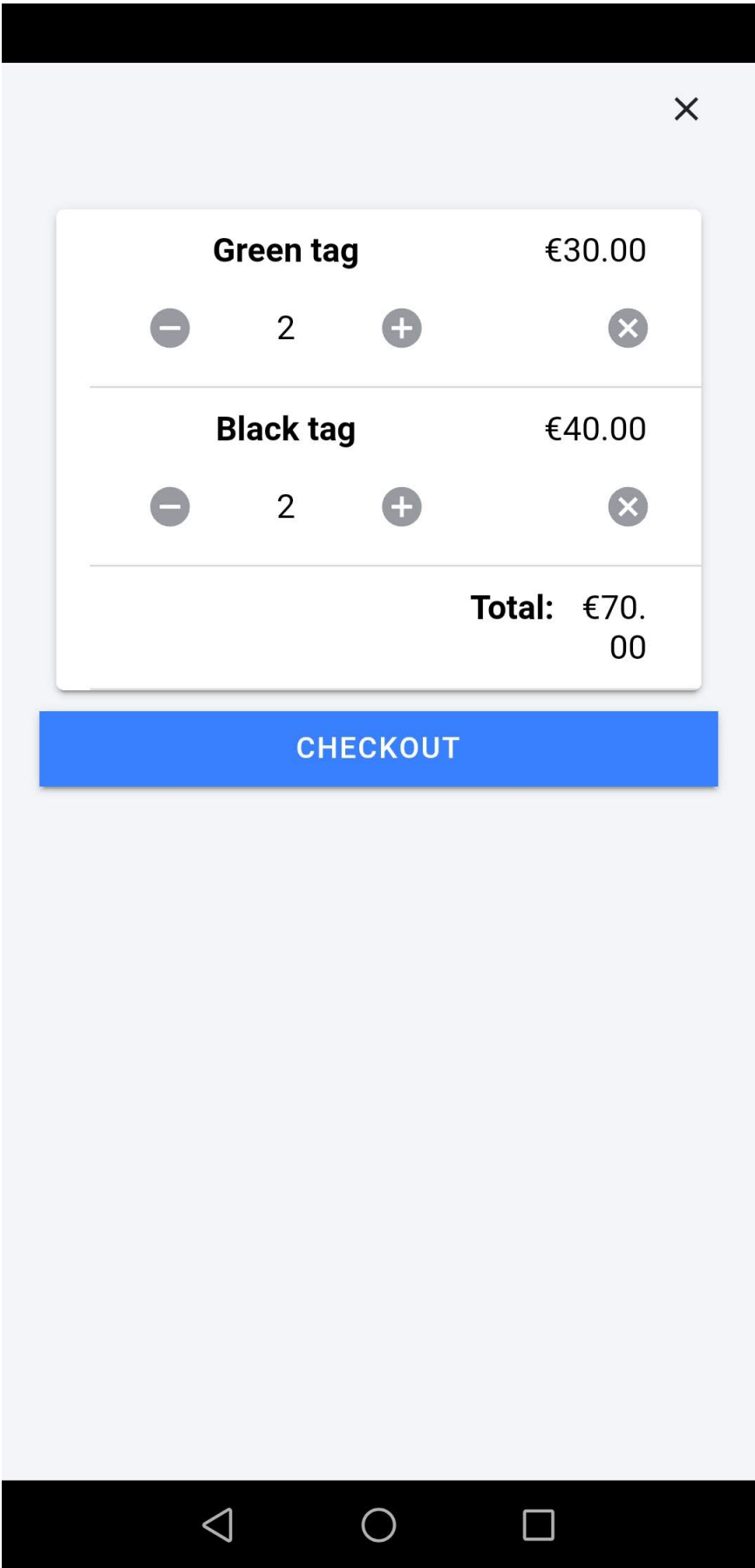


Figure 4.23: Add Product To Cart

```

addProduct(product) {
  product.creator = this.afAuth.auth.currentUser.uid;
  const imageData = product.img;
  delete product.image;

  let documentId = null;
  let storageRef: AngularFireStorageReference = null;

  return this.db.collection('products').add(product).then(ref => {
    console.log('ref: ', ref);
    documentId = ref.id;
    storageRef = this.storage.ref(`products/${documentId}`);
    const uploadTask = storageRef.putString(imageData, 'base64', { contentType: 'image/png' });
    return uploadTask;
  }).then(task => {
    console.log('task: ', task);
    return storageRef.getDownloadURL().toPromise();
  }).then(imageUrl => {
    console.log('got url: ', imageUrl);
    return this.db.doc(`products/${documentId}`).update({ img: imageUrl });
  });
}

```

Figure 4.24: Decrease Item count in Cart

```

decreaseProduct(product) {
  for (let [index, p] of this.cart.entries()) {
    if (p.id === product.id) {
      p.amount -= 1;
      if (p.amount == 0) {
        this.cart.splice(index, 1);
      }
    }
  }
  this.cartItems.next(this.cart);
  this.cartItemCount.next(this.cartItemCount.value - 1);
}

```

Figure 4.25: Remove Product from Cart

```

removeProduct(product) {
  for (let [index, p] of this.cart.entries()) {
    if (p.id === product.id) {
      this.cartItemCount.next(this.cartItemCount.value - p.amount);
      this.cart.splice(index, 1);
    }
  }
  this.cartItems.next(this.cart);
}

```

Figure 4.26: Get Cart Item Count

```
getItemCount(id) {  
  for (let [index, p] of this.cart.entries()) {  
    if (p.id === id) {  
      return p.amount;  
    }  
  }  
  return 0;  
}
```

Figure 4.27: Checkout Page

A mobile checkout form is displayed on a screen. The form is a light gray rounded rectangle with a close button (X) in the top right corner. It contains five text input fields with light blue backgrounds and rounded corners, each preceded by a label: 'Name', 'Street', 'City', 'Eircode/Zip', and 'Country'. Below these fields is a section for card payment, containing a card icon, the label 'Card number', and the label 'MM / YY'. At the bottom of the form is a large blue button with the text 'BUY NOW FOR \$70' in white. The entire form is set against a light gray background, and the screen has a black top and bottom bar.

×


Name

Street

City

Eircode/Zip

Country

 Card number MM / YY

BUY NOW FOR \$70

Figure 4.28: Checkout Form

```
this.dataForm = this.fb.group({
  name: ['', Validators.required],
  eircode: [''],
  street: ['', Validators.required],
  city: ['', Validators.required],
  country: ['', Validators.required]
});
```

Figure 4.29: Payment Intent Function

```
// payment intent
export const startPaymentIntent = functions.https.onCall(
  async(data, context) => {
    console.log('called: ', data);
    const user = await admin.firestore().collection('users').doc(data.userId).get();
    const userData = user.data();
    console.log('the user: ', userData);

    if (userData) {
      const intent = await stripe.paymentIntents.create({
        amount: data.amount,
        currency: 'EUR',
        customer: userData.customerId
      });

      await admin.firestore().collection('orders').doc(intent.id).set({
        items: data.items,
        status: 'Waiting for payment',
        amount: data.amount / 100,
        customerId: userData.customerId,
        userId: data.userId
      });

      return intent;
    } else {
      return false;
    }
  }
);
```


Figure 4.30: NFC Page

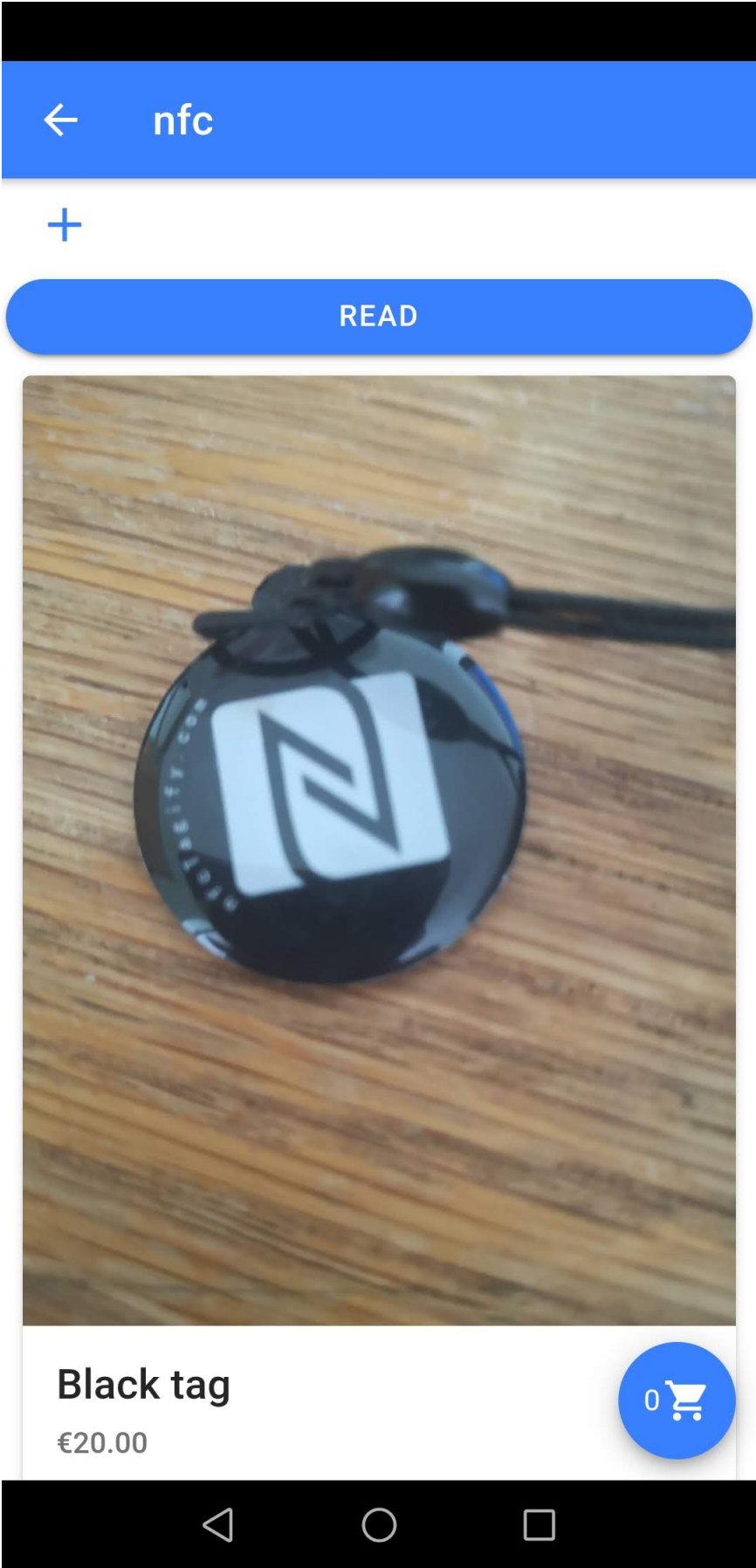


Figure 4.31: Set Ndef Listener

```
setNdefListener(): Promise<void> {  
  return new Promise<void>((resolve, reject) => {  
    this.nfc.enabled()  
    .then(() => {  
      if (!this.existingObservable) {  
        this.ndefEventObservable = this.nfc.addNdefListener();  
        this.existingObservable = true;  
        resolve();  
      } else {  
        resolve();  
      }  
    })  
    .catch(() => {  
      this.existingObservable = false;  
      reject(new Error());  
    });  
  });  
}
```

Figure 4.32: Subscribe to Ndef Event

```
private setNdefSubscription(): Promise<void> {  
  return new Promise<void>((resolve) => {  
    this.nfcSubscription = this.ndefEventObservable.subscribe((event) => {  
      this.onNdefEvent(event);  
    });  
    resolve();  
  });  
}
```

Figure 4.33: Ndef Event

```
private onNdefEvent(event) {  
  this.listenAlert.dismiss();  
  
  this.db.firestore.collection('products')  
    .where('tagid', '=', this.nfc.bytesToHexString(event.tag.id))  
    .get()  
    .then(querySnapshot => {  
      querySnapshot.forEach(doc => { this.id = doc.id; })  
      this.productService.getOneProduct(this.id).subscribe(res => {  
        this.product = res;  
        this.product.id = this.id;  
        this.amount = this.cartService.getItemCount(this.id);  
        console.log('tag id', this.product.tagid);  
  
        if(this.product.tagid == this.nfc.bytesToHexString(event.tag.id)){  
          }  
        else {  
          this.presentAlert("Incorrect tag read");  
        }  
      });  
    });  
}
```

Figure 4.34: Orders Page

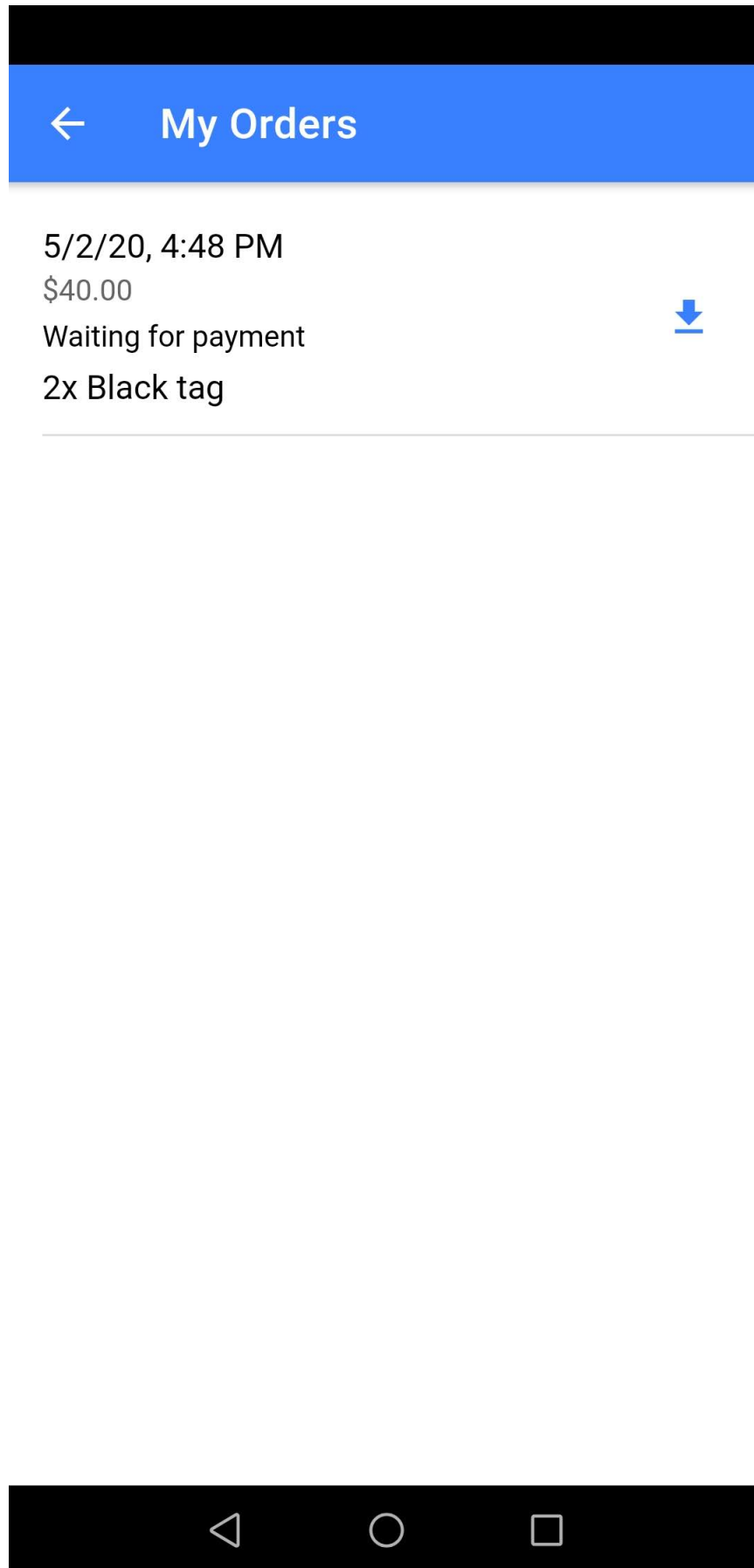


Figure 4.35: Get Customer Orders

```
export const getCustomerOrders = functions.https.onCall(  
  async (data, context) => {  
    const user = await admin  
      .firestore()  
      .collection('users')  
      .doc(data.userId)  
      .get();  
    const userData = user.data();  
    if (userData) {  
      return stripe.paymentIntents.list({ customer: userData.customerId });  
    } else {  
      return null;  
    }  
  }  
);
```

Figure 4.36: Open Invoice

```
openInvoice(item) {  
  this.productService.getOrderData(item.id).subscribe(res => {  
    console.log('my order: ', res);  
    const browser = this.iab.create(res['invoice'], '_system');  
  });  
}
```

Figure 4.37: Seller Page

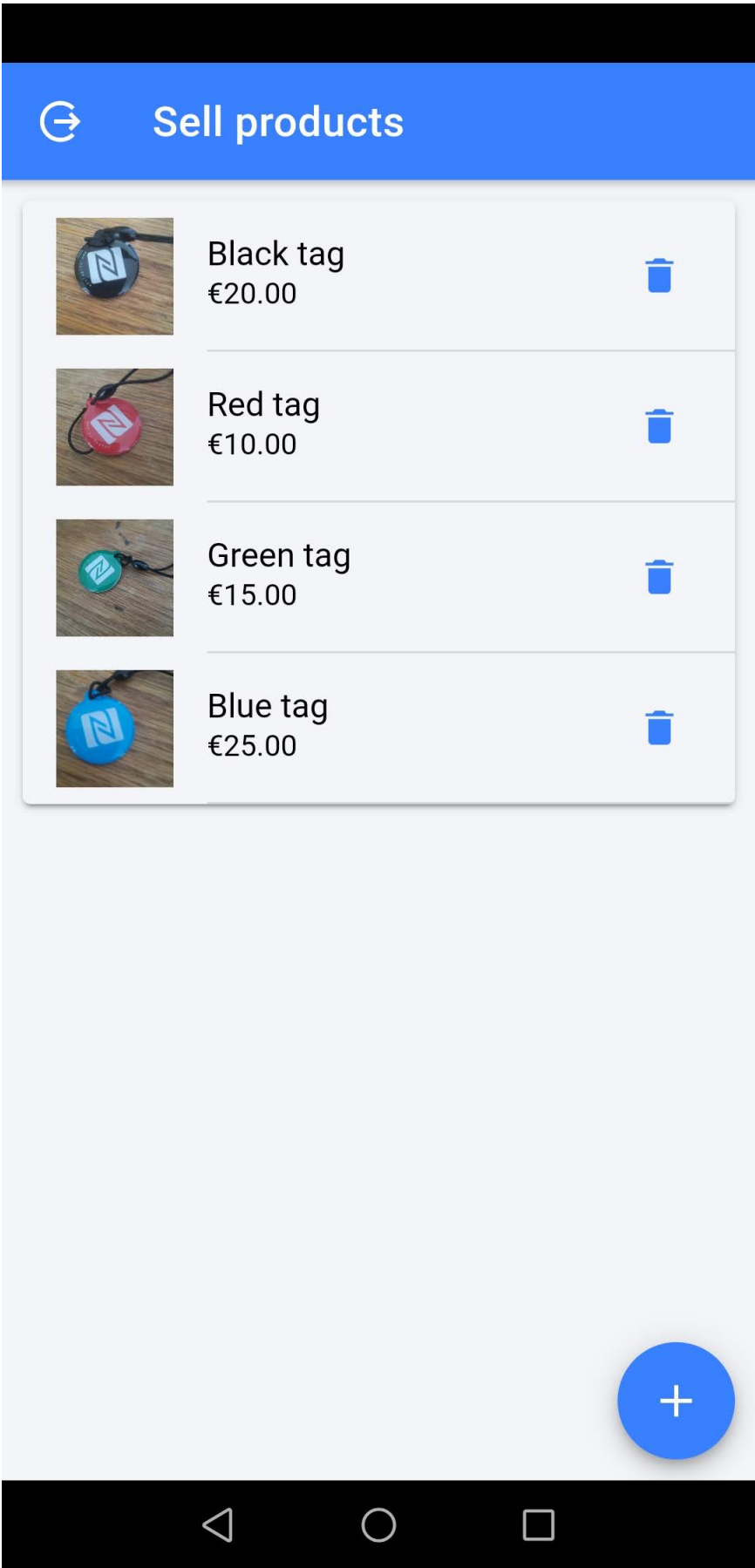


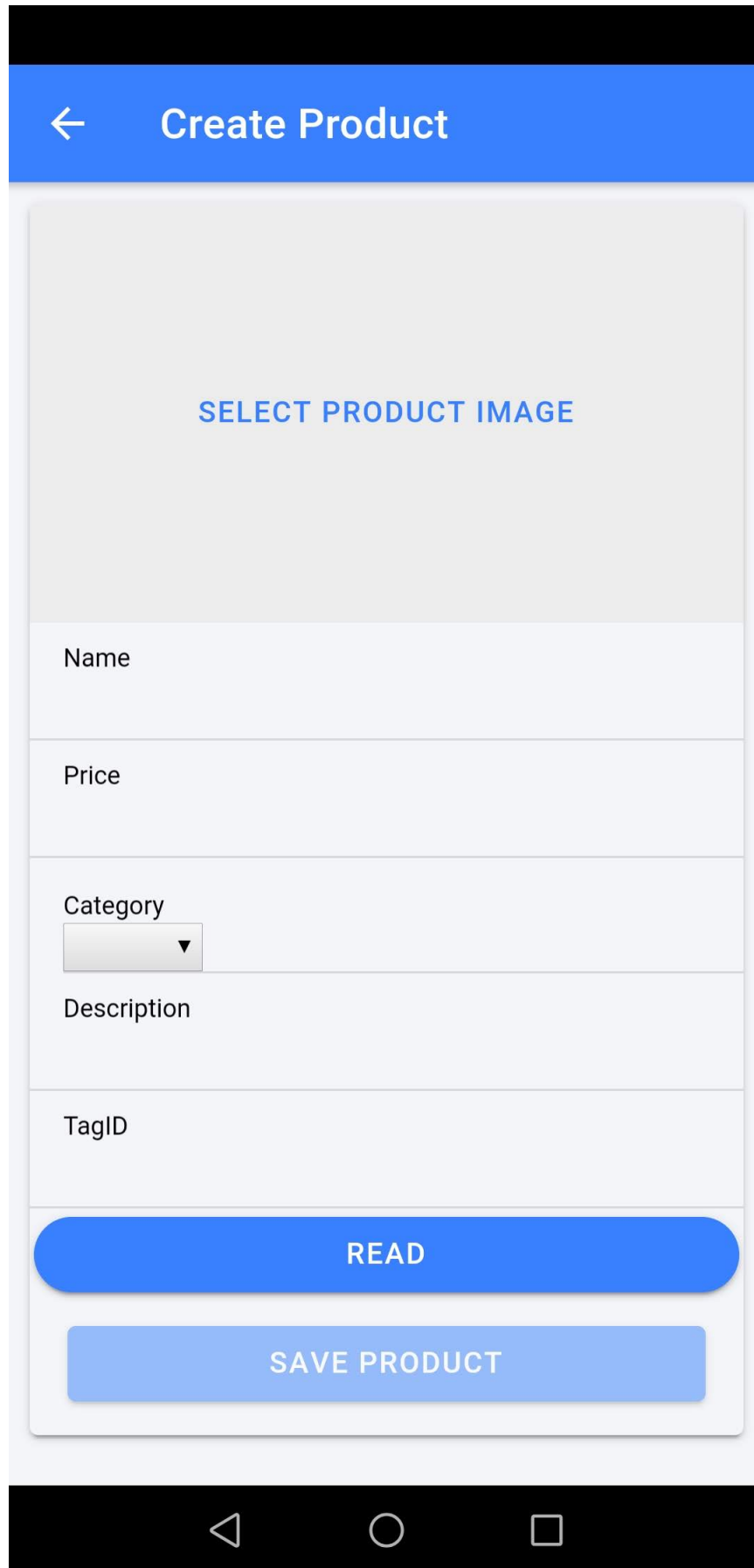
Figure 4.38: Gets Seller Products

```
getSellerProducts() {  
  const id = this.afAuth.auth.currentUser.uid;  
  
  return this.db.collection('products', ref => ref.where('creator', '==', id)).snapshotChanges().pipe(  
    map(actions => actions.map(a => {  
      const data = a.payload.doc.data();  
      const id = a.payload.doc.id;  
      return { id, ...data };  
    })))  
}
```

Figure 4.39: Delete a Product

```
//delete product  
deleteProduct(id) {  
  this.db.doc(`products/${id}`).delete();  
  this.storage.ref(`products/${id}`).delete().subscribe(res => {});  
}
```

Figure 4.40: Create New Product Page



← Create Product

SELECT PRODUCT IMAGE

Name

Price

Category

Description

TagID

READ

SAVE PRODUCT

Figure 4.41: Product Form

```

this.productForm = this.fb.group({
  name: ['', Validators.required],
  price: ['', Validators.required],
  categoryControl: [''],
  desc: '',
  category: '',
  tagid: ['', Validators.required],
  img: ''
})

```

Figure 4.42: Gets Image from Camera

```

selectImage() {
  const options: CameraOptions = {
    quality: 70,
    destinationType: 0,
    mediaType: 0,
    sourceType: 1,
    correctOrientation: true
  }

  this.camera.getPicture(options).then(data => {
    console.log(data);
    this.productImageBase64 = 'data:image/jpeg;base64,' + data;
    this.productForm.patchValue({ img: data })
  });
}

```

Figure 4.43: Read NFC tag

```

// Read tag
readNFC() {
  this.nfc.addNdefListener(() => {
    this.presentAlert('ok');
  }, (err) => {
    this.presentAlert('error' + err);
  }).subscribe((event) => {
    console.log(event);
    console.log(JSON.stringify(event));

    this.productForm.patchValue({ tagid: this.nfc.bytesToHexString(event.tag.id)});
  });
}

```

Chapter 5

System Evaluation

5.1 Reflection on Objectives

The goal of this project was to create an application that functions as an e-commerce marketplace in which users can both buy and sell product with ease. As with any similar application or website, user authentication and payment security are paramount and considerable research was undertaken in these areas regarding how other e-commerce businesses approach these issues. We also wanted to implement NFC technology in a way that goes a step beyond its very common usages such as opening links to website or connecting to Bluetooth devices. We wanted a common design theme and user interface throughout the application that is modern and user friendly.

5.1.1 Secure Payment and Authentication

As with any application that deals with online payments and sensitive user data, security and authentication are a major design factor. With the use of Stripe and Firebase Authentication, these concerns were addressed in a modern and extremely secure way. Industry level authentication standards OAuth 2.0 and OpenID Connect are used by Firebase to keep sensitive information secure.

As all payment is handled in Stripe, Payment Card Industry Data Security Standards (PCI DSS) are adhered to and the process is secure and safe for end users.

5.1.2 NFC

Although NFC tags are quite limited in functionality, there are a few other features which could have been implemented into the project. The idea of

adding a “lock” to a tag which is then unlocked when the user interacts with the tag and purchases the product was discussed but ultimately, we could not find a method of doing this that suited the project. The use of identifying the tag’s unique ID and storing this to the database turned out to be the most straight forward option both for user experience and for development.

5.1.3 Design Elements

With the use of Ionic styling elements, we are generally happy with the layout and look of the application. The icons, forms and colour schemes used give the application a minimalistic but approachable and user-friendly interface.

5.1.4 Testing

Only basic user interface functionality tests such as page navigation and login were performed on the application with the use of Selenium but more in-depth features that had to be tested on hardware were tested with Kobotin. Kobotin was extremely helpful while testing the application on the hardware device but this had to be limited as there was free tier limits that can turn into a hefty price plan. Looking back, I could have researched and identified some other mobile testing frameworks, but I went with Kobotin as it was highly rated online.

5.2 Further Development

As this was an e-commerce application, there is a wide range of additional features that could be added which are commonly found in similar applications which leaves room for an extensive amount of future development. These include more robust product filtering such as “new in” products, best sellers and user recommended products.

Although it is a relatively simple change to make, the addition of extra sign-in capabilities would be something to look at in future development. Firebase Authentication supports platforms such as Twitter, Facebook and GMail which can be used. However, there may be a slight problem with creating a Stripe account for these authentication methods.

Some basic machine learning functionality such as Google’s Machine Learning Kit would be a welcome addition to the project. It could be used to recommend products to customers based on previous purchases, for example.

Delivery of products purchased by customers would be another important decision to make. An Post could be used as well as private delivery companies

such as DPD, Nightline and Fastway Couriers. Local delivery companies such as What's for Dinner and Dial and Deliver could also be looked at. The possibility of setting up a specialised delivery service for each individual market would also be an interesting idea to explore.

5.2.1 Scalability

The Firebase Blaze plan was implemented as Firebase Functions were needed for the project. As this project was only for demonstration purposes and not live, the free-tier data limits were not exceeded. However, if the application was used in a real-world setting, database and storage maintenance costs would need to be factored in. These would amount to €5 for every 5 gigabytes of database data and €0.026 for every 5 gigabytes of storage used. [33]

For Stripe, test data was used for development, but pricing would also have to be considered for taking live payments. This amounts to 1.4% of purchase cost plus €0.25 for each transaction.

Although these pricing schemes are relatively cheap, pricing was not a factor we discussed when they were chosen for the project. In industry, maintenance costs would be one of the first considerations when developing an e-commerce application.

Chapter 6

Conclusion

In conclusion, I feel that overall, the project went well, and many of the goals set out were achieved. Secure authentication and payment were huge considerations and they were implemented well in the project. Authentication could be expanded into allowing Twitter, Facebook, etc. For a better user experience, however. Stripe uses extremely secure encryption and transport protocols which makes me confident that payments are made securely in the app. The separation of buyer and seller functionality was also done satisfactorily, and a smooth user experience is provided. The app isn't overly complicated which makes it usable by most people. I feel that the database was designed well with the separation of data into different database collections depending on their need and functionality. Although the NFC functionality is quite simple, it works well. I would have liked to explore NFC more and perhaps have figured out the locking/unlocking of tags that was discussed between group members and our project supervisor. There are other uses for NFC that we could have researched further in hindsight.

Chapter 7

Appendices

GitHub Repository is: <https://github.com/kodama96/FinalYearProject>

Bibliography

- [1] “<https://ecommerceguide.com/guides/what-is-ecommerce/>.”
- [2] “<https://www.sciencedirect.com/science/article/abs/pii/S0305048300000219>.”
- [3] “<https://www.statista.com/outlook/243/140/ecommerce/ireland>.”
- [4] “<https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/>.”
- [5] “https://medium.com/@the_manifest/android-vs-ios-which-platform-to-build-your-app-for-first-22ea8996abe1.”
- [6] “<https://medium.com/appcoda-tutorials/building-nfc-product-scanner-ios-app-with-corenfc-alfian-losari-5da0365bcde5>.”
- [7] “<https://sci-hub.im/https://www.sciencedirect.com/science/article/pii/S0164121212000532>.”
- [8] “<https://www.selenium.dev/documentation/en/>.”
- [9] “<https://docs.kobiton.com/basic/knowledge-base/>.”
- [10] “<https://www.nfcporter.com/>.”
- [11] “<https://www.androidpit.com/what-is-nfc>.”
- [12] “<https://support.google.com/pay/?hl=en&topic=7625138>.”
- [13] “<https://www.apple.com/ie/apple-pay/>.”
- [14] “<https://personalbanking.bankofireland.com/bank/current-accounts/contactless/about-contactless/>.”
- [15] “<https://aib.ie/our-products/current-accounts/contactless-payments>.”
- [16] “<https://www.electronics-notes.com/articles/connectivity/nfc-near-field-communication/tags-types.php>.”

- [17] “<https://code.visualstudio.com/docs>.”
- [18] “<https://insights.stackoverflow.com/survey/2019>.”
- [19] “<https://developer.android.com/docs>.”
- [20] “<https://www.typescriptlang.org/docs/home.html>.”
- [21] “<https://cordova.apache.org/docs/en/latest/>.”
- [22] “<https://angular.io/docs>.”
- [23] “https://www.w3schools.com/whatis/whatis_npm.asp.”
- [24] “<https://ionicframework.com/docs>.”
- [25] “<https://www.fundera.com/blog/stripe-reviews>.”
- [26] “<https://github.com/chariotsolutions/phonegap-nfc>.”
- [27] “<https://howtofirebase.com/what-is-firebase-fcb8614ba442>.”
- [28] “<https://firebase.google.com/docs/auth>.”
- [29] “<https://firebase.google.com/docs/database>.”
- [30] “<https://firebase.google.com/docs/firestore>.”
- [31] “<https://firebase.google.com/docs/storage/web/start>.”
- [32] “<https://firebase.google.com/docs/functions>.”
- [33] “<https://firebase.google.com/pricing>.”