National College of Ireland

Bachelor of Science (Honours) in Computing

Cyber Security

2020/2021

Fouad Animashaun

X16300351

X16300351@student.ncirl.ie

Nf_Kicks

Technical Report

TABLE OF CONTENTS

Executive Summary

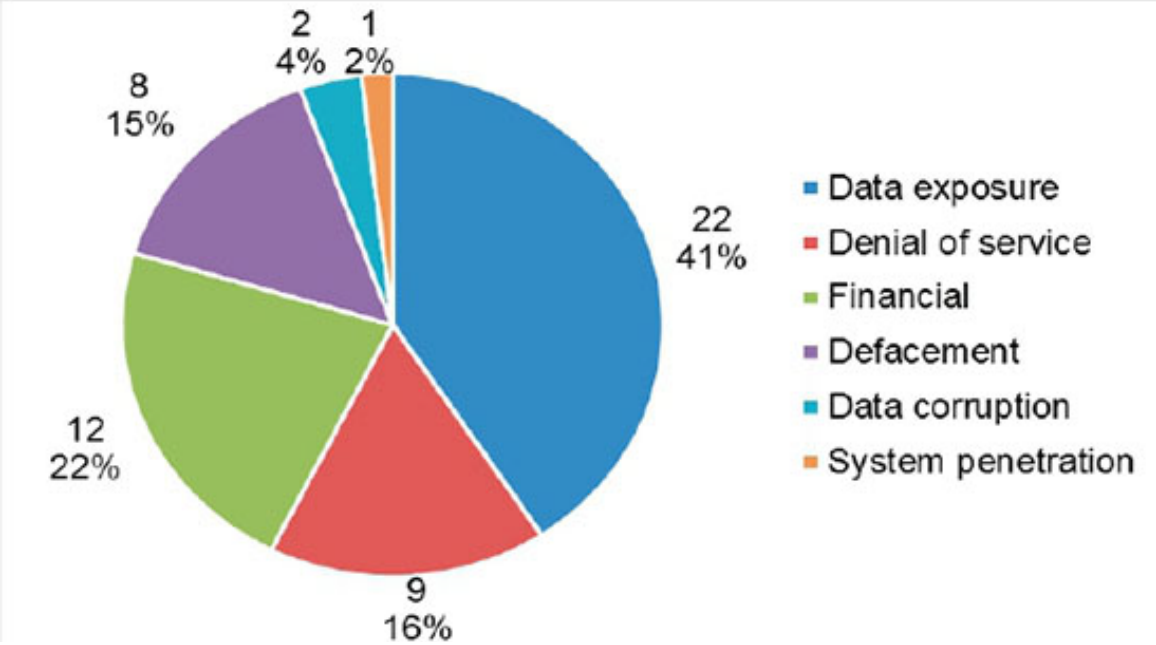
Today's e-commerce applications are not designed with security in mind; the primary risks to e-commerce and customer privacy include data exposure, denial of service attacks, brute force attacks, reverse engineering, and security misconfigurations. As well, companies must adhere to regulatory requirements, such as the General Data Protection Regulation (GDPR) and the Payment Card Industry Data Security Standard while processing consumer credit card details, to ensure that customer information does not end up stolen by an attacker. These issues can start to make the cost of doing business without security built into applications prohibitively costly, which is why I decided to make this report a security evaluation that will detail how to implement proper security measures in today's e-commerce applications by addressing the difficulties associated with developing a security-first e-commerce application, including lost productivity, sensitive data leakage, and regulatory compliance. The report will then proceed to provide answers to the issues of security in e-commerce through an analysis of the Nf_Kicks project, which will cover security features and the problems said features are addressing concerning e-commerce security.

1.0 Introduction

1.1. Background

Emerging e-commerce markets in sub-Saharan Africa are being driven by mobile technologies. This is thanks to the region's continued innovation in payments and telecommunications infrastructure, which has resulted in an 18 percent rise in the number of online shoppers since 2014, according to The United Nations Conference on Trade and Development (Masekesa, 2020). The e-commerce industry in Africa is predicted to reach 75 billion USD by 2025, with leading players such as South Africa, Nigeria, and Kenya responsible for more than half of all online customers in the region. Given that the global pandemic has only exacerbated the desire to shop online, this has led to both consumers and brick-and-mortar retailers preferring to arrange pick-ups or have goods shipped to reduce the spread of the virus, further increasing the use of mobile technologies in e-commerce.

However, as the rise in e-commerce in these developing markets continues, more focus must be given to cyber-security, as this could lead to an increase in confidence among online shoppers, as there is a reluctance to provide credit card information and personal data in these regions. Unfortunately, only 10% of African companies have adequate cyber-attack protection, implying that the vast majority of consumer-facing businesses are incapable of adequately protecting their customers and themselves from existing cyber-attacks (Singh, 2019). The following picture illustrates the percentage distribution of 54 incidents across six forms of common attacks in South Africa (Van Niekerk, 2017).



Data exposure is the attack that sticks out here, though it is extremely widespread. Such attacks are rarely ever reported or disclosed. This can include things such as theft of trade secrets or sensitive information leakage. This can result in the unauthorized access of customer information resulting in a loss of trust from customers or the disclosure of a private company document to a competitor. In 2016, a software developer found customer data exposure on the eThekwini Metropolitan Municipality platform, allowing him to access data related to registered users on the site by merely modifying a portion of the URL (Staff, 2016).

While such attacks can seem straightforward in practice, emerging markets such as South Africa see cybersecurity as a privilege, with many companies reporting that their cybersecurity budgets are less than 1% (Kshetri, 2019). This is usually due to a lack of investment, training, and an executive-level CISO (chief information security officer).

Furthermore, over 21 African countries have adopted data protection and privacy legislation, rendering non-compliance with laws and regulations very expensive. This is quote details how fines are handled by the Nigerian Data Protection Regulation in relation to data breaches:

"*According to the NDPR, where a data breach is reported and the data controller is found guilty, they are liable to a payment of a fine of 1% of the annual gross revenue of the preceding year or payment of the sum of N2,000,000 (whichever is greater) where the data controller deals with less than 10,000 data subject. On the other hand, where it is a controller of more than 10,000 data subjects, they are liable to a fine of 2% of the annual gross revenue of the preceding year or a payment of the sum of N10,000,000 (whichever is greater).*" (Ololuo, 2020)

As a result of this regulation, Nigerian companies must be conscientious about protecting customer data and disclosing data breaches, as leaks will erode customer trust, whilst a

refusal to disclose breaches early results in a large fine, creating a catch-22 scenario for businesses who do not invest in cyber-security.

Cyberattacks in Africa's emerging markets are estimated to have cost 3.5 billion USD in total damages in 2017, with the worst cases to date occurring in the big three countries of South Africa, Kenya, and Nigeria. However, the consequences of not having proper security in place does not only include financial losses but also reputational damage for the businesses in these regions in particular, for there is a reluctance when it comes to placing trust in technology to handle financial information and personal data amongst consumers (Masekesa, 2020).

In summary, the problem that my application is attempting to address is how to implement the minimum amount of security in an e-commerce mobile application that does not leak sensitive customer information and retains trust by securing said information on the device and in transit to a database. The application must also abide by the latest and greatest in data protection regulations to ensure that customer data is being protected and secured in the hands of the application.

1.2. Aims

This project aims to develop a secure mobile application that cannot be exploited to jeopardize the application or a user of the application. This will be done by following a comprehensive road map for securing e-commerce mobile applications:

1. **Secure the device**

Ensure that the user who has chosen to install the application on either an Android or IOS device is neither rooted nor jailbroken, otherwise, the attacker can use software tools to avoid the in-app payment processor.

Customer sessions must timeout adequately to avoid an unauthorized user having access to the mobile device and the customer's session in the application.

2. **Content security**

The user's customer information must be encrypted to prevent confidential data from being disclosed. User information must only be accessible and modifiable by said user and none other.

Business-specific data should not be modifiable or deleted to prevent undesirable business data from being saved and accessible by other users of the application in any way unless the change was made by the developer of the project.

3. **Application security**

Implement a secure testing strategy with Unit tests to ensure that functionality in the application is working as intended with no false positives in place. Code hardening by encryption and obfuscation would help increase the difficulty of running vulnerabilities and render the code behind the program more difficult to reverse engineer. Application dependencies are to be updated to the latest versions for security vulnerabilities in newer versions are more likely to have been patched. Secure coding best practices will be followed with the use of code linting tools to patch application bugs and errors before they arrive and are shipped to app stores.

4. Identity & Access Security

Securing the user account is critical since it serves as the primary entry point to the application's core functionality and has access to all user data; therefore, account security must begin the moment the customer creates an account. For an attacker to easily gain access to an account, the password would have to be easy to guess. This is why we do the opposite and force the user to choose a password that is difficult to guess.

Additionally, the customer's password must not be saved in plain text, making the use of password hashing critical for obscuring the password in transit and storage.

1.3. Technology

Flutter and Dart are the primary technologies I've opted to use for this project. Flutter is a cross-platform mobile device development tool that allows developers to write code once and enable their application to run on both iOS and Android. React Native is an alternative to Flutter. My rationale for using Flutter over React Native is to speed up development time. Since my project is a security evaluation, focusing on creating a ton of code that is not relevant to security in some manner will not be deemed a reasonable use of the finite period of time available to finish a project. React Native slows progress by not providing adequate instructions for setting up the framework, while Flutter provides a comprehensive guide to getting the framework running using the Flutter command-line interface (CLI). Second, Flutter's compilation time is considerably shorter than that of React Native, giving it an advantage in terms of performance. This is due to the fact that React Native builds applications in JavaScript, whereas Flutter is only compiled to ARM and x86 native libraries.

The use of Flutter is a core building block to this project due to it being an e-commerce application where having your app run on both platforms is important for larger reach. Also, the Flutter framework, by default, encrypts outgoing information to Firebase with the use of Secure Socket Layer (SSL). Dart is the name of the programming language used to write mobile applications with Flutter. The security features of Dart are that Dart is single-threaded, which makes it less possible for Deadlock or a Race condition to arise in an application and Dart supports the use of dynamically typed and strongly typed language features, rendering it more type-safe than React Native’s JavaScript when writing software code.

Dart offers extra security by requiring developers to write code that adheres to linting rules while writing code to eliminate possible vulnerabilities and errors. A code linter is an automatic source code application that analyses code for programmatic and stylistic mistakes, linting rules are guidelines to abide by when writing code. This helps the programmer to write Dart code that is not only readable to others but also has a limited number of errors. The language also comes packaged with code obfuscation for the production build of the application to protect the functionality of the applications from attackers who wish to reverse engineer the app to discover secrets.

The database being used in this application is Firebase and this is the backend of the application that not only stores the application's business-related data but handles numerous login and registration options, namely Facebook Login and Google Sign in through the use of Firebase Authentication. Firebase Cloud Firestore is a NoSQL document database that handles CRUD operations on the database through the use of web requests. Although Firebase Cloud Firestore has its own security built-in as a result of the service being owned and audited by Google, there is an additional layer of security that could be added by the developer. This is called Firebase Security Rules, and it is server-side code written in JavaScript used to analyse web requests before CRUD operations can occur. Firebase Security Rules ensure data confidentiality, integrity, and availability by determining if an individual has access to certain resources in the database before CRUD operations can take place; if authorization fails, the request will be refused access to said resource. Furthermore, Security Rules will search for special key-value pairs in data submitted to the database for create and update requests. If said key values pair didn't exist, the request would be rejected.

Stripe Payments is a credit card payment processing platform. This allows retailers to safely and securely conduct purchases on consumer bank accounts with a credit or debit card payment. Stripe has been audited and approved as a PCI Service Provider Level 1 by PCI-certified auditors. This accreditation is the most rigorous form of approval in the payments industry and signifies that the platform is capable of processing credit and debit card transactions without the details falling into the wrong hands. Stripe Payments Platform guarantees credit and debit card protection through encrypting card numbers with AES-256 and storing decryption keys in a secure location. Additionally, the Stripe payments API in Flutter requires all data sent to Stripe servers to be encrypted using HTTPS. This is critical because it makes it difficult for an attacker to read web requests with card information in plain text (Security at Stripe, 2021).

1.4. Structure

Provide a brief overview of the structure of the document and what is addressed in each section.

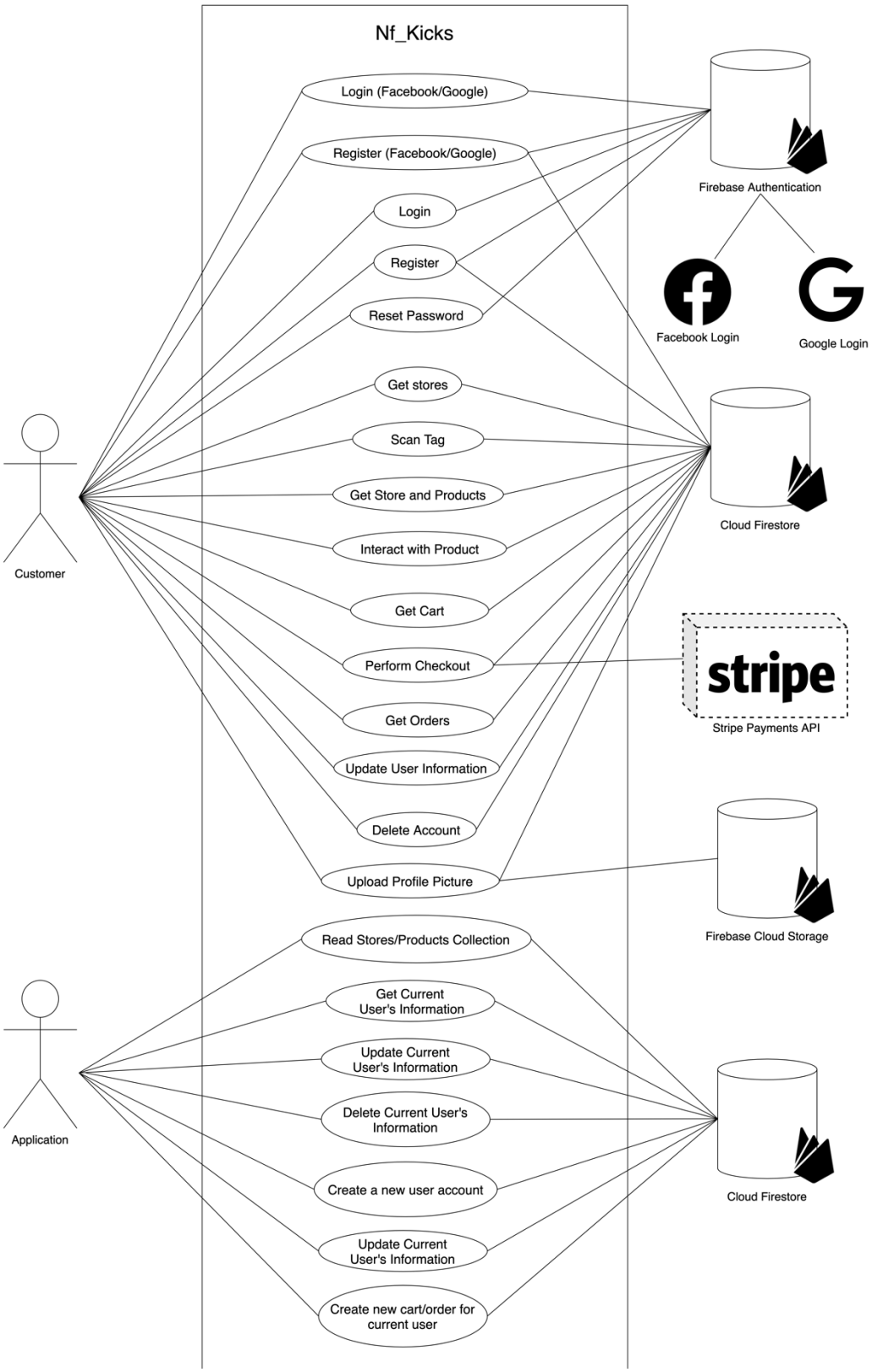
What should be here????

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Login (Facebook/Google)

2.1.1.2.1. Description & Priority

This requirement enables users to access their Nf_Kicks account from an external account provider API rather than remembering their credentials.

Priority: Low

2.1.1.2.2. Use Case

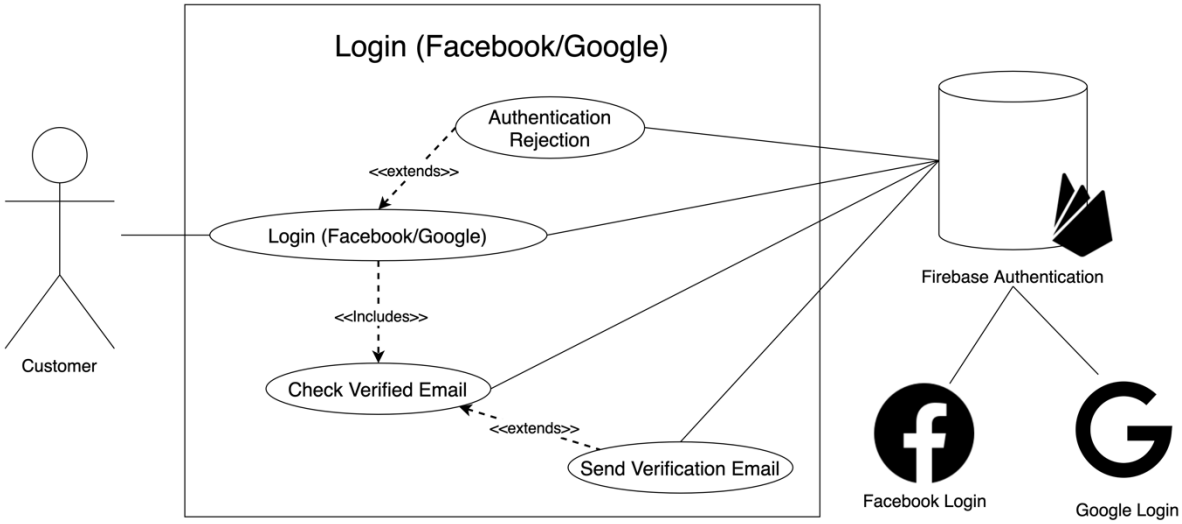
Scope

The scope of this use case is to provide the customer access to their Nf_Kicks account and validate if their email has been verified.

Description

This use case outlines the steps taken by the application to grant the customer access to their account.

Use Case Diagram



2.1.1.2.3. Flow Description

Precondition

The application is in the initialization phase.

Activation

This use case begins when a customer launches the application on their mobile device.

Main flow

1. The customer taps on the Facebook icon present on the screen.
2. The application identifies that the customer has pressed the Facebook icon.
3. The application sends a request to Firebase Authentication for a Facebook login window.
4. Firebase Authentication receives the request and sends a new request to Facebook's authentication API for a login window.

5. The Facebook API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application routes the customer to the Facebook login window.
8. The user provides their Facebook account credentials.
9. The user taps the Facebook login button.
10. The Facebook API identifies that the user has tapped the login button.
11. An authentication request is made to Facebook with the user's credentials.
12. The authentication request is successful.
13. The user is routed to a page from Facebook asking if the user is sure they would like to share information.
14. The user accepts.
15. The Facebook Login API sends user account data to Firebase Authentication.
16. Firebase Authentication uses this data to create a session for the user in the application.
17. The application now sends a request to Firebase Authentication to check if the user has verified their email address.
18. Firebase Authentication responds with information on if the user is verified.
19. The application has identified that the customer is indeed verified.
20. The application navigates the customer to the home screen.

Alternate flow

A1 : Chooses Google

1. The customer taps on the Google icon present on the screen.
2. The application identifies that the customer has pressed the Google icon.
3. The application sends a request to Firebase Authentication for a Google login window.
4. Firebase Authentication receives the request and sends a new request to Google's authentication API for a login window.
5. The Google API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application displays a login window
8. The user chooses either their current Google account or logins in with another.
9. The Google API identifies that the user has tapped their Google account logo.
10. An authentication request is made to Google with the user's credentials.
11. The authentication request is successful.
12. The Google Login API sends user account data to Firebase.
13. The use case continues at position 16 of the main flow.

Exceptional flow

E1 : User's email is not verified

1. The application identifies that the user has not verified their email.

2. The application sends a new verification email to the user's email.
3. The application routes the user to a screen asking them to check their email and verify their email to continue.

E2 : Login aborted

1. The customer taps on the Google icon present on the screen.
2. The application identifies that the customer has pressed the Google icon.
3. The application sends a request to Firebase Authentication for a Google login window.
4. Firebase Authentication receives the request and sends a new request to Google's authentication API for a login window.
5. The Google API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application displays a login window.
8. The user chooses to cancel.
9. The application identifies that the user has cancelled the user login.
10. The application navigates back to the login screen.

E3 : Device is Jailbroken or Rooted

1. The application scans the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E4 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

E5 : Firebase Authentication server is down

1. The user taps on the Facebook icon present on the screen.
2. The application identifies that the user has pressed the Facebook icon.
3. The application sends a request to Firebase Authentication for a Facebook login window.
4. The application identifies that the authentication server down.
5. The application displays a window to the user saying that an error has occurred and to try again later.

Termination

The application navigates to the login screen.

Post condition

The application waits for the user to perform an action.

2.1.1.3. Requirement 2: Register (Facebook/Google)

2.1.1.3.1. Description & Priority

This requirement enables users to create a Nf_Kicks account from an external account provider API, rather than creating and remembering a password.

Priority: Low

2.1.1.3.2. Use Case

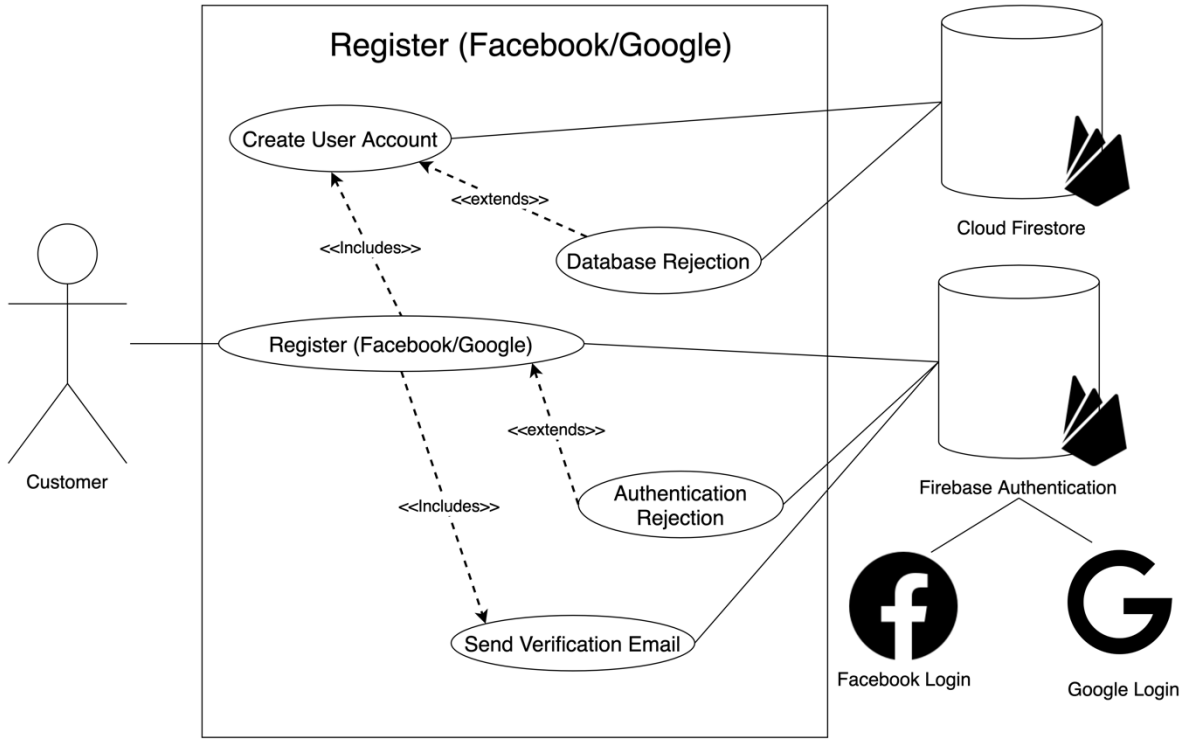
Scope

The scope of this use case is to enable the customer to create a Nf_Kicks account and sent a verification email.

Description

This use case describes the actions the application performs to create an account for the customer with the use of an external provider.

Use Case Diagram



2.1.1.3.3. Flow Description

Precondition

The application is on the registration screen.

Activation

This use case begins when a user chooses either one of the account providers.

Main flow

1. The customer taps on the Facebook icon present on the screen.
2. The application identifies that the customer has pressed the Facebook icons.
3. The application sends a request to Firebase Authentication for a Facebook login window.
4. Firebase Authentication receives the request and sends a new request to Facebook's authentication API for a login window.
5. The Facebook API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application routes the customer to the Facebook login window.
8. The user provides their Facebook account credentials.
9. The user taps the Facebook login button.
10. The Facebook API identifies that the user has tapped the login button.
11. An authentication request is made to Facebook with the user's credentials.
12. The authentication request is successful.
13. The user is routed to a page from Facebook asking if the user is sure they would like to share information.
14. The user accepts.
15. The Facebook Login API sends user account data to Firebase.
16. Firebase Authentication uses this data to create a session for the user in the application.
17. The application receives this user data and makes a request to Cloud Firestore to save it.
18. Firebase Cloud Firestore uses this data to create a record in the database that corresponds to the user.
19. The application now sends a request to Firebase Authentication to send a verification email to the user.
20. Firebase Authentication sends a verification email to the user.
21. The application navigates to a page that requires the user to check their email and login once verified.

Alternate flow

A1 : Chooses Google

1. The customer taps on the Google icon present on the screen.
2. The application identifies that the customer has pressed the Google icon.
3. The application sends a request to Firebase Authentication for a Google login window.
4. Firebase Authentication receives the request and sends a new request to Google's authentication API for a login window.
5. The Google API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application displays a login window.
8. The user chooses either their current Google account or logins in with another.

9. The Google API identifies that the user has tapped their Google account logo.
10. An authentication request is made to Google with the user's credentials.
11. The authentication request is successful.
12. The Google Login API sends user account data to Firebase.
13. The use case continues at position 16 of the main flow.

Exceptional flow

E1 : Registration aborted

1. The customer taps on the Google icon present on the screen.
2. The application identifies that the customer has pressed the Google icon.
3. The application sends a request to Firebase Authentication for a Google login window.
4. Firebase Authentication receives the request and sends a new request to Google's authentication API for a login window.
5. The Google API responds with code for said login window.
6. Firebase Authentication receives the response and sends it to the client.
7. The application displays a login window.
8. The user chooses to cancel.
9. The application identifies that the user has cancelled the user registration.
10. The application navigates back to the registration screen.

E2 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E3 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

E4 : Firebase Authentication server is down

1. The user taps on the Facebook icon present on the screen.
2. The application identifies that the user has pressed the Facebook icon.
3. The application sends a request to Firebase Authentication for a Facebook login window.
4. The application identifies that the authentication server down.
5. The application displays a window to the user saying that an error has occurred and to try again later.

Termination

The application navigates to the check email screen.

Post condition

The application waits for the user to perform an action.

2.1.1.4. Requirement 3: Login

2.1.1.4.1. Description & Priority

This requirement allows the user to enter their Nf_Kicks account using their default email address and password. To ensure account security, the use case performs validation on both the client and server sides.

Priority: High

2.1.1.4.2. Use Case

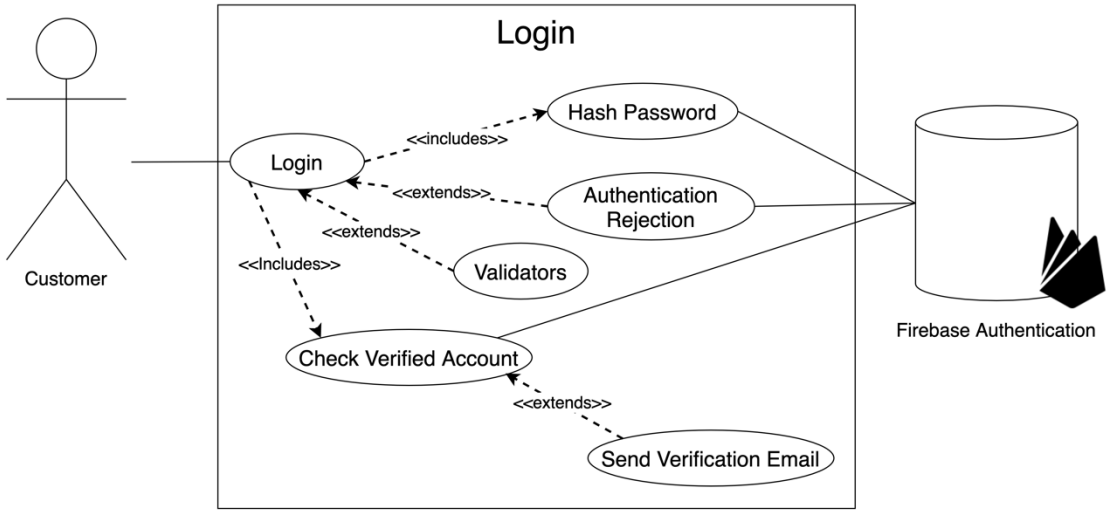
Scope

The scope of this use case is to provide the customer access to their Nf_Kicks account and perform validation on their credentials.

Description

This use case describes the actions the application performs to provide the customer access to their account.

Use Case Diagram



2.1.1.4.3. Flow Description

Precondition

The application is in the initialization phase.

Activation

This use case begins when a customer launches the application on their mobile device.

Main flow

1. The user enters their valid email address and valid password to assigned fields.
2. The application using form-field validation detects that the user has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application hashes the user's password.
6. The application sends the user account credentials to Firebase Authentication.
7. Firebase Authentication checks if the email and password are correct.
8. Firebase Authentication checks if the customer has verified their account.
9. Firebase Authentication sends a response back to the application that allows the customer to login.
10. The application receives a request from Firebase Authentication.
11. The application logs the user into their account.

Alternate flow

A1 : Form Field Validation Failure on Email Field

1. The user provides an invalid email address.
2. The form-field validators detect that the email provided is incorrect.
3. The use case continues at position 1 of the main flow.

A2 : Form Field Validation Failure on Password Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided is incorrect.
3. The use case continues at position 1 of the main flow.

A3 : Wrong Password

1. The user provides an email and the wrong password to assigned fields.
2. The application using form-field validation detects that the customer has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application hashes the user's password.
6. The application sends the user account credentials to Firebase Authentication.
7. Firebase Authentication identifies that the password provided by the user is incorrect.
8. Firebase Authentication response about the error.
9. The application receives this response and displays a window saying that one of the credentials provided is incorrect.

10. The use case continues at position 1 of the main flow.

A4 : User Does Not Exist

1. The user provides an email and the password to assigned fields.
2. The application using form-field validation detects that the customer has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application hashes the user's password.
6. The application sends the user account credentials to Firebase Authentication.
7. Firebase Authentication identifies that the user does not exist.
8. Firebase Authentication response about the error.
9. The application receives this response and displays a window saying that the user does not exist.
10. The use case continues at position 1 of the main flow.

A5 : User Account is Disabled

1. The user provides a valid email and a valid password to assigned fields.
2. The application using form-field validation detects that the customer has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application hashes the user's password.
6. The application sends the user account credentials to Firebase Authentication.
7. Firebase Authentication identifies that the user's account is disabled.
8. Firebase Authentication response about the error.
9. The application receives this response and displays a window saying that the user's account is disabled.
10. The use case continues at position 1 of the main flow.

Exceptional flow

E1 : User's email is not verified

1. The application identifies that the user has not verified their email.
2. The application sends a new verification email to the user's email.
3. The application routes the user to a screen asking them to check their email and verify their email to continue.

E2 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.

3. The application routes the user to a screen asking them to have an un-rooted device.

E3 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

E4 : Firebase Authentication server is down

1. The user enters their valid email address and valid password to assigned fields.
2. The application using form-field validation detects that the user has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application hashes the user's password.
6. The application sends the user account credentials to Firebase Authentication.
7. The application identifies that the authentication server down.
8. The application displays a window to the user saying that an error has occurred and to try again later.

Termination

The application navigates to the home page.

Post condition

The application waits for the user to perform an action.

2.1.1.5. Requirement 4: Register

2.1.1.5.1. Description & Priority

This requirement allows the customer to create a Nf_Kicks account using their preferred email address and password. To ensure account security, the use case performs validation on both the client and server sides.

Priority: High

2.1.1.5.2. Use Case

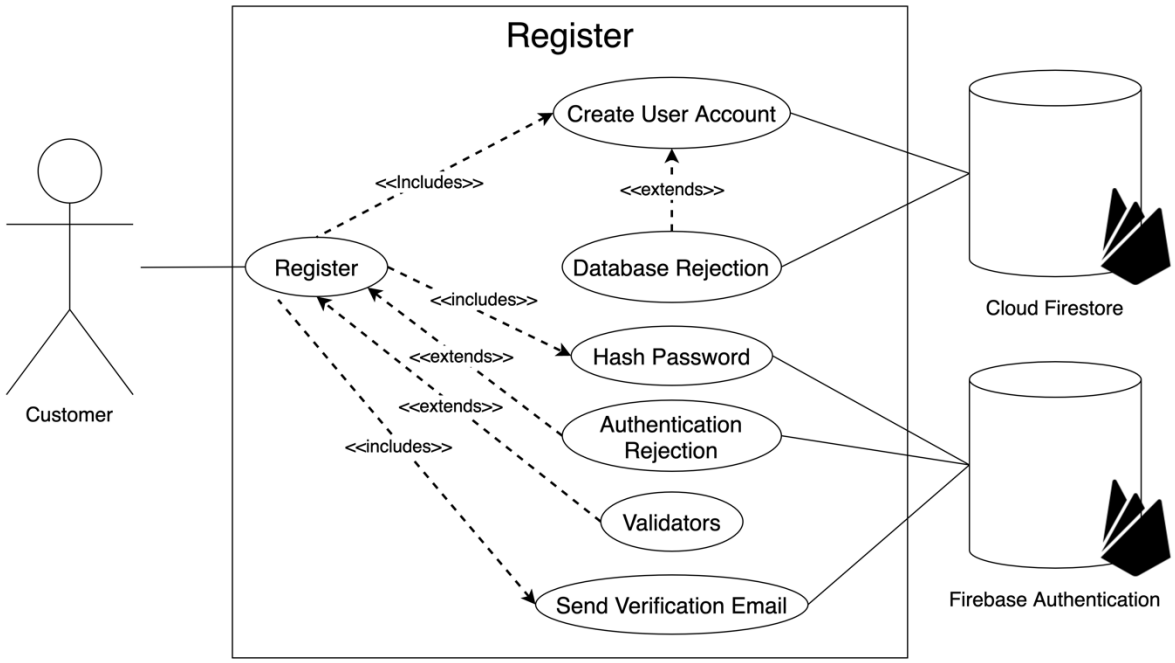
Scope

The scope of this use case is to allow customers to create a Nf_Kicks account with a secure password.

Description

This use case describes the actions taken when a customer registers for a Nf_Kicks account.

Use Case Diagram



2.1.1.5.3. Flow Description

Precondition

The application is on the Login page.

Activation

This use case begins when a customer presses the sign-up button.

Main flow

1. The application identifies that the sign-up button has been pressed.
2. The application navigates to the registration page.
3. The user enters their desired email and password to the assigned fields.
4. The application identifies that the user has entered the correct information into the fields.
5. The user presses the sign-up button.
6. The application identifies that the sign-up button was pressed.
7. The application sends a request to the Pwned Password's API to check if the user-provided password appeared in previous breaches.
8. The application receives a response indicating that the user's password is secure.
9. The application hashes the user's entered password.

10. The application sends a request to Firebase authentication to create a user account and send a verification email.
11. Firebase Authentication creates the account and sends a verification email.
12. Firebase Authentication responds with a created user account.
13. The application sends a request to Cloud Firestore to create a user account in the database.
14. The application navigates the customer to the email verification page.
15. The user verifies their email by clicking the click sent to their account.
16. The user presses the back button.
17. The application identifies that the back button has been pressed.

Alternate flow

A1 : Form Field Validation Failure on Email Field

1. The user provides an invalid email address.
2. The form-field validators detect that the email provided is incorrect.
3. The use case continues at position 3 of the main flow.

A2 : Form Field Validation Failure on Password Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided is incorrect.
3. The use case continues at position 3 of the main flow.

A3 : Form Field Validation Failure on Confirm Password Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided does not match with the password in the previous field.
3. The use case continues at position 3 of the main flow.

A4 : Email Already in use

1. The user provides an email that already exists as a user and the password to assigned fields.
2. The application using form-field validation detects that the user has entered a valid email and password.
3. The user presses the login button.
4. The application identifies that the user has pressed the login button.
5. The application sends a request to the Pwned Password's API to check if the user-provided password appeared in previous breaches.
6. The application receives a response indicating that the user's password is secure.
7. The application hashes the user's password.
8. The application sends the user account credentials to Firebase Authentication.

9. Firebase Authentication identifies that the email provided already exists as a user in the database.
10. Firebase Authentication response about the error.
11. The application receives this response and displays a window saying that the user with that email already exists.
12. The use case continues at position 3 of the main flow.

A5 : Password has been compromised

1. The user provides an email and a compromised password to assigned fields.
2. The application using form-field validation detects that the customer has entered a valid email and password.
3. The user presses the login button.
4. The application sends a request to the Pwned Password's API to check if the user-provided password appeared in previous breaches.
5. The application receives a response indicating that the user's password has appeared in previous branches.
6. The application displays a window saying that the user's password has been compromised and to choose another.
7. The use case continues at position 3 of the main flow.

A6 : Firebase Cloud Firestore server is down

1. The application sends a request to Cloud Firestore to create a user account in the database.
2. The application identifies that the database server down.
3. The request is saved in memory until the server is back up.
4. The use case continues at position 14 of the main flow.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

E3 Firebase Authentication server is down

1. The user enters their desired email and password to the assigned fields.

2. The application identifies that the user has entered the correct information into the fields.
3. The user presses the sign-up button.
4. The application identifies that the sign-up button was pressed.
5. The application sends a request to the Pwned Password's API to check if the user-provided password appeared in previous breaches.
6. The application receives a response indicating that the user's password is secure.
7. The application hashes the user's entered password.
8. The application sends a request to Firebase authentication to create a user account and send a verification email.
9. The application identifies that the authentication server down.
10. The application displays a window to the user saying that an error has occurred and to try again later.

Termination

The application navigates to the home page.

Post condition

The application waits for the user to perform an action.

2.1.1.6. Requirement 4: Get Stores

2.1.1.6.1. Description & Priority

This requirement retrieves store information for the customer.

Priority: High

2.1.1.6.2. Use Case

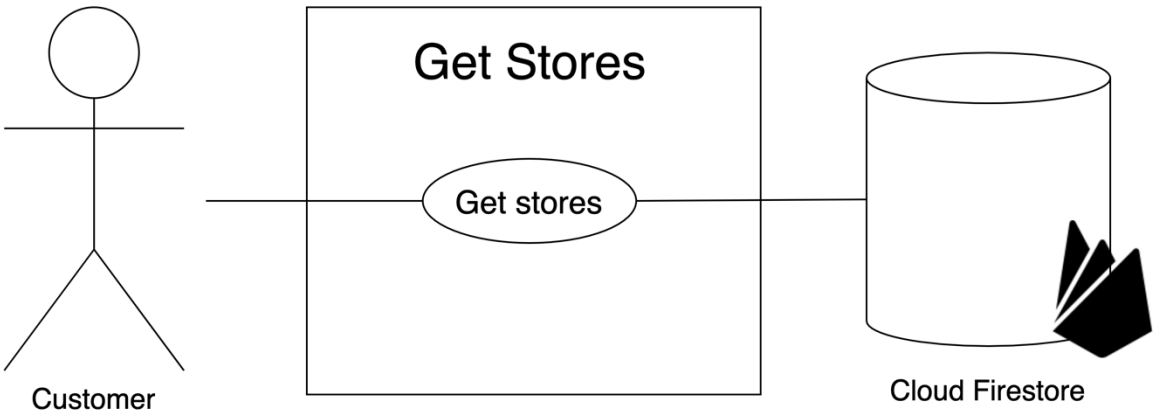
Scope

The scope of this use case is to retrieve store data for the user.

Description

This use case shows how stores are retrieved from the database.

Use Case Diagram



2.1.1.6.3. Flow Description

Precondition

The application is on the Home page.

Activation

This use case begins when a customer arrives on the home page.

Main flow

1. The application sends a request for stores.
2. Cloud Firestore responds with store data.
3. The application displays this store data on a map.
4. The customer taps a store.
5. The application identifies that a store marker has been tapped.
6. The application sends a request to Cloud Firestore for more data on the store that was tapped.
7. Cloud Firestore responses with data on the said store.
8. The application navigates the user to the store page.

Alternative flow

A1 : Firebase Cloud Firestore server is down

1. The application sends a request for stores.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.

3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application navigates to another page in the application.

Post condition

The application waits for the user to perform an action.

2.1.1.7. Requirement 6: Scan NFC Tag

2.1.1.7.1. Description & Priority

This requirement retrieves product information for the customer when an NFC tag is scanned.

Priority: Low

2.1.1.7.2. Use Case

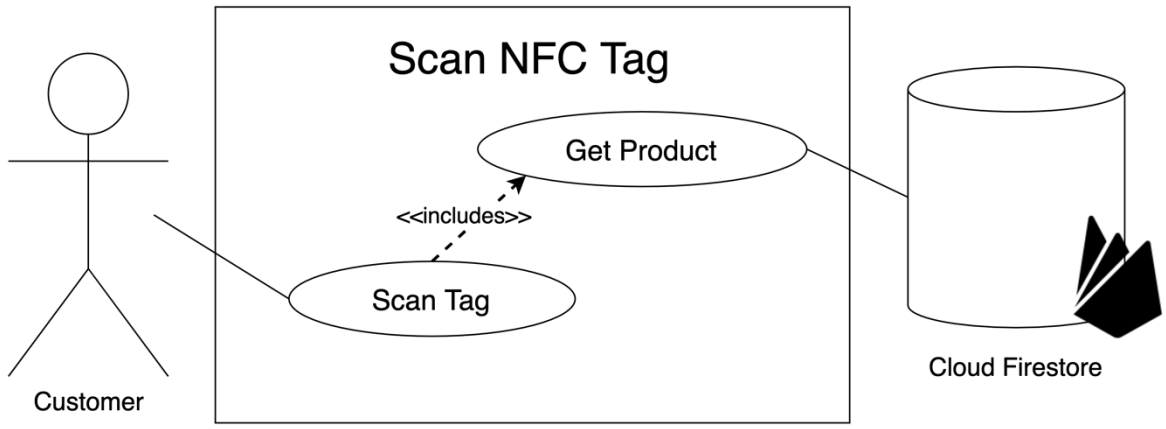
Scope

The scope of this use case is to navigate the customer to the product page for the NFC tag they scanned.

Description

This use case shows what happens when an NFC tag is scanned.

Use Case Diagram



2.1.1.7.3. Flow Description

Precondition

The application is on the Home page.

Activation

This use case begins when a customer scans an NFC Tag.

Main flow

1. The customer scans an NFC tag.
2. The application reads the NFC tag for a name to search.
3. The application retrieves the name from the NFC tag.
4. The application sends the name on the tag in a request to Cloud Firestore.
5. Cloud Firestore responds with the product's data.
6. The application receives the request.
7. The application navigates the user to the product's page.

Alternate flow

A1 : Scan a tag when the application is not open

1. The customer scans an NFC tag with their mobile device.
2. The application identifies that an NFC tag has been scanned
3. The mobile device opens the application.
4. The use case continues at the main flow 2.

A2 : Firebase Cloud Firestore server is down

1. The application sends a request for the product data.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Empty NFC Tag

1. The customer scans an empty NFC tag with their mobile device.
2. The application identifies that an NFC tag has been scanned
3. The application detects that the NFC tag is empty.
4. No requests are made to the database.

E2 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E3 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application navigates to another page in the application.

Post condition

The application waits for the user to perform an action.

2.1.1.8. Requirement 7: Get Store and Products

2.1.1.8.1. Description & Priority

This requirement retrieves store information along with its associated products for the customer.

Priority: High

2.1.1.8.2. Use Case

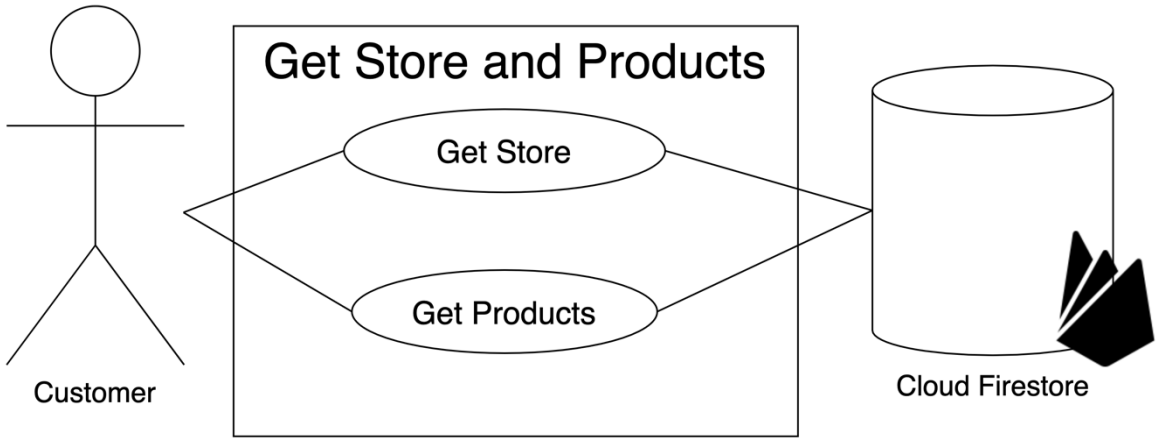
Scope

The scope of this use case is to illustrate the interaction between the store page and Cloud Firestore.

Description

This use case shows what happens in the background when store information is requested.

Use Case Diagram



2.1.1.8.3. Flow Description

Precondition

The application has navigated to a store page.

Activation

This use case begins when a customer navigates to the store page.

Main flow

1. The application navigates the customer to a store page.
2. The application sends a request to Cloud Firestore for store data and products.
3. Cloud Firestore responds with both store and product.
4. The application receives the data and displays it to the user.

Alternate flow

A1 : Firebase Cloud Firestore server is down

1. The application sends a request for store data.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application navigates to another page in the application.

Post condition

The application waits for the user to perform an action.

2.1.1.9. Requirement 8: Interact with Product

2.1.1.9.1. Description & Priority

This requirement retrieves product information along with similar associated products for the customer.

Priority: High

2.1.1.9.2. Use Case

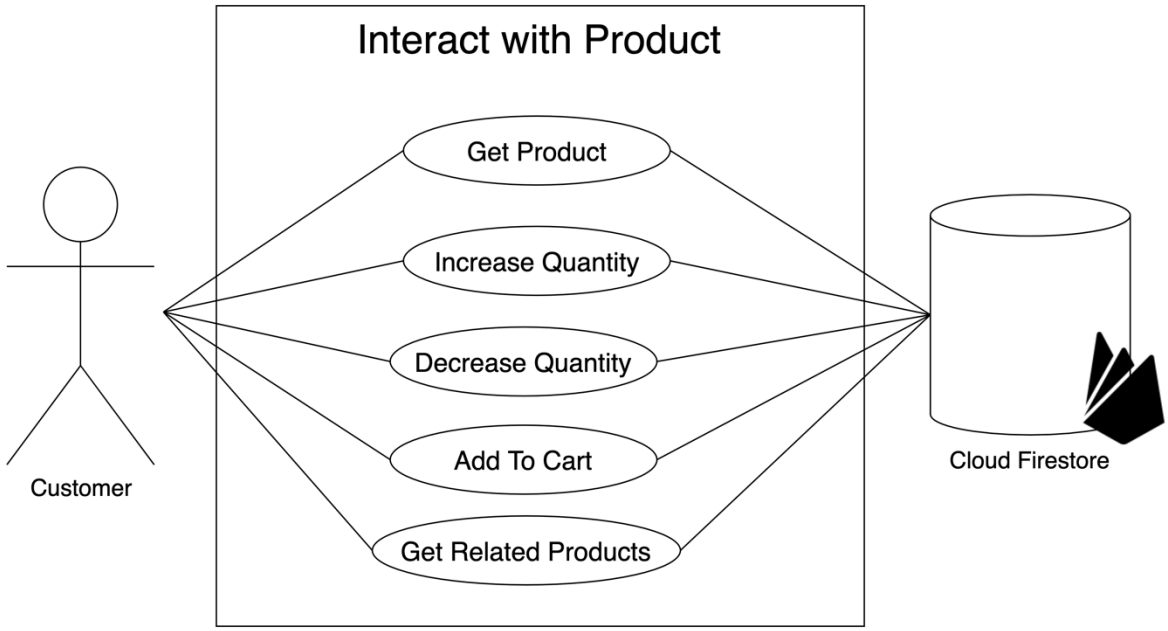
Scope

The scope of this use case is to illustrate the interaction between the product page and Cloud Firestore along with customer interactions.

Description

This use case shows what happens in the background when product information is requested and sent.

Use Case Diagram



2.1.1.9.2.1. Flow Description

Precondition

The application has navigated to a product page.

Activation

This use case begins when a customer navigates to the store page.

Main flow

1. The application navigates the customer to a product's page.
2. The application sends a request to Cloud Firestore for product data and related products.

3. Cloud Firestore responds with both the product's data and related product.
4. The application receives the data and displays it to the user.
5. The customer increases/decreases the quantity of product.
6. The application identifies that the customer has increased/decreased in the quantity of the product.
7. The customer added the product to their cart along with the quantity.
8. The application identifies that the user has added the product to their cart.
9. The application sends a request with the cart information to Cloud Firestore.
10. Cloud Firestore saves this cart data.
11. The application receives a response and notifies the user that the product has been added to the cart.

Alternate flow

A1 : Firebase Cloud Firestore server is down

1. The application sends a request for product data.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

A2 : Product is out of stock

1. The user attempts to add an out-of-stock product to their cart.
2. The application identifies that the product is out of stock
3. The application alerts the user that the product is out of stock.
4. The use case continues at the main flow 5.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when a product has been added to the cart.

Post condition

The application waits for the user to perform an action.

2.1.1.10. Requirement 9: Get Cart

2.1.1.10.1. Description & Priority

This requirement retrieves cart information for all store and has the ability to check out the products in the cart.

Priority: High

2.1.1.10.2. Use Case

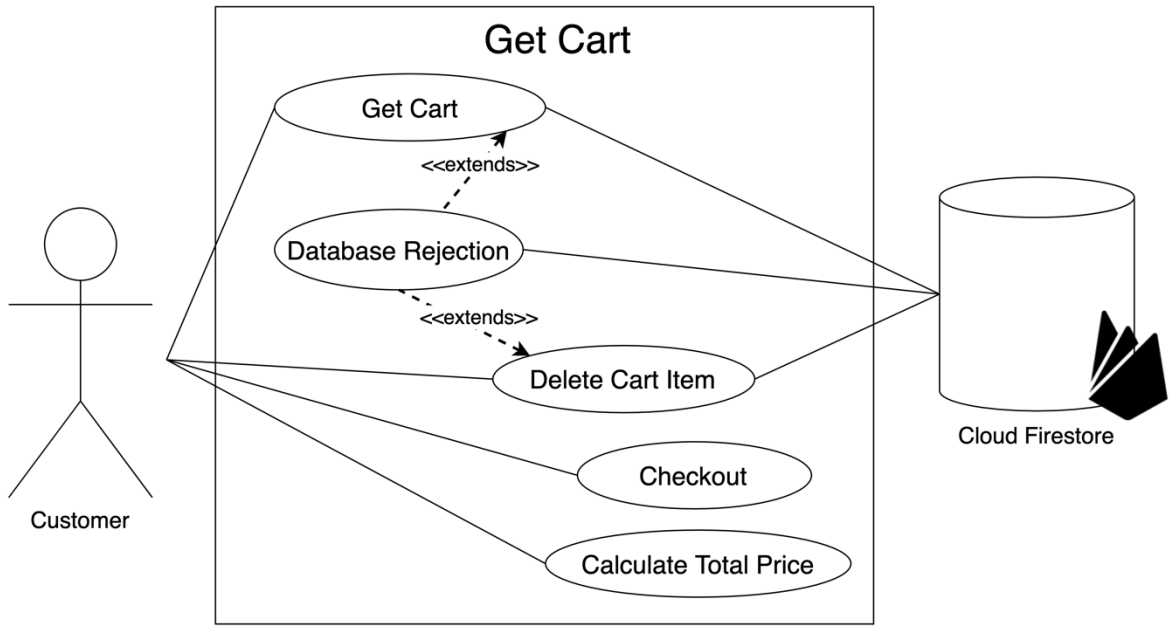
Scope

The scope of this use case is to illustrate the interaction between the cart page and Cloud Firestore along with a checkout.

Description

This use case shows what happens in the background when the customer interacts with the cart page.

Use Case Diagram



2.1.1.10.3. Flow Description

Precondition

The application has navigated to a cart page.

Activation

This use case begins when a customer navigates to the cart page.

Main flow

1. The application navigates the customer to a cart page.
2. The application sends a request to Cloud Firestore for cart data for all stores.
3. Cloud Firestore responds with cart data.
4. The application receives the data and displays it to the user.
5. The application calculates the total price of all the products.
6. The customer proceeds to press the checkout button to make an order.
7. The application navigates them to the checkout page for payment.

Alternate flow

A1 : Firebase Cloud Firestore server is down

1. The application sends a request for cart data.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

A2 : Delete Cart Item

1. The user swipes on an item in their cart to delete it.
2. The application identifies that an item in the cart has been swiped.
3. The application sends a request to Cloud Firestore to delete the select item from the current user's cart.
4. Cloud Firestore performs a delete action in the database.
5. The use case continues at the main flow 2.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is navigated to the checkout page.

Post condition

The application waits for the user to perform an action.

2.1.1.11. Requirement 10: Perform Checkout

2.1.1.11.1. Description & Priority

This requirement allows the customer to make a payment and create an order for the product in the cart.

Priority: High

2.1.1.11.2. Use Case

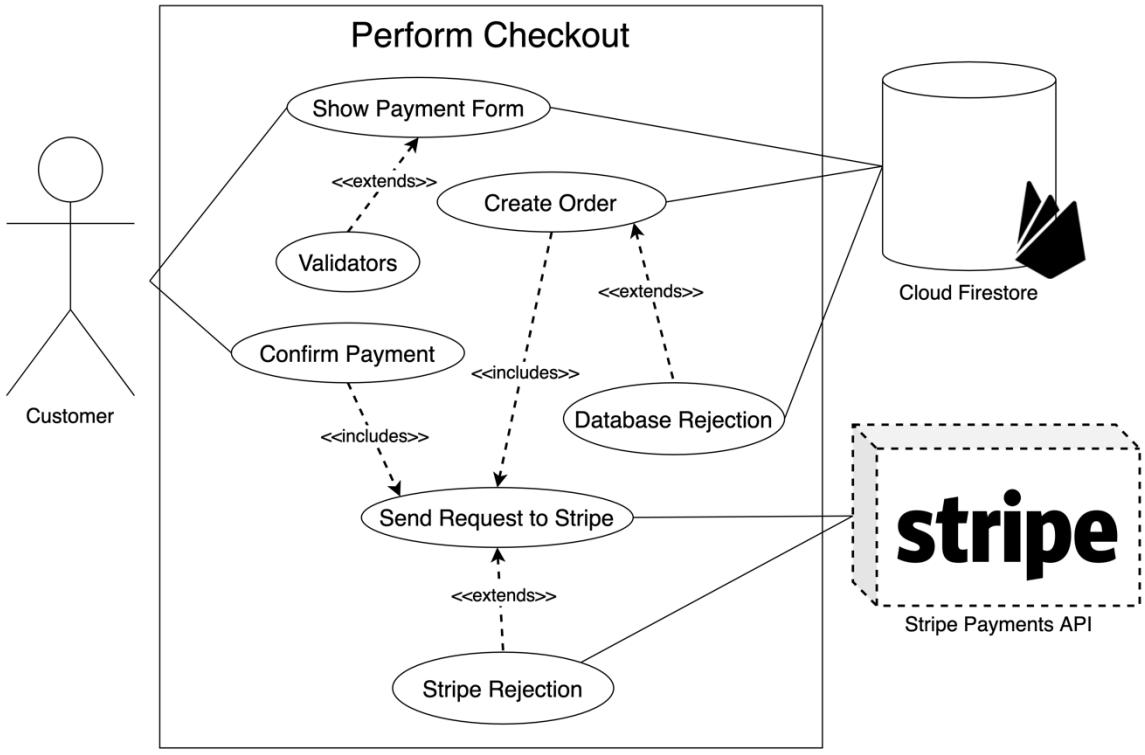
Scope

The scope of this use case is to illustrate the interaction between the checkout page, Cloud Firestore and the stripe payment API.

Description

This use case shows what happens in the background when the customer interacts with the checkout page.

Use Case Diagram



2.1.1.11.3. Flow Description

Precondition

The application has navigated to a checkout page.

Activation

This use case begins when a customer navigates to the checkout page.

Main flow

1. The application navigates the customer to a checkout page.
2. The customer enters their credit card information for payment.
3. The application validates the credit card information.
4. The customer confirms their payment.
5. The application makes a request to Stripe.
6. Stripe checks if the credit card information is correct.
7. Stripe identifies that the credit card information is indeed valid.
8. Stripe makes a transaction on the credit card.
9. Stripe sends a response to the application.
10. The application receives this response.
11. The application sends a request to Cloud Firestore to create an order for the customer.
12. Cloud Firestore creates the customer order.
13. Cloud Firestore responds saying the order has been made.
14. The application receives this response and tells the user that an order has been made.

Alternate flow

A1 : User provides an invalid card number

1. The application navigates the customer to a checkout page.
2. The customer enters incorrect card number.
3. The application validates the card number.
4. The application identifies that the card number is incorrect.
5. The application notifies the user that the card number is incorrect.
6. The use case continues at the main flow 2.

A2 : User provides an incorrect date or CVC number

1. The application navigates the customer to a checkout page.
2. The customer enters a valid card number.
3. The application validates the card number.
4. The user proceeds to enter the incorrect Expiration date and/or CVC
5. The customer confirms their payment.
6. The application makes a request to Stripe.
7. Stripe checks if the credit card information is correct.
8. Stripe identifies that the credit card information is incorrect.
9. Stripe sends a failed payment response back to the application.
10. The application receives this response and notifies the user that their payment has failed.

11. The use case continues at the main flow 2.

A3 : Insufficient funds

1. The application navigates the customer to a checkout page.
2. The customer enters a valid card number.
3. The application validates the card number.
4. The user proceeds to enter the correct expiration date and CVC
5. The customer confirms their payment.
6. The application makes a request to Stripe.
7. Stripe checks if the credit card information is correct.
8. Stripe identifies that the credit card information is indeed valid.
9. Stripe attempts to make a transaction on the credit card.
10. The transaction fails due to insufficient funds.
11. Stripe sends an insufficient funds response back to the application.
12. The application receives this response and notifies the user that their payment has failed.
13. The use case continues at the main flow 2.

A4 : Firebase Cloud Firestore server is down

1. The application sends a request to create an order.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The order data is stored in memory until a connection is established with Cloud Firestore.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Stripe payments server is down

1. The application sends a request to Stripe.
2. The application identifies that there is an issue connecting to Stripe.
3. The application cancels the payment and navigates back to the cart screen.
4. The use case continues at the main flow 1.

E2 : User cancels their payment

1. The application navigates the customer to a checkout page.
2. The user taps the cancel button.
3. The application cancel’s the payment for the products.

E3 : Device is Jailbroken or Rooted

1. The application scan the user’s device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E4 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user navigated back to the cart page.

Post condition

The application waits for the user to perform an action.

2.1.1.12. Requirement 11: Get Orders

2.1.1.12.1. Description & Priority

This requirement retrieves order information for all store.

Priority: High

2.1.1.12.2. Use Case

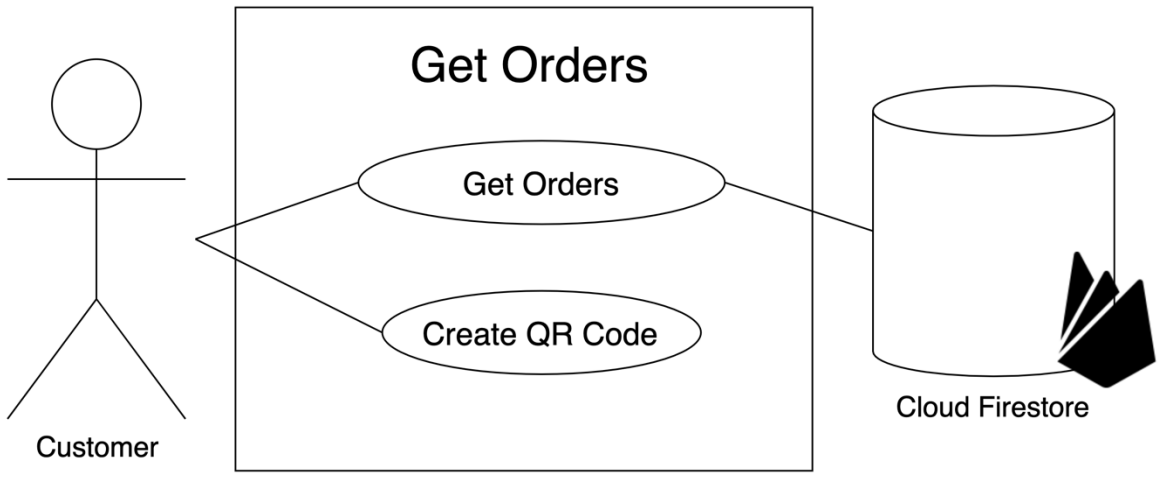
Scope

The scope of this use case is to illustrate the interaction between the orders page and Cloud Firestore.

Description

This use case shows what happens in the background when the customer interacts with the orders page.

Use Case Diagram



2.1.1.12.3. Flow Description

Precondition

The application has navigated to the orders page.

Activation

This use case begins when a customer navigates to the orders page.

Main flow

1. The application navigates the customer to an orders page.
2. The application sends a request to Cloud Firestore for order data for all stores.
3. Cloud Firestore responds with order data.
4. The application receives the data and displays it to the user.
5. The application generates a QR code for the customer.

Alternate flow

A1 : Firebase Cloud Firestore server is down

1. The application sends a request to retrieve order data.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The application displays a circular progress bar.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is displayed order information.

Post condition

The application waits for the user to perform an action.

2.1.1.13. Requirement 12: Update User Information

2.1.1.13.1. Description & Priority

This requirement allows a customer to update their current user information.

Priority: Medium

2.1.1.13.2. Use Case

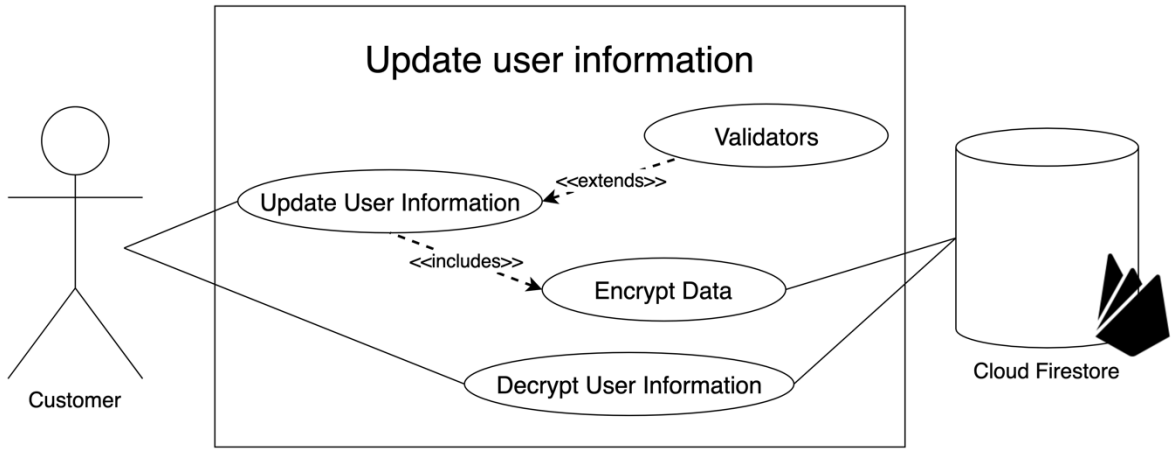
Scope

The scope of this use case is to illustrate the interaction between the update user information feature and Cloud Firestore.

Description

This use case shows what happens in the background when the customer decides to update their information stored in the application.

Use Case Diagram



2.1.1.13.3. Flow Description

Precondition

The application is currently on the settings page.

Activation

This use case begins when a customer taps the update information button.

Main flow

1. The application navigates the customer display two form fields.
2. The customer enters their desired user information.
3. The application validates the information that the user has provided.
4. The customer presses the submit button.
5. The application identifies that the submit button was pressed.
6. The application encrypts the new user information.

7. The application sends this new user information in a request to Cloud Firestore.
8. Cloud Firestore updates the current user's information.
9. The application requests the new user information to display to the user
10. The application receives the data and decrypts it.

Alternate flow

A1 : Form Field Validation Failure on Full-Name Field

1. The user provides an invalid full name.
2. The form-field validators detect that the full name provided is incorrect as it is not alphanumerical.
3. The use case continues at position 2 of the main flow.

A2 : Form Field Validation Failure on Phone Number Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided is incorrect.
3. The use case continues at position 2 of the main flow.

A3 : Firebase Cloud Firestore server is down

1. The application sends a request with the new user account information.
2. The application identifies that there is an issue connecting to Cloud Firestore.
3. The order data is stored in memory until a connection is established with Cloud Firestore.
4. The use case continues at the main flow 1.

Exceptional flow

E1 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E2 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is can view their new information.

Post condition

The application waits for the user to perform an action.

2.1.1.14. Requirement 13: Reset Password

2.1.1.14.1. Description & Priority

This requirement allows a customer to reset their password.

Priority: Medium

2.1.1.14.2. Use Case

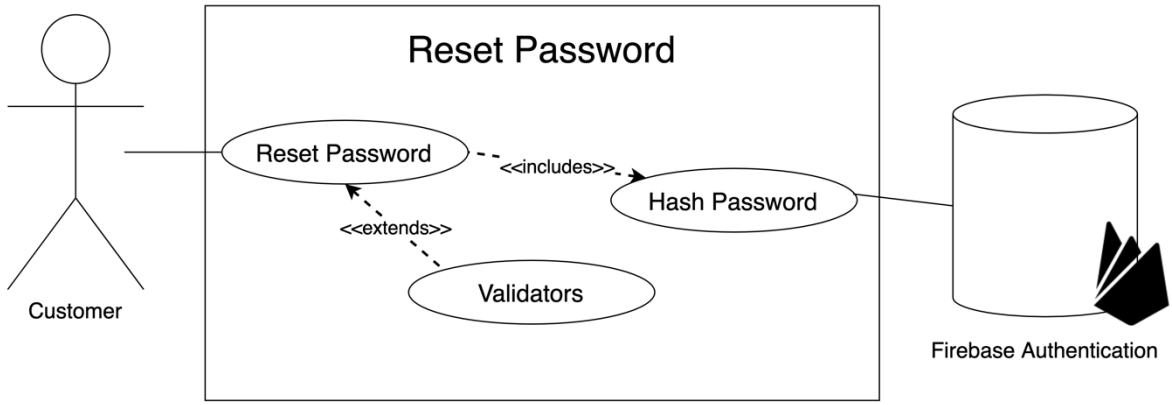
Scope

The scope of this use case is to illustrate the interaction between the reset password feature and Firebase Authentication.

Description

This use case shows what happens in the background when the customer decides to reset their password in the application.

Use Case Diagram



2.1.1.14.3. Flow Description

Precondition

The application is currently on the settings page.

Activation

This use case begins when a customer taps the reset-password button.

Main flow

1. The application navigates the customer display two form fields.
2. The customer enters their desired password.
3. The application validates the password that the user has provided.
4. The customer presses the submit button.
5. The application identifies that the submit button was pressed.
6. The application hashes the new password.

7. The application sends this new user information in a request to Firebase Authentication.
8. Firebase Authentication updates the user's password.
9. The application logs the user out of their account.

Alternate flow

A1 : Form Field Validation Failure on Password Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided is incorrect.
3. The use case continues at position 2 of the main flow.

A2 : Form Field Validation Failure on Confirm Password Field

1. The user provides an invalid password.
2. The form-field validators detect that the password provided does not match with the password in the previous field.
3. The use case continues at position 2 of the main flow.

A3 : Password has been compromised

1. The application sends a request to the Pwned Password's API to check if the user-provided password appeared in previous breaches.
2. The application receives a response indicating that the user's password has appeared in previous branches.
3. The application displays a window saying that the user's password has been compromised and to choose another.
4. The use case continues at position 2 of the main flow.

Exceptional flow

E1 : Firebase Authentication server is down

1. The application sends a request for Firebase Authentication to update their password.
2. The application identifies that the database server down.
3. The request is saved in memory until the server is back up.

E2 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E3 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.

2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is logged out of their Nf_Kicks account.

Post condition

The application waits for the user to perform an action.

2.1.1.15. Requirement 14: Delete Account

2.1.1.15.1. Description & Priority

This requirement allows a customer to delete their Nf_Kicks account and their information.

Priority: Medium

2.1.1.15.2. Use Case

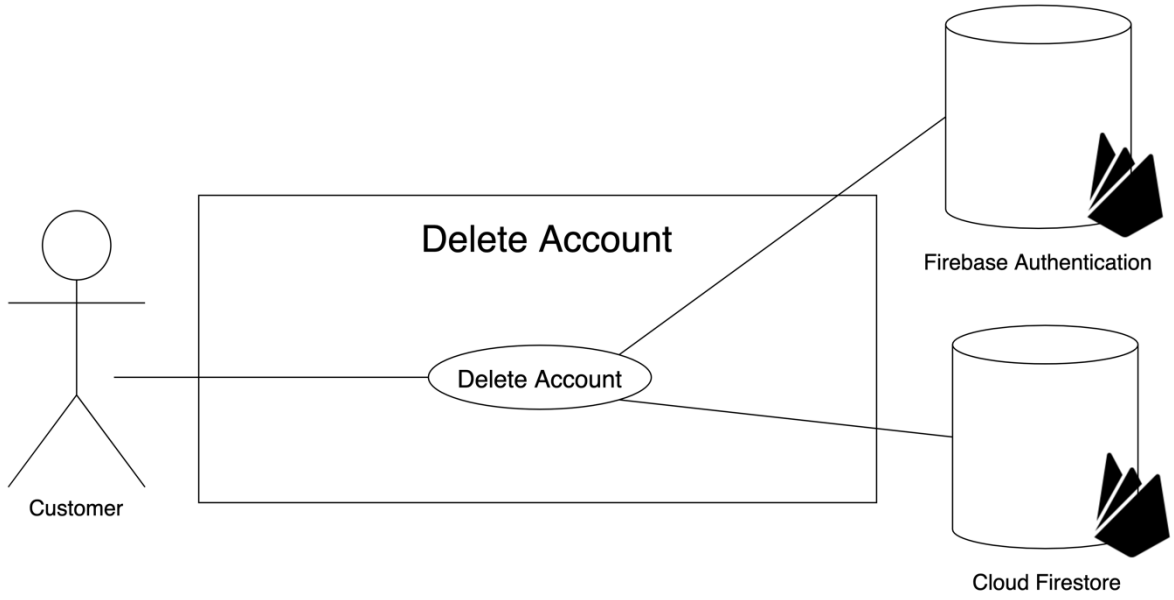
Scope

The scope of this use case is to illustrate the interaction between the delete account feature and Firebase services

Description

This use case shows what happens in the background when the customer decides to delete their account in the application.

Use Case Diagram



2.1.1.15.3. Flow Description

Precondition

The application is currently on the settings page.

Activation

This use case begins when a customer taps the delete account button.

Main flow

1. The application identifies that the customer has decided to delete their account.
2. The application asks the customer if they are sure in the form of a popup.
3. The customer confirms their decision.
4. A request is sent to Cloud Firestore to delete the current user's account information.
5. Cloud Firestore receives this request and performs a delete on said user's information.
6. The application proceeds to send a delete request to Firebase Authentication.
7. Firebase Authentication receives this request and performs a delete of the current user's account.
8. The application identifies that all user information has been purged.
9. The application navigates the user back to the login page.

Exceptional flow

E1 : User cancels their action

1. The user decides to not delete their account and taps the cancel button.
2. The application identifies that the user has tapped the cancel button.
3. The application cancels the deletion action.

E2 : Firebase Authentication server is down

1. The application sends a request for Firebase Authentication to update their password.
2. The application identifies that the database server down.
3. The request is saved in memory until the server is back up.

E3 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E4 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.

3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is navigated to the login page.

Post condition

The application waits for the user to perform an action.

2.1.1.16. Requirement 15: Upload Profile Picture

2.1.1.16.1. Description & Priority

This requirement allows a customer to upload a profile picture to their Nf_Kicks account.

Priority: Medium

2.1.1.16.2. Use Case

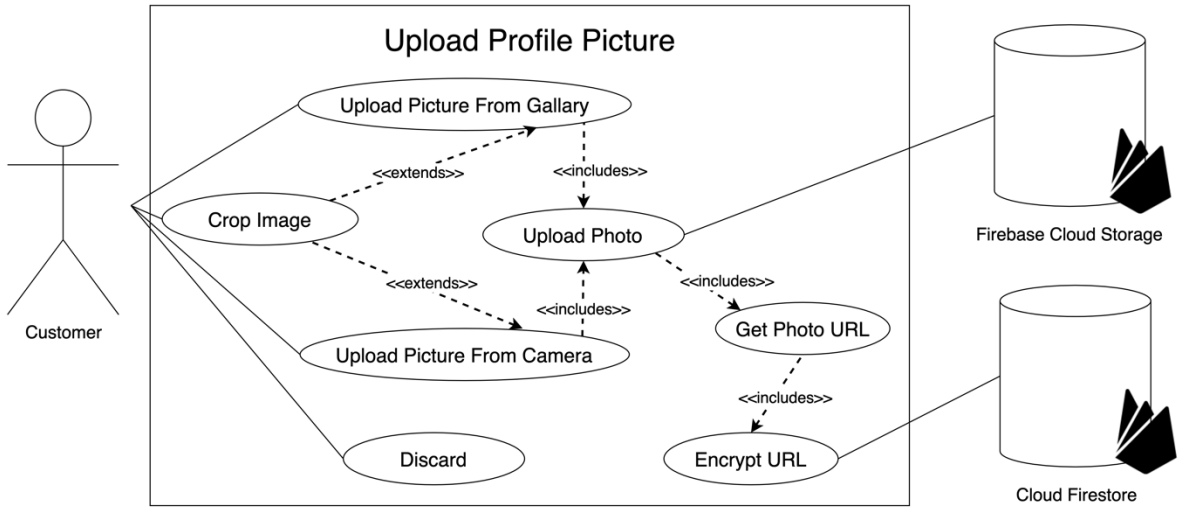
Scope

The scope of this use case is to illustrate the interaction between the profile upload feature and Firebase services

Description

This use case shows what happens in the background when the customer decides to upload a profile picture to their account in the application.

Use Case Diagram



2.1.1.16.3. Flow Description

Precondition

The application is currently on the settings page.

Activation

This use case begins when a customer taps their current profile picture button.

Main flow

1. The application identifies that the user tapped their current profile picture.
2. The application navigates the user to the profile upload page.
3. The user has the option to choose a picture from their device or using their camera.
4. When the user chooses a photo.
5. When the user is satisfied with their chosen photo, they can proceed to update their profile photo.
6. The user presses the submit button.
7. The application identifies that the submit button was pressed.
8. The application sends the image in the form of a request to Firebase Cloud Storage.
9. Cloud Storage saves the user profile picture in a file and responds with the URL for said picture.
10. The application takes this URL and encrypts it.
11. The application proceeds to send the encrypted URL to Cloud Firestore to be saved as part of the user's information.
12. The application navigates back to the setting page where it requests the user's information to show the user's profile picture.

Alternate flow

A1 : User discards the image

1. The user decides to discard their chosen image.
2. The application identifies that the user has tapped the discard button.
3. The application removes the chosen image.
4. The use case continues on the main flow 3.

A2 : Crop the image

1. The user decides to crop their chosen image.
2. The application identifies that the user has tapped the crop button.
3. The application navigates the user to a screen to crop their image.
4. The user crops their image.
5. The user taps the check button.
6. The application updates the current chosen image with the cropped image.
7. The use case continues on the main flow 5.

Exceptional flow

E1 : Firebase Cloud Firestore server is down

1. The application sends a request to Cloud Firestore to save the new profile image URL.

2. The application identifies that the database server down.
3. The request is stored in memory until a connection is established.
4. The request is saved in memory until the server is back up.

E2 : Firebase Cloud Storage server is down

1. The application sends a request to Cloud Storage to save the new profile image.
2. The application identifies that the server down.
3. The request is stored in memory until a connection is established.
4. The request is saved in memory until the server is back up.

E3 : Device is Jailbroken or Rooted

1. The application scan the user's device for specific directories.
2. The application identifies that root or jailbroken device directories exist.
3. The application routes the user to a screen asking them to have an un-rooted device.

E4 : No WIFI connectivity

1. The application checks if there is a WIFI connection on the device.
2. The application identifies that there is no WIFI connection.
3. The application routes the user to a screen asking them to check their connection.

Termination

The application terminates when the user is navigated back to the settings page.

Post condition

The application waits for the user to perform an action.

2.1.1.17. Requirement 16: Perform Read On Stores/Products – Security Rules

2.1.1.17.1. Description & Priority

This requirement is able to retrieve either store or product information from Cloud Firestore.

Priority: Medium

2.1.1.17.2. Use Case

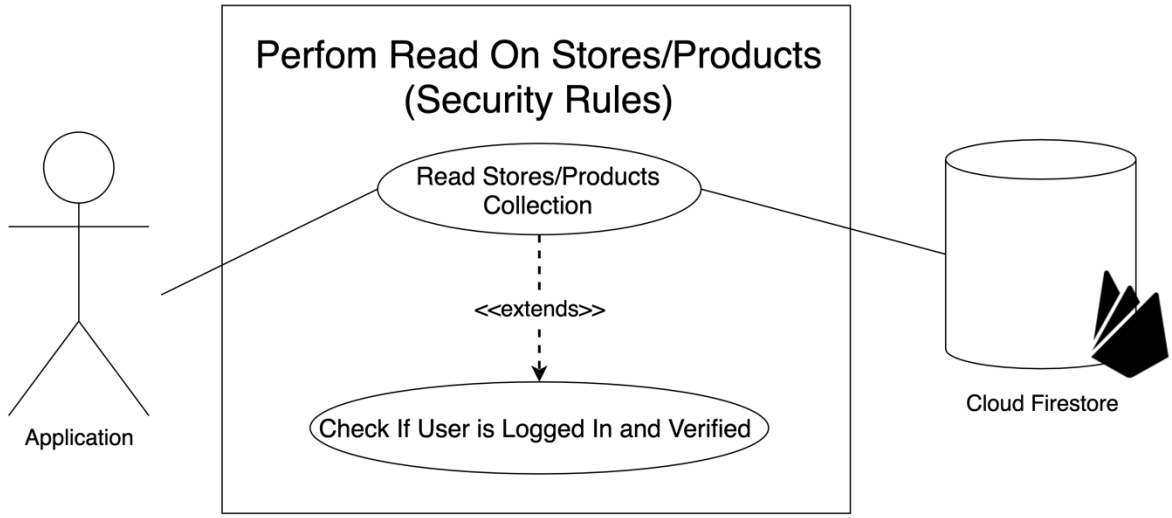
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore for Stores or Products.

Description

This use case shows what happens in the background when a request to Cloud Firestore.

Use Case Diagram



2.1.1.17.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore for either stores or products data.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check the request to see if a user was logged in when it was sent.
4. Security rule identifies that the user was indeed logged in and responds with either product or store data.

Exceptional flow

E1 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user
2. The security rules reject the request to read store and product data.

E2 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the request to read store and product data.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.18. Requirement 17: Get current user's information – Security Rules

2.1.1.18.1. Description & Priority

This requirement is able to retrieve the current logged in user's information from Cloud Firestore.

Priority: Medium

2.1.1.18.2. Use Case

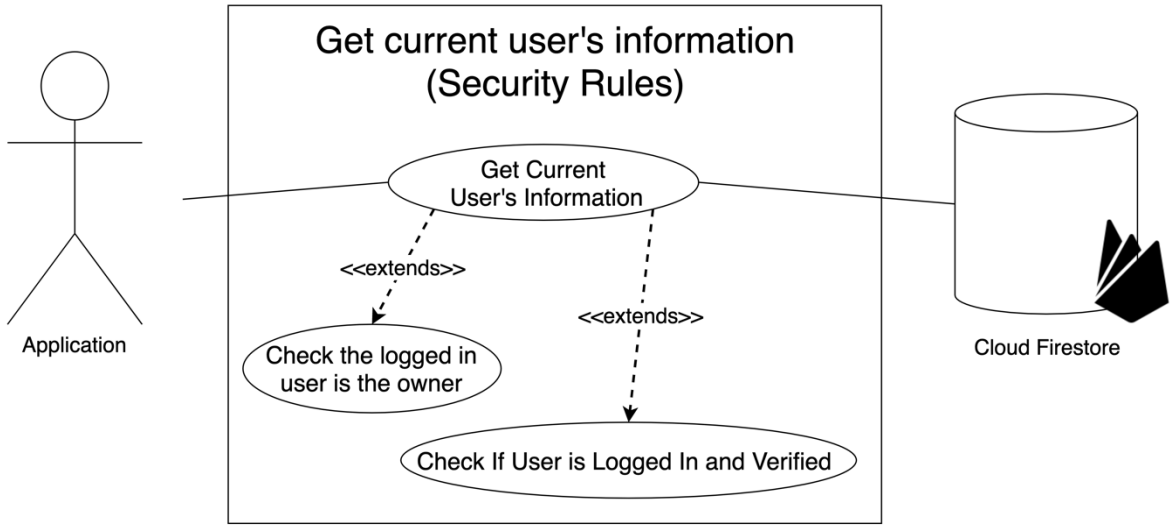
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore for the current user's information.

Description

This use case shows what happens in the background when a request to Cloud Firestore.

Use Case Diagram



2.1.1.18.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore for the current logged in user's information.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check the request to see if a user was logged in and verified when it was sent.
4. Security rules proceed to check if the logged-in user is the owner of the information.
5. Security rules identify that the user was indeed logged in, verified and is the owner of the information.
6. Cloud Firestore responds with the current user's information.

Exceptional flow

E1 : User Is Not The Owner

1. The security rules identify that the user who sent the request is not the owner of the information collection they are trying to read from.
2. The security rules reject the request to read user information.

E2 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user
2. The security rules reject the request to read user information.

E3 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the request to read user information.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.19. Requirement 18: Update current user's information – Security Rules

2.1.1.19.1. Description & Priority

This requirement is able to update the current logged in user's information from Cloud Firestore.

Priority: Medium

2.1.1.19.2. Use Case

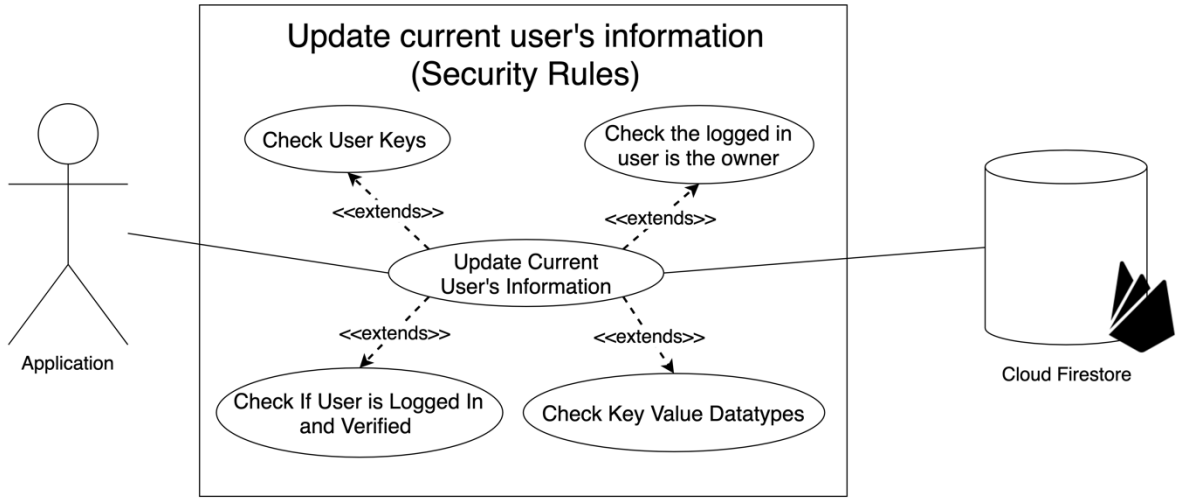
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore to update the current user's information.

Description

This use case shows what happens in the background when an update request is sent to Cloud Firestore.

Use Case Diagram



2.1.1.19.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore to update the current logged in user's information.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check request to see if a user was logged in and verified when it was sent.
4. Security rules proceed to check if the logged-in user is the owner of the information.
5. Security rules check if specific keys related to the user's collection are present.
6. Security rules all key values for specific data types.
7. Security rules identify that the user was indeed logged in, verified, is the owner of the information, has the correct keys and key-value data types.
8. Cloud Firestore updates with current user's information.

Exceptional flow

E1 : Request Does Not Have Correct Keys

1. The security rules identify that the request does not have the right keys for updating user information.
2. The security rules reject the update request.

E2 : Request Does Not Have Correct Data Types

1. The security rules identify that the request does not have the right data types for updating user information.
2. The security rules reject the update request.

E3 : User Is Not The Owner

1. The security rules identify that the user who sent the request is not the owner of the information collection they are trying to read from.
2. The security rules reject the update request.

E4 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user.
2. The security rules reject the update request.

E5 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the update request.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.20. Requirement 19: Delete current user's information – Security Rules

2.1.1.20.1. Description & Priority

This requirement is able to delete the current logged in user's information from Cloud Firestore.

Priority: Medium

2.1.1.20.2. Use Case

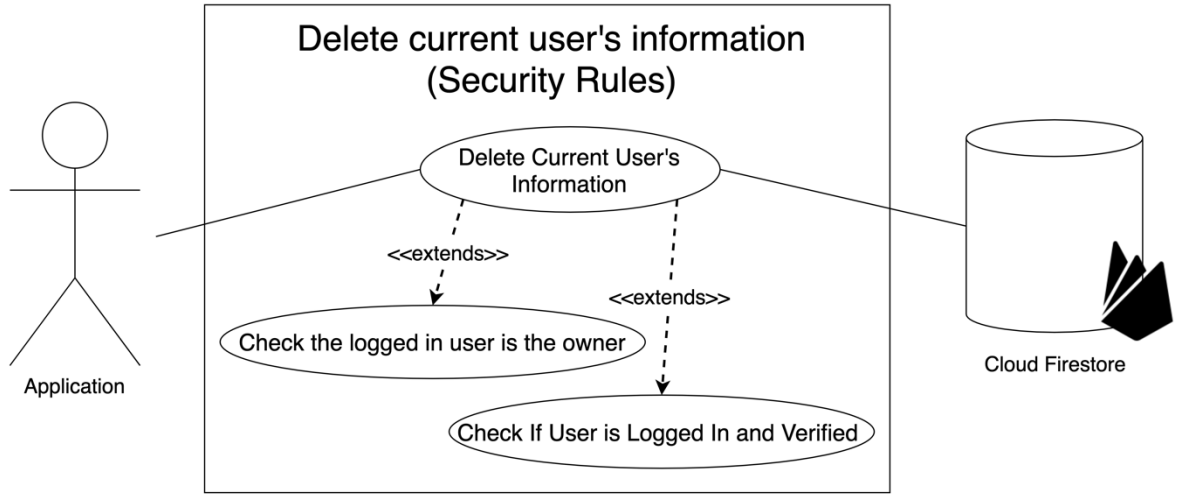
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore to delete the current user's information.

Description

This use case shows what happens in the background when a delete request is sent to Cloud Firestore.

Use Case Diagram



2.1.1.20.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore to delete the current logged in user's information.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check request to see if a user was logged in and verified when it was sent.
4. Security rules proceed to check if the logged-in user is the owner of the information.
5. Security rules identify that the user was indeed logged in, verified and is the owner of the information.
6. Cloud Firestore deletes the current user's information.

Exceptional flow

E1 : User Is Not The Owner

1. The security rules identify that the user who sent the request is not the owner of the information collection they are trying to read from.
2. The security rules reject the delete request.

E2 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user.
2. The security rules reject the delete request.

E3 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the delete request.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.21. Requirement 20: Create a new user account – Security Rules

2.1.1.21.1. Description & Priority

This requirement is able to create a new user account by sending a request to Cloud Firestore.

Priority: Medium

2.1.1.21.2. Use Case

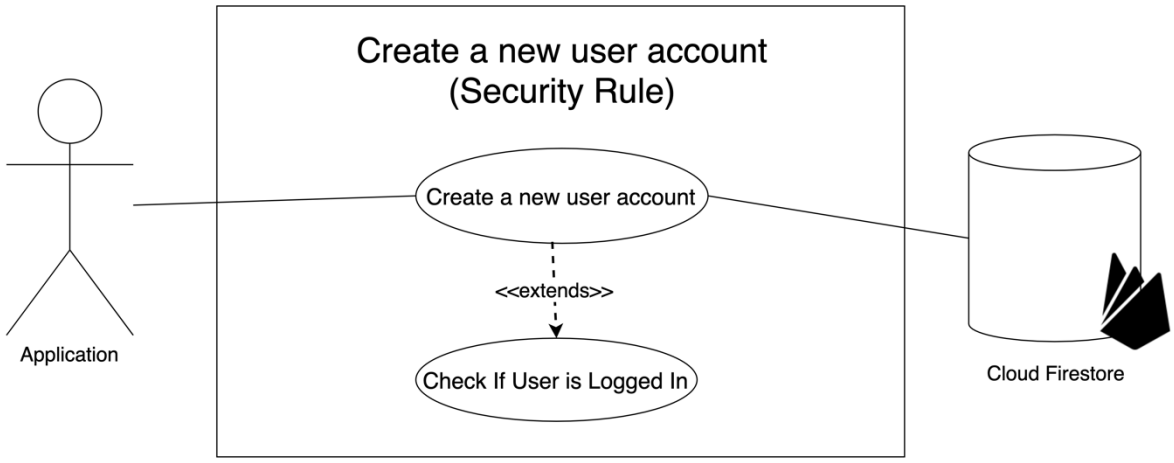
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore to perform a create request on a new account.

Description

This use case shows what happens in the background when a create request is sent to Cloud Firestore.

Use Case Diagram



2.1.1.21.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore to create a new user.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check request to see if a user was logged in when the request was sent.
4. Security rules identify that the user was indeed logged in.
5. Cloud Firestore creates an account for the current user.

Exceptional flow

E1 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user.
2. The security rules reject the create request.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.22. Requirement 21: Create new cart/order for current user – Security Rules

2.1.1.22.1. Description & Priority

This requirement is able to make a request to Cloud Firestore to either make a cart or an order for the currently logged in user.

Priority: Medium

2.1.1.22.2. Use Case

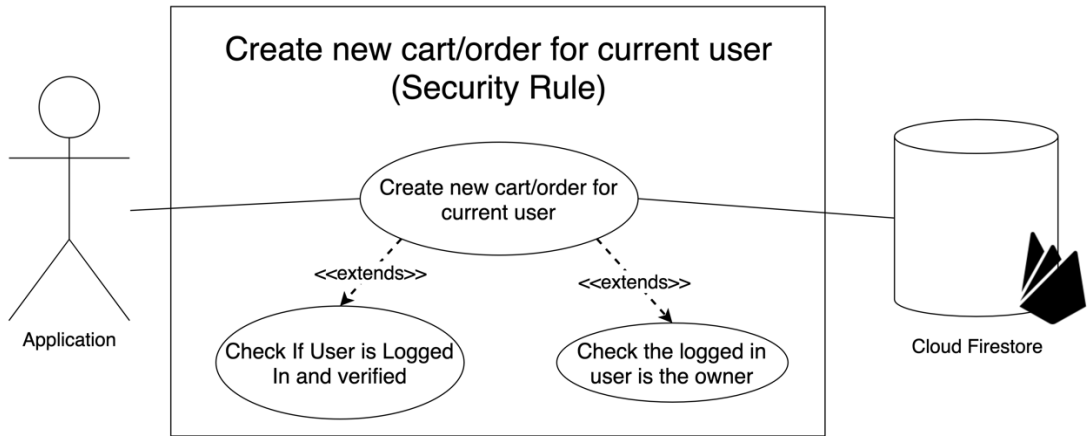
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore to perform a create request.

Description

This use case shows what happens in the background when a create request is sent to Cloud Firestore.

Use Case Diagram



2.1.1.22.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore to create a new cart or order for the current user.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check request to see if a user was logged in and verified when the request was sent.

4. Security rules check if the user that made the request is the owner of the account.
5. Security rules identify that the user was indeed logged in and is the owner of the account.
6. Cloud Firestore creates a new cart/order for the current user.

Exceptional flow

E1 : User Is Not The Owner

1. The security rules identify that the user who sent the request is not the owner of the information collection they are trying to read from.
2. The security rules reject the create request.

E2 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user.
2. The security rules reject the create request.

E3 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the create request.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

2.1.1.23. Requirement 22: Update/Delete current user's cart – Security Rules

2.1.1.23.1. Description & Priority

This requirement is able to make a request to Cloud Firestore to either update or delete a cart for the current logged in user.

Priority: Medium

2.1.1.23.2. Use Case

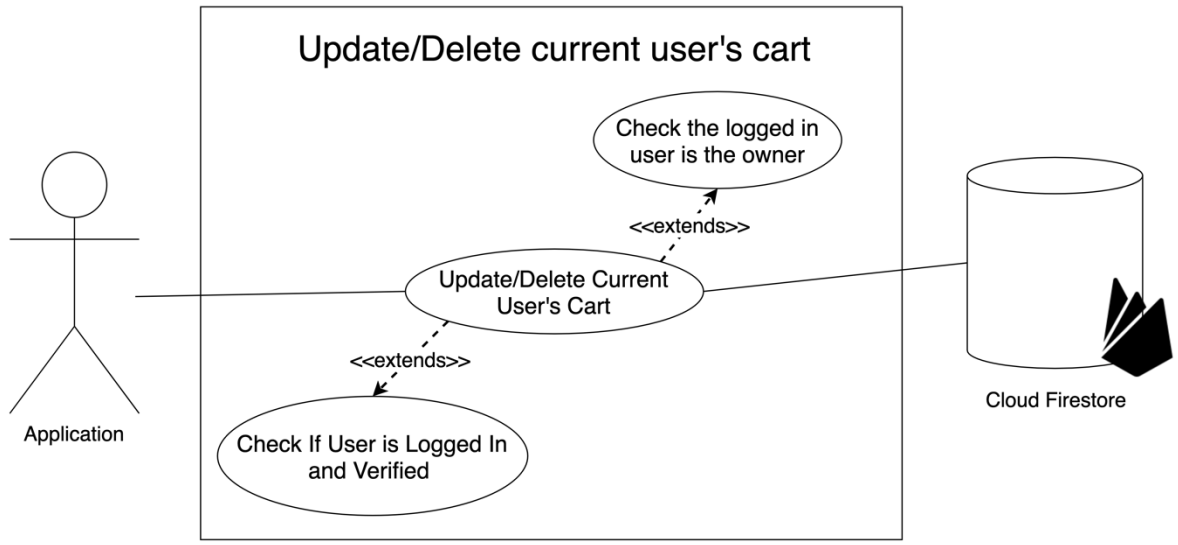
Scope

The scope of this use case is to illustrate what occurs when the application makes a request to Cloud Firestore to perform either an update or delete request.

Description

This use case shows what happens in the background when an update or delete request is sent to Cloud Firestore.

Use Case Diagram



2.1.1.23.3. Flow Description

Precondition

The application makes a request to Cloud Firestore.

Activation

This use case begins when Cloud Firestore Security rules analyse the request from the application.

Main flow

1. The application makes a request to Cloud Firestore to update or delete a cart for the current user.
2. Cloud Firestore receives the request and filters it through its security rules.
3. Security rules check request to see if a user was logged in and verified when the request was sent.
4. Security rules check if the user that made the request is the owner of the account.
5. Security rules identify that the user was indeed logged in and is the owner of the account.
6. Cloud Firestore performs an update or a delete request on a cart for the current user.

Exceptional flow

E1 : User Is Not The Owner

1. The security rules identify that the user who sent the request is not the owner of the information collection they are trying to read from.
2. The security rules reject the update or delete request.

E2 : User Is Not Logged In

1. The security rules identify that the request was not sent by a logged-in user.
2. The security rules reject the update or delete request.

E3 : User Is Not Verified

1. The security rules identify that the request was not sent by a verified user
2. The security rules reject the update or delete request.

Termination

The application terminates when a response is sent back to the application.

Post condition

The application waits for the user to perform an action.

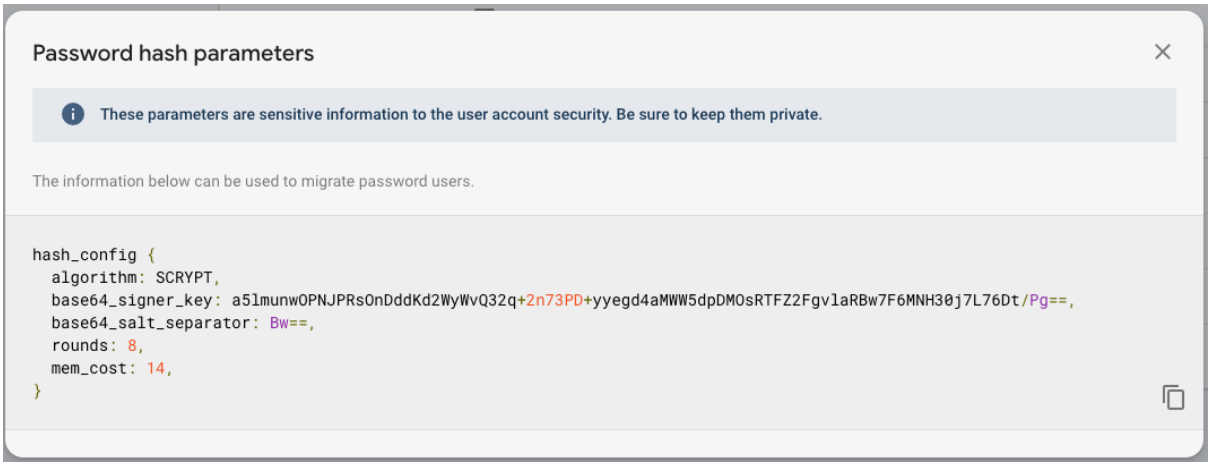
2.1.2. Data Requirements

Firebase Authentication

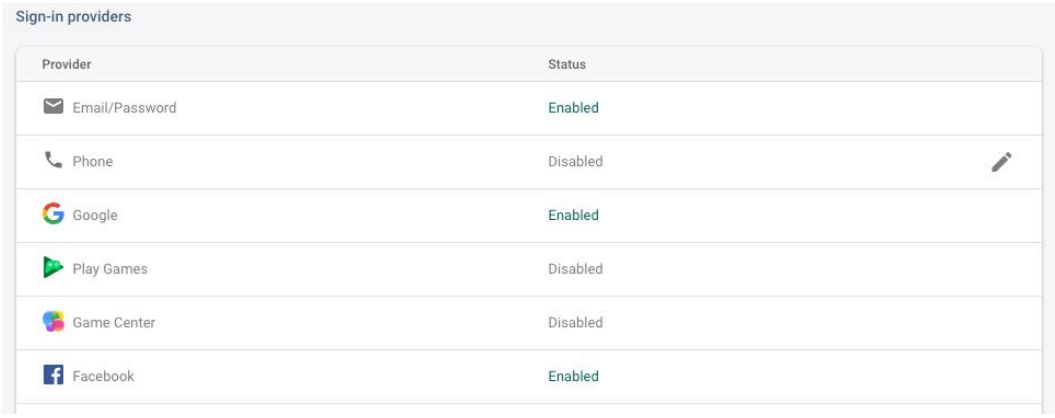
Nf_Kicks delegates the registration and login of customers to Firebase Authentication. This service enables Nf_Kicks to securely store user information in a way that not even the developer of the application can have full access to. When a customer account is created on Nf_Kicks the developer only has access to their basic profile. This includes information such as their Unique User UID, email, information on if the email is verified, phone number, name, profile image URL and sign-in provider.

In Nf_Kicks the User UID is used heavily utilized to provide the customer access to Firebase Storage for saving images and Cloud Firestore for saving user information in the application. The User UID is employed to further enhance the security of the user's account by using it togrant access to said user information.

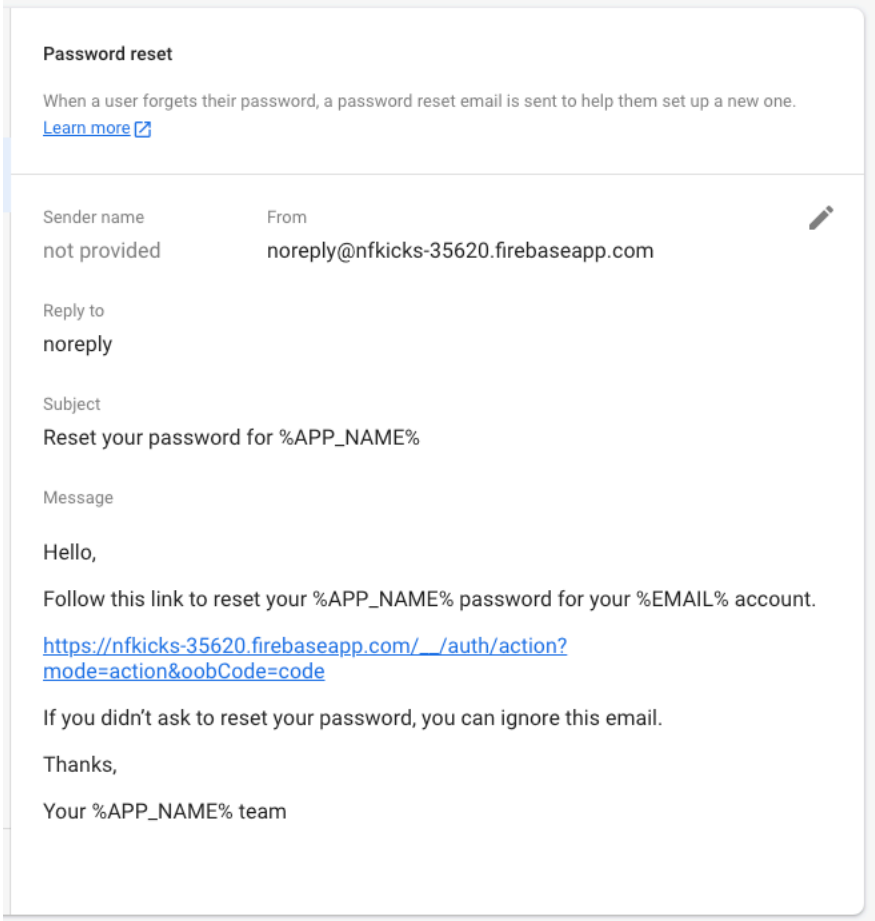
Other user data is protected and secured through Firebase. This includes the password. When a customer chooses to save their password, it is stored by Google with Password hashing parameters that only the developers have access to and should not be shown to an unauthorized user. Firebase authentication also provides developers with the option to delete user accounts if customers decide to leverage their right to be forgotten by requesting account erasure.



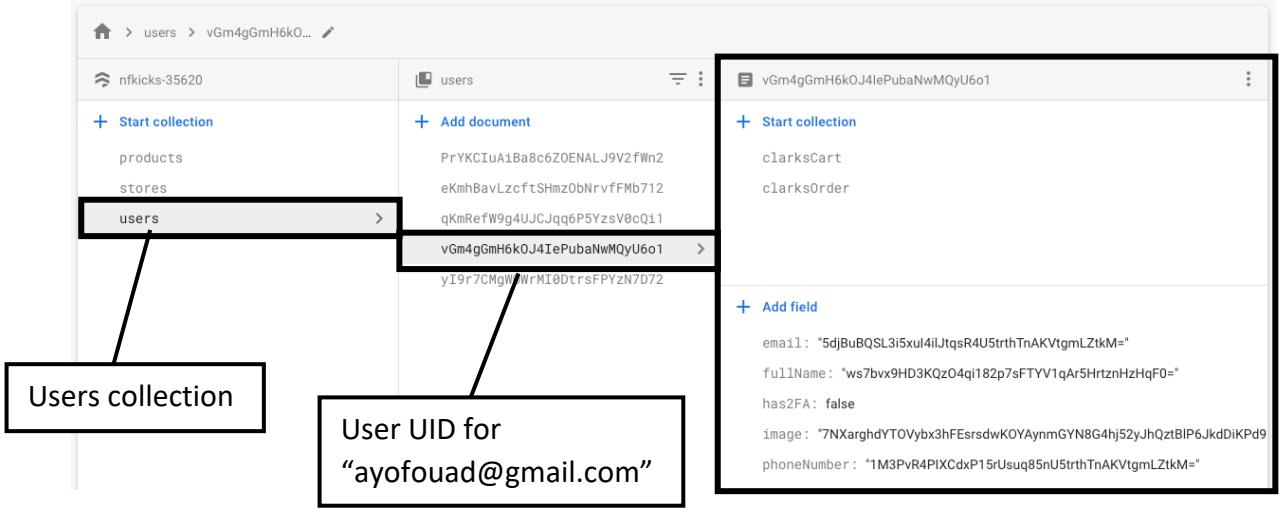
Choice of sign-in provider was important to the application for an untrustworthy provider may compromise a customer's Nf_Kicks account in such a way that the application would have no way of securing said customer account. The reason Google and Facebook were selected over other alternatives is not just that they are extremely popular sites with the capability to easily sign-in customers, but also because these sign-in providers rigorously check their services for security vulnerabilities, enabling the developer to utilize up-to-date security updates offered by these providers.



The final data requirements that I'll be discussing is account recovery through a password reset mechanism which Firebase Authentication provides. There is even a default email template that is currently being used in the application, as can be seen here:



Firebase Cloud Firestore



As mentioned previously, the User UID from Firebase Authentication is used to perform CRUD operations on user-specific information and to restrict access to said information via

security rules. The following image is an example of user information specific to the customer “ayofouad@gmail.com”. I know this because the User UID of said user is present as the name of the document.

As can be seen under ayofouad@gmail’s user information, all personal information is encrypted to ensure user privacy and compliance with Article 32 of the General Data Protection Regulation.

“Article 32 of the General Data Protection Regulation (GDPR) requires Data Controllers and Data Processors to implement technical and organizational measures that ensure a level of data security appropriate for the level of risk presented by processing personal data.” (Encryption | General Data Protection Regulation (GDPR), 2021)

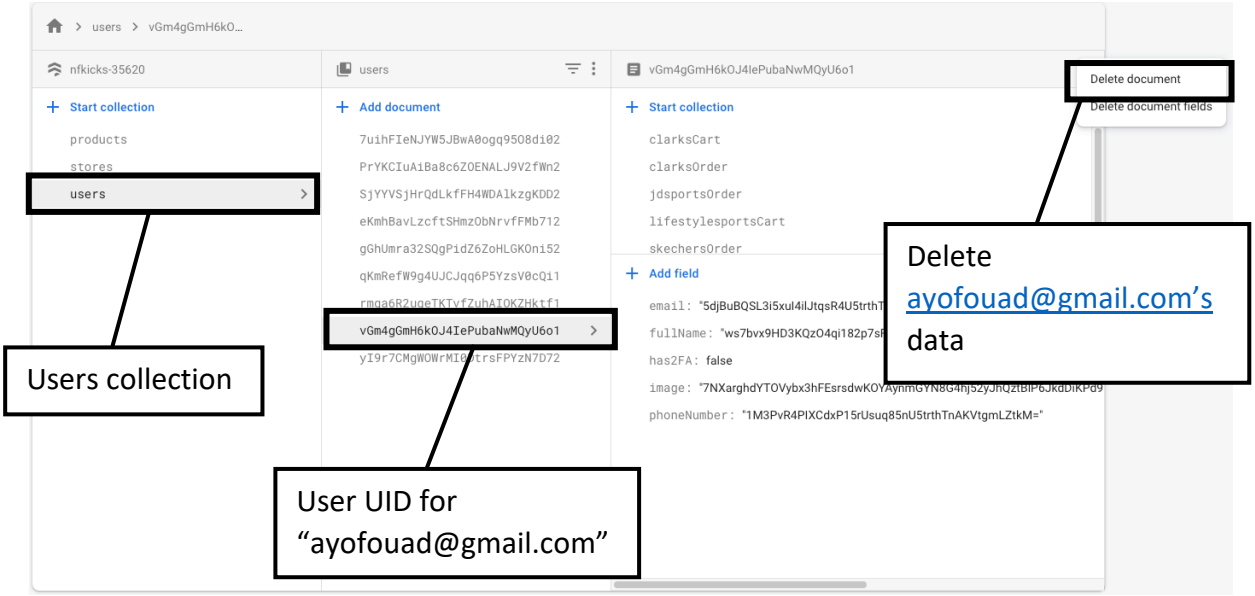
The technical measures used to ensure a level of data security in this instance is the use of encryption, while the organizational measure would be restricting access to the Firebase Console. The console is by default restricted because only I have access.

The only data that is not encrypted is information about their cart and orders, because GDPR does not consider this to be personally identifiable data. Furthermore, to ensure data integrity, any requests with the intent of modifying or deleting user data without customer permission have a security rule written to mitigate against such acts, of which I have written many and will detail later in this document.

Firebase Cloud Firestore also makes it easy to comply with Article 17 of the General Data Protection Regulation.

“The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay.” (Art. 17 GDPR – Right to erasure (‘right to be forgotten’) | General Data Protection Regulation (GDPR), 2021)

Cloud Firestore provides this by allowing the deletion of user documents in the database to quick and easily delete the information based on a user’s UUID, like so:



Stripe Payments

As discussed previously in the technologies section, Flutter's Stripe payments API demands that all data submitted to Stripe servers be protected using HTTPS over TLS; this is important since it makes it more challenging for an attacker to interpret containing card information in plain text (Security at Stripe, 2021). Additionally, no payment history is retained or retrievable in any way due to the fact that customers are unable to maintain their credit card information for future transactions inside the application.

2.1.3. User Requirements

To properly use the Nf_Kicks application, a user must have an Android or an IOS smartphone that is linked to the internet; however, if the device is equipped with NFC, additional features will be available inside the application.

A Google or Facebook account is needed to speed up the registration and login phase but is also optional.

To make payments in the application a credit card is needed, however, a test card can be used:

NUMBER	BRAND	CVC	DATE
4242 4242 4242 4242	Visa	Any 3 digits	Any future date

2.1.4. Environmental Requirements

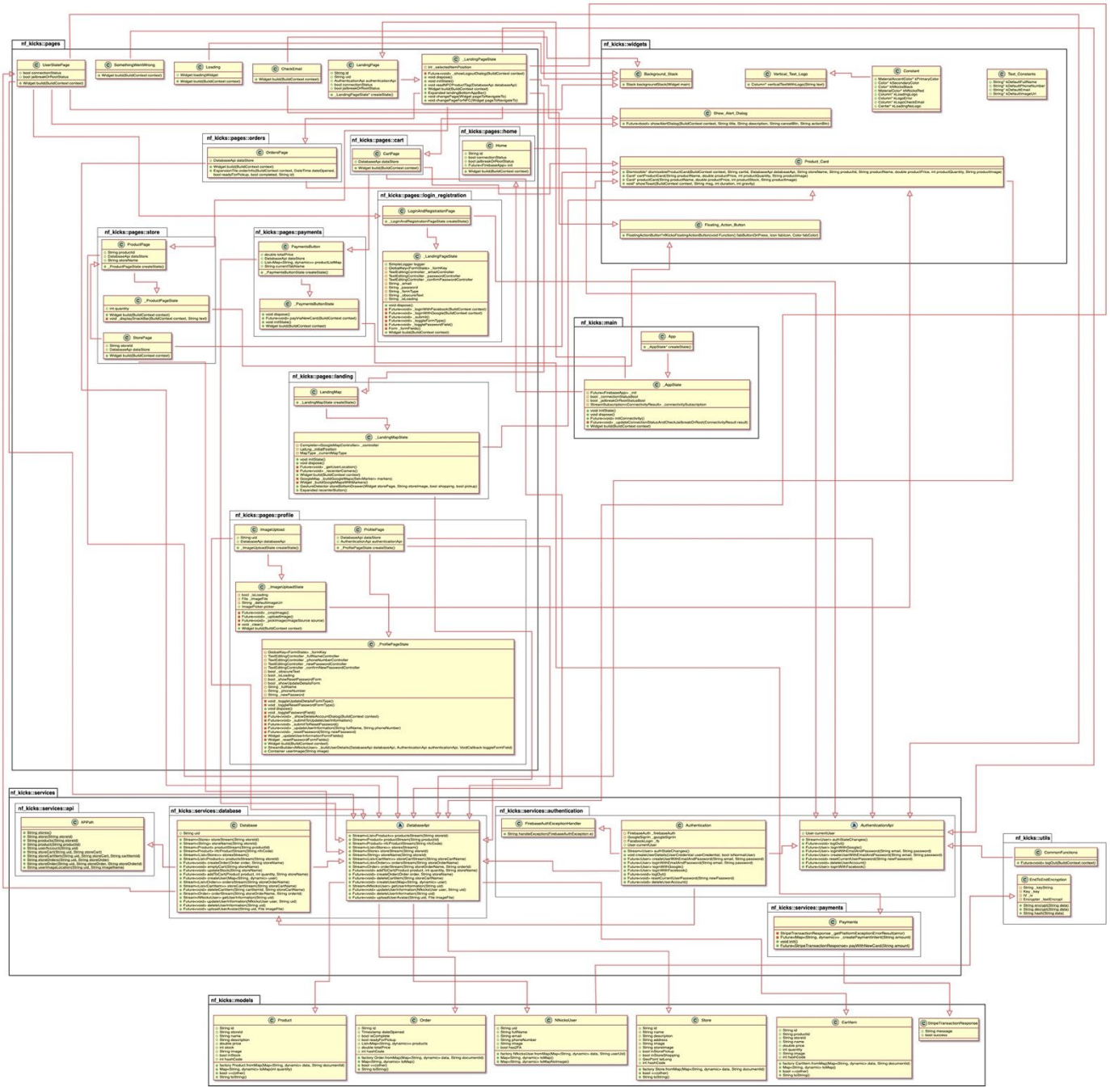
To run the code for this program, you'll need a laptop or a desktop computer; preferably a MacBook, because in order to build and run IOS applications, a mac is required. A device that can run Android Studio and XCode is desirable.

To build and compile the code-base the [Flutter SDK] by Google is a requirement.

2.1.5. Usability Requirements

The type of users to utilize this application are users on both the IOS and Android mobile operating systems with an interest in purchasing products online.

2.2. Design & Architecture



Link to diagram

The main packages to look at here are the services and models, which both have classes that heavily interact with Firebase services. For example, the Authentication class interacts with Firebase Authentication by having functions that handle login and logout features. However, the Authentication class also interacts with the Database class's abstract class, which is

primarily used for performing operations such as creating requests to Cloud Firestore. The model class NfkicksUser assists with this create a request by requiring the user account document to adhere to a strict set of rules, such as what data type the user's phone number and fullname should be. The NfkicksUser model is also responsible for encrypting the user's information because it makes calls to the EndToEndEncryption class.

2.3. Implementation

2.3.1. Security Rules – Cloud Firestore – Input Validation

Although Firebase Services like Cloud Firestore and Cloud Storage are useful for storing and retrieving data in the cloud, unintended or malicious writes may occur, resulting in the loss of critical data critical to the application's functionality. This is why Security Rules are required to monitor and prevent access to specific resources when rules are not met.

The following security rule code checks for the inclusion of specific keys and key-value pairs with relevant data types in update requests sent to Firebase.

```
// Only update User Information when
// the user is logged in, verified,
// has specific keys, they own the
// resource and key values must have
// specific datatype
allow update: if checkUserKeys()
&& checkKeyValuesInUsersDatatypes()
```

Through reviewing the functions that are named, we can learn more about the types of keys that the rule expects in a request.

```
// Check that the keys in the request have the
// following names, if they do not reject the request
function checkUserKeys() {
  return request.resource.data.keys().hasAll([
    'email','fullName','has2FA','image','phoneNumber'
  ]);
}
```

As we can see in this instance, the necessary keys are email, fullName, has2FA, image, and phoneNumber; if any of these keys are absent from the update request submitted to the database, the request will be rejected. This function is critical for input validation and representation because we want data submitted to the database to be checked for fields critical to the application's functionality and without these fields, the user's information cannot be represented properly in the application.

The next function to take a look at checks that the key values in the update request have specific data types.

```
// Check that the values assigned to the keys are strings,
// if they are not reject the request
function checkKeyValuesInUsersDatatypes() {
  return (request.resource.data.email is string)
  && (request.resource.data.fullName is string)
  && (request.resource.data.has2FA is bool)
  && (request.resource.data.image is string)
  && (request.resource.data.phoneNumber is string);
}
```

The data types that this function would search for in an update request are the following: email key contains a string, fullname key contains a string, has2FA key contains a Boolean

value, image key contains a string, and phoneNumber key contains a string. This function is critical in relation to input validation and representation because we do not want the user's information to affect the application's client-side functionality or to protect user details from being manipulated by a malicious user.

2.3.2. Security Rules – Cloud Storage – Input validation

The following is a Firebase Cloud Storage security rule mainly present to performs input validation on the images sent to Cloud Storage.

```
// Allow only logged in and verified users to
// saved images to the database. Allow images
// under a certain size and with specific
// extensions to be saved to the database.
allow create: if request.auth.uid != null
  && request.auth.token.email_verified
  && request.resource.size < 25 * 1024 * 1024
  && (request.resource.contentType.matches('image/png')
  || request.resource.contentType.matches('image/jpeg')
  || request.resource.contentType.matches('image/gif')
  || request.resource.contentType.matches('image/jpg'));
```

Cloud storage will validate the image before saving it to ensure that it is under a certain size and contains one of the four extensions specified in the rule; otherwise, it will be rejected. This rule is critical since malicious users may upload possible malware from an image or a file that does not use one of the image file extensions mentioned above.

2.3.3. Password Requirements & Pwned Passwords

Previously, the Nf_Kicks needed passwords to be at least eight characters long and comprised of a combination of upper and lowercase letters, as well as one special character. However, according to the National Institute of Standards and Technology's Password Standards (NIST), using a variety of different characters in a password puts undue pressure on the user to recall their password for that service. Rather than having a password of unique characters, NIST advises that the provider compare the user's password to a database of known breached passwords from past data breaches. Additionally, supporting copy and paste features for users who use password managers and allowing for emoji support (NIST Password Standards, 2020).

I've chosen to use a mix of what I already have and NIST's recommendations in my project. To compare the user's password to previously compromised passwords, I used the pwned passwords API, which helps a user to determine whether or not their password of choice has been exposed in previous data breaches. This information assists the user in selecting a safe and secure password that is not easily guessed and is not susceptible to brute force attack. The API is provided through the use of the password-compromised package in Flutter. The following is the code that checks for a compromised password.

```
final compromised = await isPasswordCompromised(_password);
if (compromised) {
  showAlertDialog(context,
      title: 'Sign up failed',
      description:
          'The password you have chosen has been compromised choose another.',
      actionBtn: 'OK');
} else {
```

To ensure that the customer adheres to the NF_Kicks password policies I've written validation code that verifies that their preferred password meets the minimum and maximum length requirements, as well as the inclusion of a special character.

```
final MultiValidator _passwordValidator = MultiValidator([
  RequiredValidator(errorText: 'Password is required'),
  MinLengthValidator(8,
      errorText: 'Password must be at least 8 digits long'),  // MinLengthValidator
  MaxLengthValidator(64,
      errorText: 'Password must be under 64 digits long'),  // MaxLengthValidator
  PatternValidator(r'(?=.*?[#?!@$%^&*-])',
      errorText: 'Passwords must have at least one special character')  // PatternValidator
]);  // MultiValidator
```

2.3.4. Double Password Hashing

As mentioned previously in the data requirements section, the password hashing is provided by default with the use of Firebase Authentication. However, to protect the user's password in the application before and when it is transmitted to Firebase Authentication for password hashing again, we hash it using SHA 512.

```
static String hash({String password}) {
  final _bytes = utf8.encode(password);
  final _hash = sha512.convert(_bytes);
  return _hash.toString();
}
```

Double password hashing is beneficial not only if a user's password is intercepted by an attacker, but also in the event that the project's Firebase Authentication instance suffers a data leak since it significantly complicates the password cracking process.

2.3.5. End-to-End Encryption

End to end encryption is used to protect user information in the event of a data breach and to hide information during data transmission to a database in the event of a man-in-the-middle attack on a user. Access is often restricted when data is encrypted since only permitted parties possessing the decryption key may view the contents of the data. Article 32 of the General Data Protection Regulation requires the service provider to incorporate

appropriate technical and organizational measures to protect personal data. In this case, the technical measure is the use of encryption. (GDPR Encryption, 2018)

Here is the code that handles the encryption of the data.

```
static String encrypt({String data}) {
  return _textEncrypt.encrypt(data, iv: _iv).base64;
}
```

Here is the code that handles the decryption of the data.

```
static String decrypt({String data}) {
  return _textEncrypt.decrypt64(data, iv: _iv);
}
```

The algorithm used for end-to-end encryption is AES, which is the standard for encrypting private information within mobile applications. A 16-byte initial vector (IV) is used alongside the plain text and ciphertext.

```
final String _keyString = env['NFKICKS_KEY'].toString();
final Key _key = Key.fromUtf8(_keyString);
final IV _iv = IV.fromLength(16);
final Encrypter _textEncrypt = Encrypter(AES(_key));
```

Here is an example of encrypted data from the database.

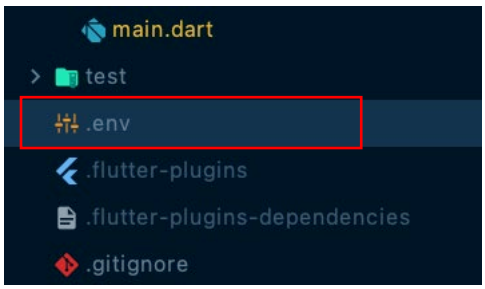
```
email: "7sTarB4JK2+NyM87ikciqoV2MvhptBblAqdvgGDbtEE="
fullName: "zs7GsFsjIXnxqYdR6zZLwQ=="
has2FA: false
image: "7NXarghdYTOfyP08gV8wrNhhMucE33/HacQM6wCy2T1lH3QP1NLpj9H5D
phoneNumber: "1M3PvR4PIXCdxP15rUsuq85nU5trthTnAKVtgmLZtkM="
```

2.3.6. Secrets File

You might have noticed the following previous code snippet:

```
final String _keyString = env['NFKICKS_KEY'].toString();
```

This variable refers to an encryption key included in a .env file that contains all the secrets that should not be included in the application's code. This is where the real encryption key is located:



Here's what is inside:

```
NFKICKS_KEY=r4u7x!A%D*G-KaPd
STRIPE_API_URL=https://api.stripe.com/v1
STRIPE_SECRET=sk_test_77GGQnProPxjztAvaC3dCiUM00a1Xc5vdW
PUBLISHABLE_KEY=pk_test_uWsbmRDANMClUuQjJpY4cAxZ
GOOGLE_MAPS_KEY=AIzaSyDFQ7tdsvCGI5T35-NqywZOsGqq_i4zWi4
```

When the program is executed, these secrets are injected into the application at the appropriate locations. This helps prevent API keys from being compromised by a user browsing through the project's GitHub repository or to mitigate against a reverse engineering attack by making it far more difficult to locate these keys.

2.3.7. Code obfuscation

Obfuscation protects against unwanted access or manipulation of the application functionality by attackers seeking to either extract sensitive information such as API keys or reset the price of products to zero. Obfuscation guards against this by encrypting the application's source code, rendering it near impossible to decipher and interpret for an attacker.

In my application, code obfuscation occurs twice on the android build; the first instance is by the usage of the Flutter framework itself. To obfuscate our Flutter application, we must use the following command to create the APK in release mode:

```
flutter build apk --obfuscate
```

This will generate an APK in release mode with code obfuscation. This form of obfuscation is available for both IOS and Android builds.

We may further obfuscate the data by using an Android source code obfuscation method named ProGuard to obfuscate it once more. Unlike Flutter obfuscation, ProGuard can shrink and optimize the Flutter application's Java code. To make ProGuard accessible, we must provide the following buildType in our build.gradle file.

```
buildTypes {
    release {
        shrinkResources true
        zipAlignEnabled true
        minifyEnabled true
        signingConfig signingConfigs.release
        useProguard true

        setProguardFiles([getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'])
    }
    debug {
```

These are the configurations needed to enable ProGuard. The last thing to do is to create a proguard-rules.pro file in the same directory as the build.gradle file. With the following

```
#Flutter Wrapper
-keep class io.flutter.app.** { *; }
-keep class io.flutter.plugin.**  { *; }
-keep class io.flutter.util.**  { *; }
-keep class io.flutter.view.**  { *; }
-keep class io.flutter.**  { *; }
-keep class io.flutter.plugins.**  { *; }
```

This file contains classes needed for the Flutter framework to function. ProGuard's obfuscation will be validated in the testing section.

2.3.8. Rooted/Jailbroken Device Detection

Rooted/Jailbroken mobile devices have access to features that normal mobile users do not, such as modifying the price of products in an app or bypassing the stripe payment system to avoid paying at the checkout, to protect against this I have used a Flutter package called trust_fall to check if a user's device has been rooted or jailbroken and pre-emptively prevent them from using the application. Here is the code that handles this:

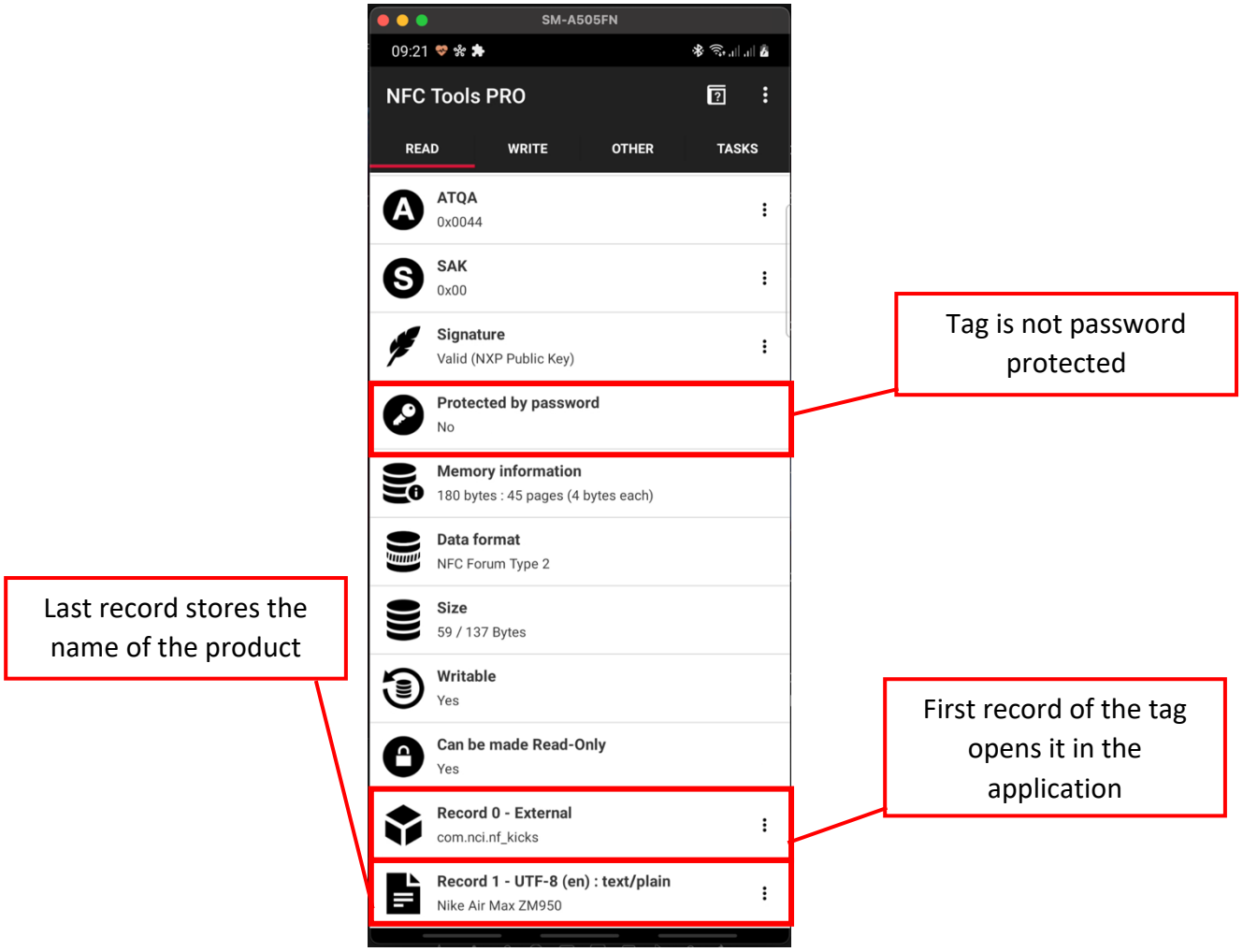
```
Future<void> _updateConnectionStatusAndCheckJailbreakOrRoot(
    ConnectivityResult result) async {
  final bool isRootedOrJailbroken = await TrustFall.isJailBroken;
  if (isRootedOrJailbroken == false) {
    _jailbreakOrRootStatusBool = false;
    if (result == ConnectivityResult.wifi ||
```

This shows us what directories the application is looking for in the android device. If anyone of these directories is present, the application will cease to function. The testing section will validate this feature.

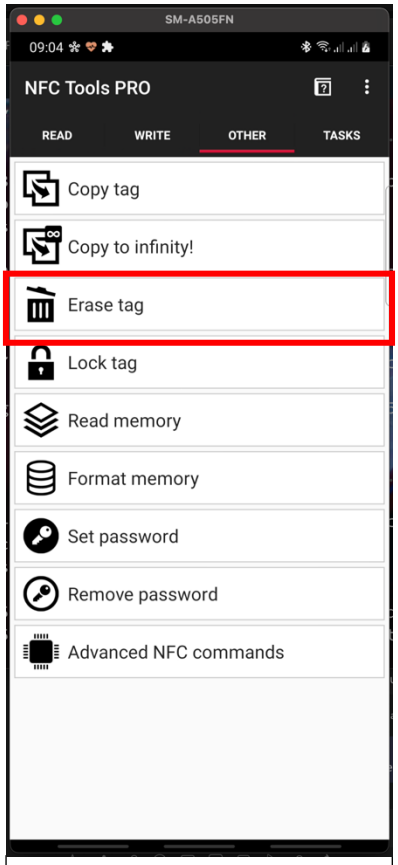
2.3.9. NFC Security

NFC tags are used in this application to allow for product discovery. This is accomplished by storing the product's name in the NFC tag. But, if adequate security measures are not in effect, the aforementioned tags may be modified for harmful purposes, such as redirecting the user to a malicious website.

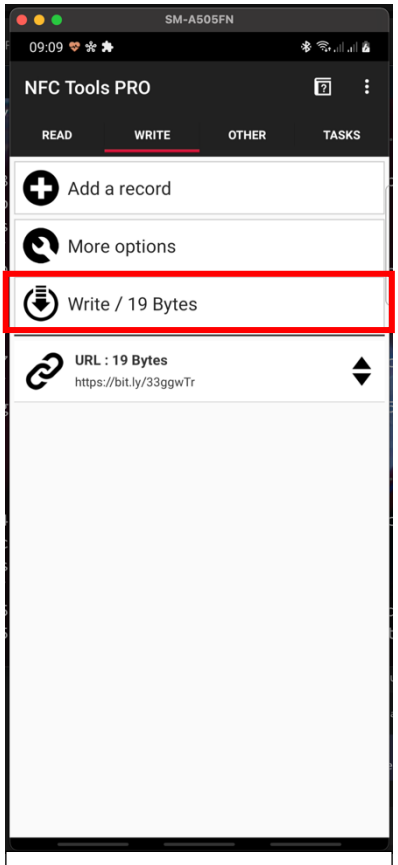
Using an Android application by the name of NFC Tools PRO, I am able to view information stored inside an NFC tag. Here is an example with an insecure NFC tag:



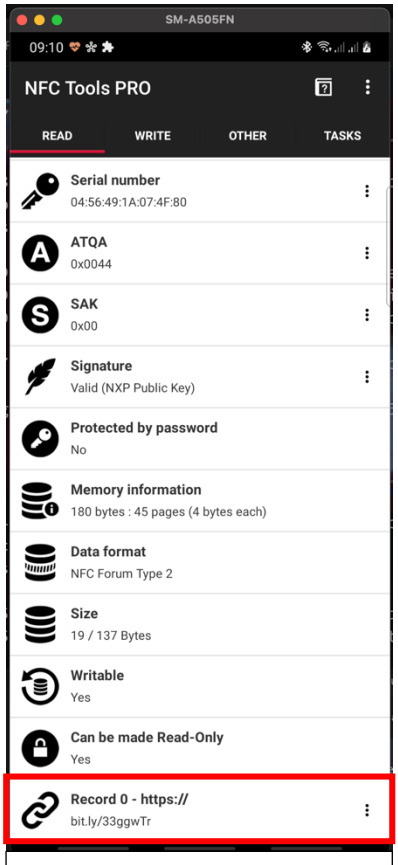
As can be seen here, the NFC tag contains two records; the first launches the Nf_Kicks application, while the second contains the name of the product stored on the tag; but, as can be seen, the tag is not password protected. This is unsafe since anybody can overwrite the tag's contents to then delete it or redirect the victim to a malicious website. Here are the steps a malicious user would take to modify a tag.



Erase the tag

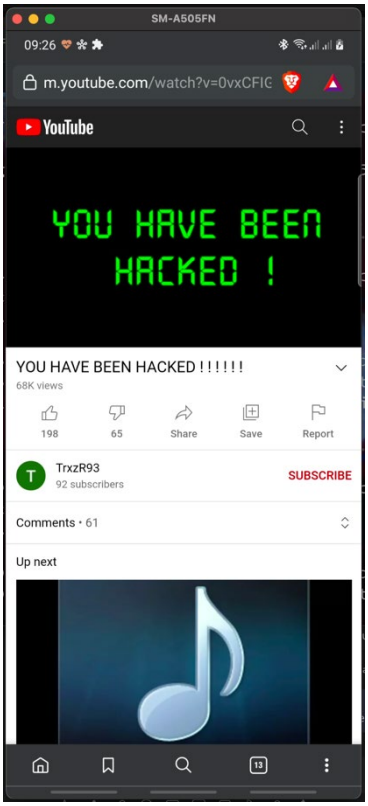


Write the malicious link to the tag

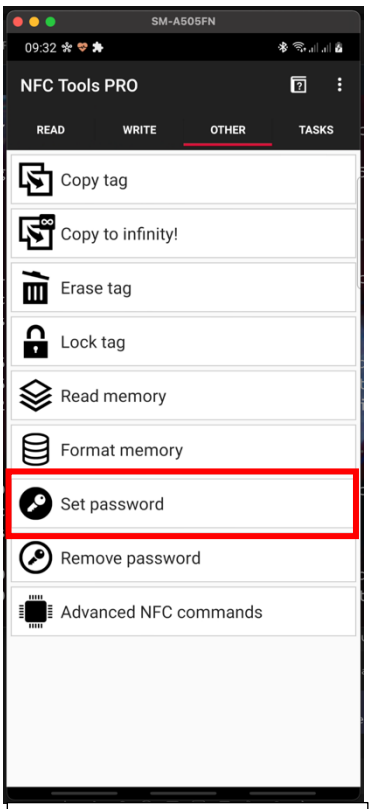


Tag has been modified

Now if a victim scans the tag, they will be redirected to the following website:

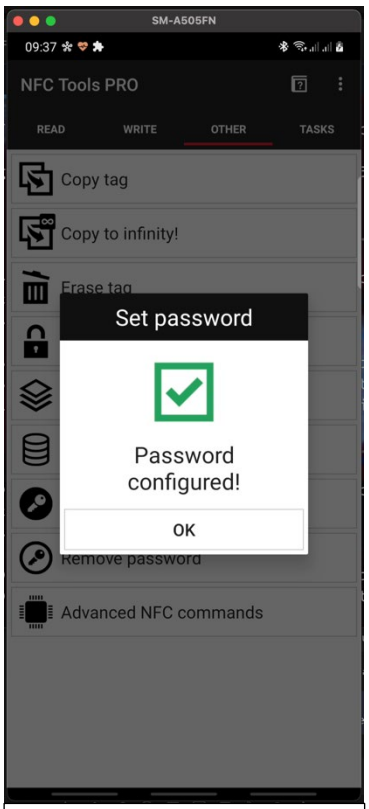


As you can see, this is the danger of having an insecure NFC tag, I will now demonstrate how a tag can be protected.



Tap Set password

Choose a secure password



Tap OK and scan a tag to configure

In the testing section, we can verify that the password has been set correctly and is functioning properly.

The following code manages NFC detection and scanning; the package used is named Flutter NFC reader. What happens here is that when a tag is approached, the product name recorded on the tag is retrieved, a database call is made to fetch the product's id from Cloud Firestore and the user is then navigated to the product's page in the application.

```
void readNFCProductTag(DatabaseApi databaseApi) {
  FlutterNfcReader.onTagDiscovered().listen((onData) async {
    final String _productCode = onData.content;
    final Product _productStream =
        await databaseApi.nfcProductStream(nfcCode: _productCode).first;
    final String _storeName =
        await databaseApi.storeName(storeId: _productStream.storeId).first;
    changePageForNFC(ProductPage(
      dataStore: databaseApi,
      productId: _productStream.id,
      storeName: _storeName,
    ));
  });
}
```

2.3.10.Security Rules – Cloud Firestore – Permission Checks

Permission checks are performed in Cloud Firestore security rules to deter unauthorized users from reading and writing data; in this section, I will detail how I protected the database's user collection from unauthorized read and write requests.

I've restricted the types of requests that can be made at the root of the users collection. As shown here, only create requests are permitted, and a user must be signed into their account in order for the request to succeed.

```
match /{users}/{document=**} {
  // Allow only create requests
  // Allow only logged in users to create a user account
  allow create: if request.auth.uid != null;
}
```

This rule is crucial since it allows for just create requests; if read requests were allowed at this level of the user collection, all user information would be compromised and freely available from the outside, resulting in a data breach.

The only way to access individual user information is to be logged in as the individual. This will be apparent in the next security rule because it requires a userId to be present in order for the request to be successful.

```
match /{users}/{userId} {
  // Allow only logged in users to perform create requests
  // and the user
  allow create: if request.auth.uid != null && checkIfUserOwnsResource(userId);

  // Only perform get when the user is logged
  // in and they own the resource
  allow get: if checkIfUserIsLoggedInAndUserIsVerified()
  && checkIfUserOwnsResource(userId);

  // Only update User Information when
  // the user is logged in, verified,
  // has specific keys, they own the
  // resource and key values must have
  // specific datatype
  allow update: if checkUserKeys()
  && checkKeyValuesInUsersDatatypes()
  && checkIfUserIsLoggedInAndUserIsVerified()
  && checkIfUserOwnsResource(userId);

  // Perform delete when user owns the resource and
  // are logged in and verified
  allow delete: if checkIfUserIsLoggedInAndUserIsVerified()
  && checkIfUserOwnsResource(userId);
}
```

As shown in this picture, this is a list of actions that may be performed on an individual user's account; any request that makes use of one of these actions requires a user to be signed in; otherwise, the request will be denied. These security rules make it extremely difficult for an attacker to request a user's behaviour without access to the user's account.

To further understand how this security rule works, we must understand the functions being called. The following function’s purpose is to check if the user making the request is both logged in and verified.

```
// If the request.auth.uid is null the action will fail
// If the request.auth.uid is not equal to
// the resource uid (resource.data.uid)
// that we are trying to perform an action on it will fail
function checkIfUserIsLoggedInAndUserIsVerified() {
  return (request.auth.uid != null)
  && (request.auth.token.email_verified);
}
```

This means that when a request is made to perform an operation on the database, it must be sent by a logged-in and verified user or the requests will be rejected.

The next function to take a look at checks if the user is the owner of the resource in which they are attempting to perform an operation on.

```
// Check that the user owns the resource
function checkIfUserOwnsResource(userId) {
  return request.auth.uid == userId;
}
```

This checks if the current logged in userId is equal to the userId of the collection they are intending to perform an operation on.

2.3.11.Security Rules – Cloud Storage – Permission Checks

Apart from the input validation function listed previously, there is another rule in my Cloud Storage security rules that conducts permission checks on requests submitted to the server, ensuring that not just anybody can access the files present.

```
// Allow a request only when the
// request was sent by a logged in user
// and when the user verified
allow get: if request.auth.uid != null && request.auth.token.email_verified;
```

The following security rule verifies that the request originated from a logged-in user; if any of these checks fails, the request is rejected.

2.3.12.Errors Handling

Inadequate error management, or the lack thereof, has the ability to introduce a variety of security issues into an application. The most serious problem happens when an attacker is shown unique internal error messages, such as stack traces or error codes, which expose implementation details to the outside world, supplying hackers with visibility into potential application vulnerabilities. In this section, I will go over the evidence of error handling in my project and how it relates to how the application is secured.

2.3.12.1. Payments.dart

```
static Future<StripeTransactionResponse> payWithCard({String amount}) async {
  try {
    final PaymentMethod paymentMethod =
        await StripePayment.paymentRequestWithCardForm(
            CardFormPaymentRequest());
    final Map<String, dynamic> paymentIntent =
        await _createPaymentIntent(amount);
    final PaymentIntentResult response =
        await StripePayment.confirmPaymentIntent(
      PaymentIntent(
          clientSecret: paymentIntent['client_secret'].toString(),
          paymentMethodId: paymentMethod.id),
    );
    if (response.status == 'succeeded') {
      return StripeTransactionResponse(
          message: 'Transaction successful', success: true);
    } else {
      return StripeTransactionResponse(
          message: 'Transaction failed', success: false);
    }
  } on PlatformException catch (error) {
    return _getPlatformExceptionError(error);
  } catch (error) {
    return StripeTransactionResponse(
        message: 'Transaction failed: ${error.toString()}', success: false);
  }
}
```

As can be seen here, the payWithCard function utilizes a try-catch block that catches two types of important errors that the Stripe Payment API may generate. The first is a PlatformException that is generated by the Stripe dependency in the project and is triggered when there's an error with either the developer's code or the package itself. The second catch block takes the error from the HTTP response generated by Stripe's API. In this case, if Stripe runs into an error while making a transaction with the card, this catch block will be executed. These catch blocks are important because the user needs to be informed about the status of their transaction. Without them, the user would not know if the transaction went through or not.

2.3.12.2. Authentication.dart

```dart
@override
Future<User> loginWithEmailAndPassword(String email, String password) async {
  try {
    final UserCredential userCredential =
        await _firebaseAuth.signInWithCredential(
      EmailAuthProvider.credential(
          email: email,
          password: EndToEndEncryption.hash(password: password)),
    );
    return userCredential.user;
  } on FirebaseAuthException catch (e) {
    throw FirebaseAuthException(
      code: 'FIREBASE_LOGIN_ERROR',
      message: FirebaseAuthExceptionHandler.handleException(e),
    );
  } on PlatformException catch (e) {
    throw PlatformException(code: e.code, message: e.message);
  }
}
```

This function handles logging a user into their Nf_Kicks account using Firebase and, as can be seen, with the use of two catch blocks, a few errors can be generated here. The first exception being caught is a FirebaseAuthException to indicate to the user that there may be an issue with their credentials or their account on Firebase itself. The second catch block is a PlatformException responsible for generating errors related to the Firebase dependency.

2.3.13.Code Quality – Secure Coding

Due to the Flutter framework being written in Dart, there is already a static analysis tool built in, as can be seen here:

```
'showSnackBar' is deprecated and shouldn't be used. Use ScaffoldMessenger.showSnackBar. This feature w [nf_kicks] lib/pages/store/product_page.dart:356
'FlatButton' is deprecated and shouldn't be used. Use TextButton instead. See the migration guide in flutt... [nf_kicks] lib/widgets/show_alert_dialog.dart:21
'FlatButton' is deprecated and shouldn't be used. Use TextButton instead. See the migration guide in flutt... [nf_kicks] lib/widgets/show_alert_dialog.dart:23
'FlatButton' is deprecated and shouldn't be used. Use TextButton instead. See the migration guide in flutt... [nf_kicks] lib/widgets/show_alert_dialog.dart:30
Unused import: 'package:flutter/foundation.dart'.                                                           [nf_kicks] lib/widgets/text_constants.dart:3
The parameter 'price' is required.                                                                          [nf_kicks] test/cartitem_test.dart:163
Unused import: 'package:cloud_firestore/cloud_firestore.dart'.                                              [nf_kicks] test/order_test.dart:2
Unused import: 'package:nf_kicks/models/product.dart'.                                                      [nf_kicks] test/order_test.dart:7
The parameter 'storeId' is required.                                                                        [nf_kicks] test/product_test.dart:170
```

This tool is capable of telling developers whether the code they are writing adheres to Dart programming standards provided by Google. Nevertheless, this tool can be enhanced by including a Dart linting tool capable of statically analysing the Dart-code in real-time. This is accomplished by the usage of the linting tool named lint, which is capable of detecting security bugs, stylistic, and compile-time errors in code. Here is an example of suggestions:

```
Avoid `print` calls in production code.                                                                    [nf_kicks] lib/main.dart:109
Avoid `print` calls in production code.                                                                    [nf_kicks] lib/main.dart:115
Unnecessary new keyword.                                                                                   [nf_kicks] lib/models/order.dart:32
Unnecessary new keyword.                                                                                   [nf_kicks] lib/models/order.dart:40
Unnecessary new keyword.                                                                                   [nf_kicks] lib/models/order.dart:42
'List' is deprecated and shouldn't be used. Use a list literal, [], or the List.filled constructor instead. [nf_kicks] lib/models/order.dart:43
Unnecessary new keyword.                                                                                   [nf_kicks] lib/models/order.dart:43
Use collection literals when possible.                                                                     [nf_kicks] lib/models/order.dart:43
Avoid bool literals in conditional expressions.                                                            [nf_kicks] lib/models/order.dart:52
Avoid bool literals in conditional expressions.                                                            [nf_kicks] lib/models/order.dart:53
Type annotate public APIs.                                                                                 [nf_kicks] lib/models/order.dart:66
Avoid bool literals in conditional expressions.                                                            [nf_kicks] lib/models/product.dart:42
```

As can be seen here, there is a difference in information provided as these are more related to flaws that can be exploited in production code rather than only general clean code practices. These code suggestions listed are based on the following documentation.

I'll now go through some code suggestions that stood out to me in terms of secure coding and clarify why writing code in this manner could be critical for overall application security.

2.3.13.1. avoid_print

```
Widget _buildGoogleMapsWithMarkers() {
  final database = Provider.of<DatabaseApi>(context, listen: false);

  return Stack(
    children: [
    StreamBuilder<List<Store>>(
      stream: database.storesStream(),
      builder: (context, snapshot) {
        Set<Marker> _markers = <Marker>{};

        if (snapshot.hasError) {
          print("Errors: ${snapshot.error}");
          return kLoadingLogo;
        }
```

The following block of code is currently printing errors to the console for debugging purposes. Nevertheless, this type of code should not appear in production builds of the application since it exposes implementation details and information about the application to the console, which an attacker may use to manipulate bugs in the application. To improve this code and enhance security, it must be removed entirely.

```
builder: (context, snapshot) {
  Set<Marker> _markers = <Marker>{};

  if (snapshot.hasError) {
    return kLoadingLogo;
  }
```

2.3.13.2. deprecated_member_use

```
: OutlineButton(
    borderSide: BorderSide(color: Colors.black),
    onPressed: () => _toggleResetPasswordFormType(),
    child: Text(
      "Reset Password".toUpperCase(),
      style: TextStyle(
        color: Colors.black,
        fontWeight: FontWeight.bold,
      ),  // TextStyle
    ),  // Text
  ),  // OutlineButton
```

The following code has been deprecated, which means it will no longer be supported by the Flutter development team. may be deprecated for a variety of reasons, one of which is because the feature might be unsafe to use, and if the code is not updated, that can result in a security flaw in the codebase that an attacker may exploit if left unchecked. To update the code we must do the following.

```
ButtonTheme(
  child: OutlinedButton(
    style: ButtonStyle(
      backgroundColor: MaterialStateProperty.all<Color>(
        Colors.white,
      ),
      side: MaterialStateProperty.all<BorderSide>(
        const BorderSide(),
      ),
    ),  // ButtonStyle
    onPressed: () => _toggleResetPasswordFormType(),
    child: Text(
      "Reset Password".toUpperCase(),
      style: const TextStyle(
        color: Colors.black,
        fontWeight: FontWeight.bold,
      ),  // TextStyle
    ),  // Text
  ),  // OutlinedButton
),  // ButtonTheme
```

As can be seen, there is no longer an issue with this code.

2.3.13.3. always_declare_return_types

```
main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await dot_env.load(fileName: ".env");

  const bool isProduction = bool.fromEnvironment('dart.vm.product');
  if (isProduction) {
    debugPrint = (String message, {int wrapWidth}) {};
  }

  runApp(App());
}
```

Since the function lacks a return type, the linter is unable to properly analyse the code for bugs and errors. To correct the code we do the following.

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await dot_env.load(fileName: ".env");

  const bool isProduction = bool.fromEnvironment('dart.vm.product');
  if (isProduction) {
    debugPrint = (String message, {int wrapWidth}) {};
  }

  runApp(App());
}
```

2.3.13.4. valid_regexps

```
@override
Future<void> deleteCartItem({String cartItemId, String storeCartName}) async {
  final String storeNameCart =
      "${storeCartName.toLowerCase().replaceAll(RegExp(r"(\s+"), "")}Cart";
  final path = APIPath.storeCartItem(uid, storeNameCart, cartItemId);
  final documentReference = FirebaseFirestore.instance.doc(path);
  await documentReference.delete();
}
```

The improper usage of a regular expression in an application has the potential to break critical functionality, and in this function, it may assign a storeNameCart incorrectly, which can have a ripple effect through the whole application if not addressed properly. We must use a proper regular expression to correct our code.

```
@override
Future<void> deleteCartItem({String cartItemId, String storeCartName}) async {
  final String storeNameCart =
      "${storeCartName.toLowerCase().replaceAll(RegExp(r"\s+"), "")}Cart";
  final path = APIPath.storeCartItem(uid, storeNameCart, cartItemId);
  final documentReference = FirebaseFirestore.instance.doc(path);
  await documentReference.delete();
}
```

2.3.13.5. empty_catches

```
Future<void> initConnectivity() async {
  ConnectivityResult result;

  try {
    result = await Connectivity().checkConnectivity();
  } on PlatformException catch (e) {}

  if (!mounted) {
    return Future.value();
  }

  return _updateConnectionStatusAndCheckJailbreakOrRoot(result);
}
```

Here we have a missing catch block. The catch block is required because we don't want the exception to be shown on the console. To fix this code we must do the following.

```
Future<void> initConnectivity() async {
  ConnectivityResult result;

  try {
    result = await Connectivity().checkConnectivity();
  } on PlatformException catch (_) {}

  if (!mounted) {
    return Future.value();
  }

  return _updateConnectionStatusAndCheckJailbreakOrRoot(result);
}
```

2.3.13.6. test_types_in_equals

```
@override
bool operator ==(Object other) {
  if (identical(this, other)) return true;
  if (runtimeType != other.runtimeType) return false;
  final Store otherStore = other as Store;
  return id == otherStore.id &&
      name == otherStore.name &&
      description == otherStore.description &&
      address == otherStore.address &&
      image == otherStore.image &&
      storeImage == otherStore.storeImage &&
      inStorePickup == otherStore.inStorePickup &&
      inStoreShopping == otherStore.inStoreShopping &&
      latLong == otherStore.latLong;
}
```

When we run the code above, we may encounter a null pointer exception; to change the code, we do the following.

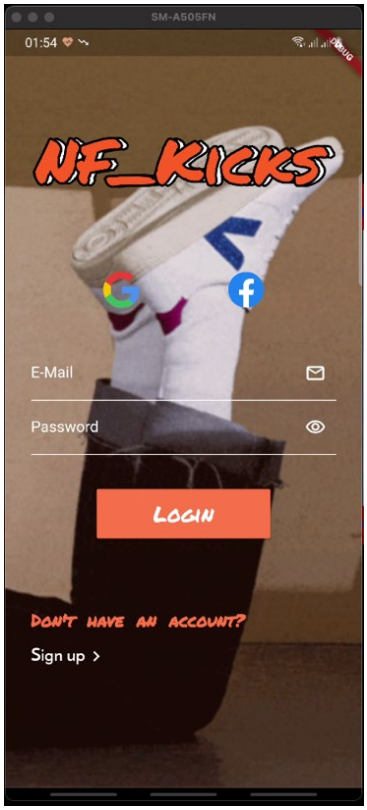
```dart
@override
bool operator ==(Object other) {
  if (identical(this, other)) return true;
  if (runtimeType != other.runtimeType) return false;
  final Object otherStore = other;
  return otherStore is Store &&
      id == otherStore.id &&
      name == otherStore.name &&
      description == otherStore.description &&
      address == otherStore.address &&
      image == otherStore.image &&
      storeImage == otherStore.storeImage &&
      inStorePickup == otherStore.inStorePickup &&
      inStoreShopping == otherStore.inStoreShopping &&
      latLong == otherStore.latLong;
}
```

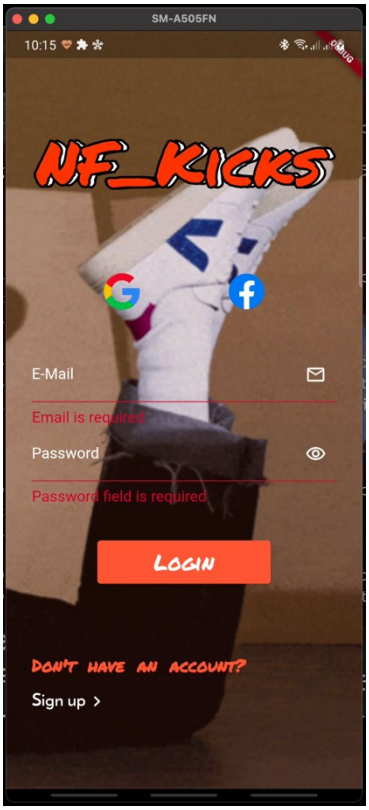
2.4. Graphical User Interface (GUI)

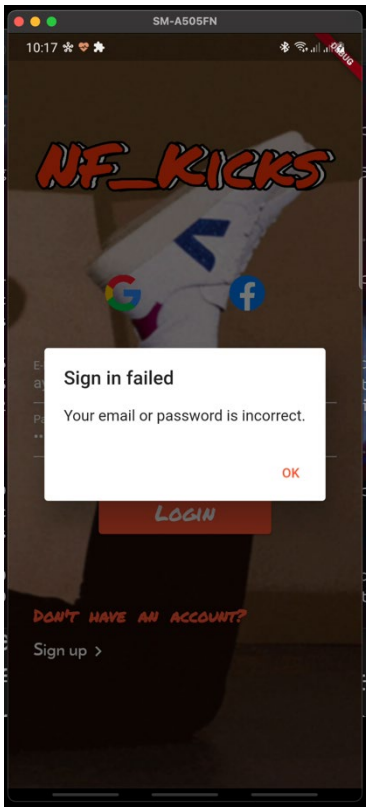
2.4.1. Login Screen

This is the login screen and the first screen that is shown to a new Nf_Kicks user. There are mainly four options present in the GUI. Two of which are alternative Login options for the user to choose from in Google and Facebook sign-in. Next is the option to login with credentials of your own. The fourth option is the sign-up or register for an account button.

This page also has the ability to generate errors when credentials are incorrect or there is an issue in the back end.

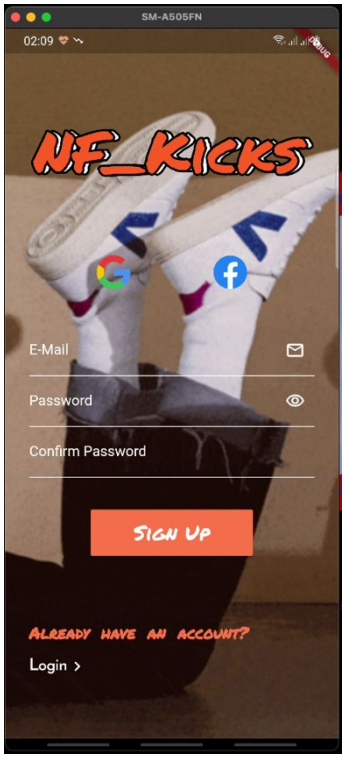


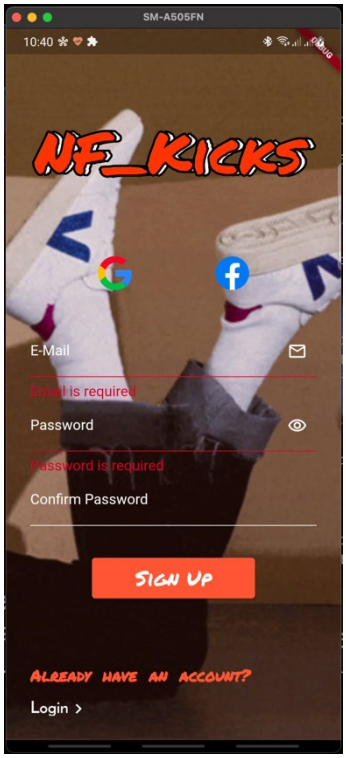


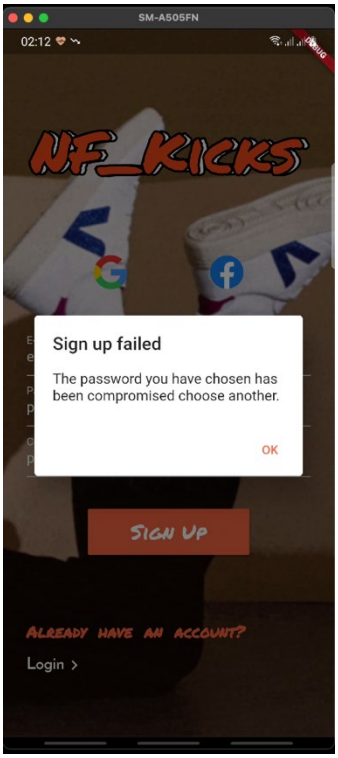


2.4.2. Registration Screen

The registration screen performs validation on the client-side to indicate to the user if their email or password is not valid. Validation performed on the server-side is sent to the client to be read by the user in the example of a compromised password.

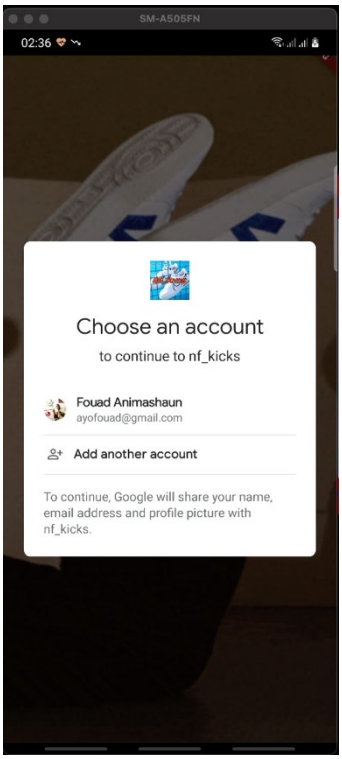






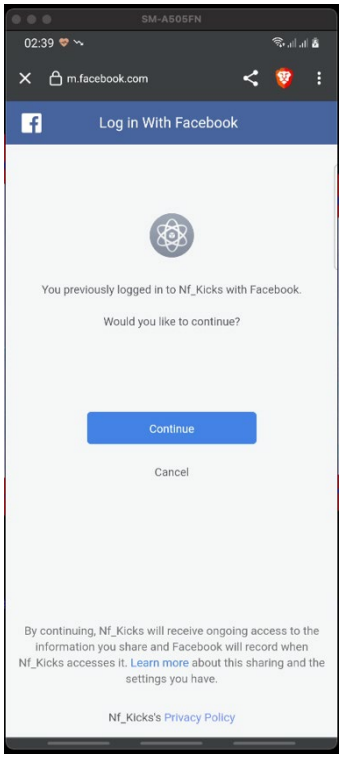
2.4.3. Login With Google

This page allows the user to choose their Google account or sign in with an existing one.



2.4.4. Login With Facebook

If a user chooses their Facebook account, they will be required to login to their account via a WebView within the application.

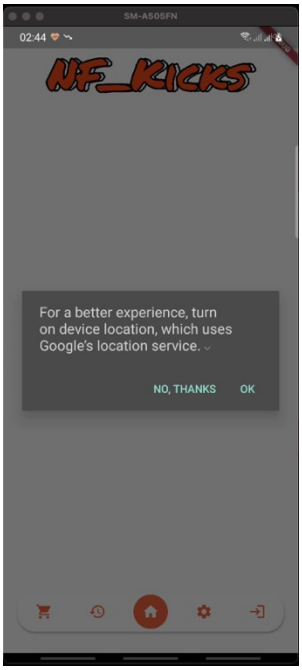


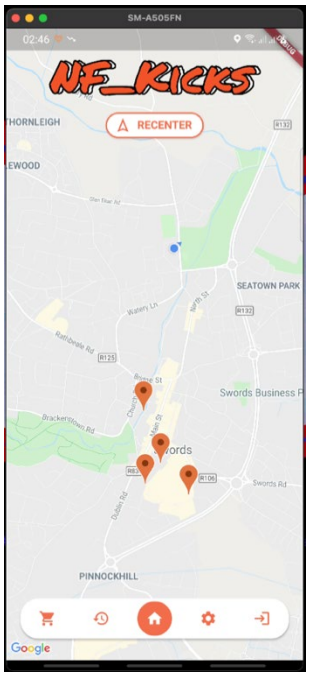
2.4.5. Verify Email Address

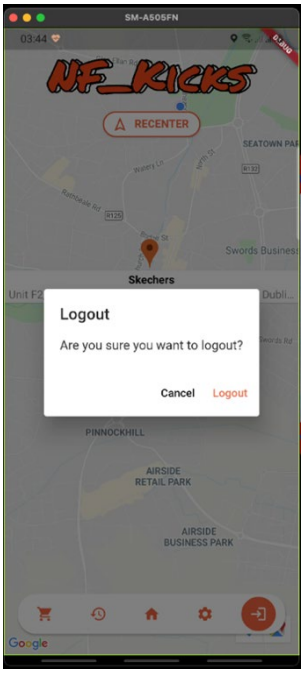
This screen is responsible for telling users to check their email once their Nf_Kicks account has been successfully made. There is a back button present to help the user go back to the login page.

2.4.6. Landing Screen

Upon landing on this screen, the user is asked for their device's location to enhance the in-app experience. The main screen has various pages that a user can navigate to as can be seen below and a re-center button in case the user loses their location. When a marker on the map is tapped, a user receives some information about the store along with a link to said store.

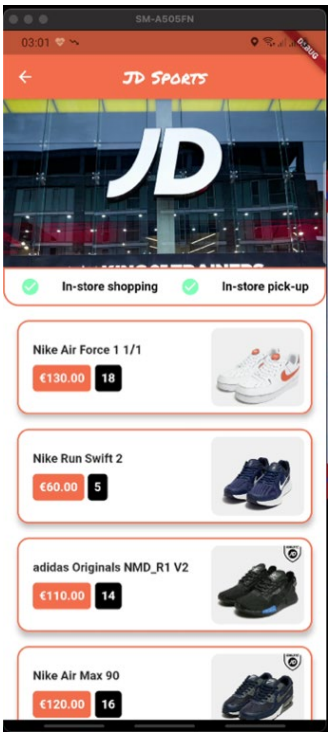






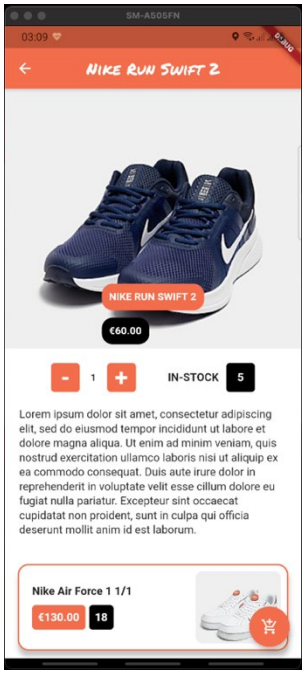
2.4.7. Store Screen

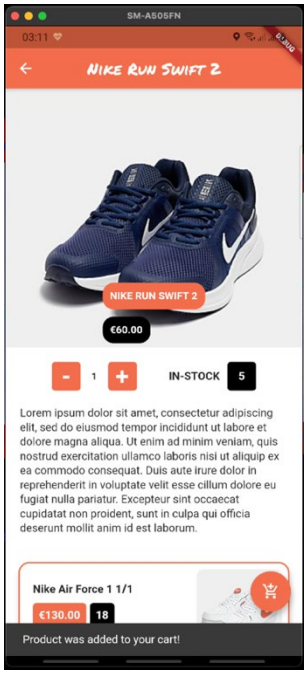
When a user enters the store screen, they are presented with a list of products to choose from as well as additional store information such as whether the store is open or closed.



2.4.8. Product Screen

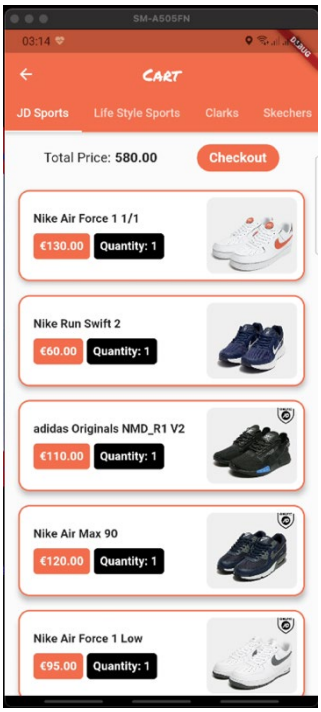
As shown below, the product screen provides extra product details to the buyer, as well as the ability to increase or decrease the amount of the product. When the customer is satisfied, they may add the product to their cart. When a product is added to the cart, the customer is notified that the item has been added to their cart. Additionally, related items that are similar to this one are shown.





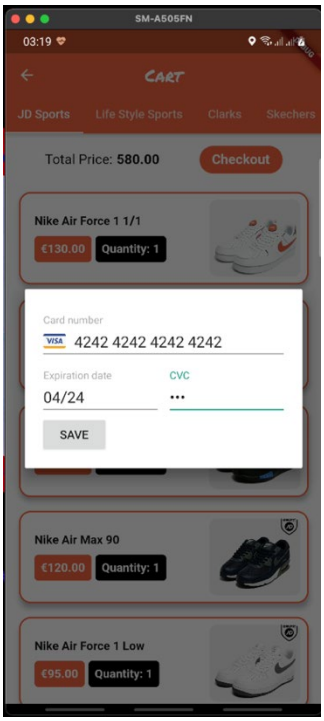
2.4.9. Cart Screen

The cart screen displays all of the items that the customer has added to their cart for checkout, along with the total amount.



2.4.10. Checkout Screen

The customer will pay for their products here using a legitimate credit card. However, in the picture, I used a test card. If the transaction is successful, the buyer will be provided with the transaction details.

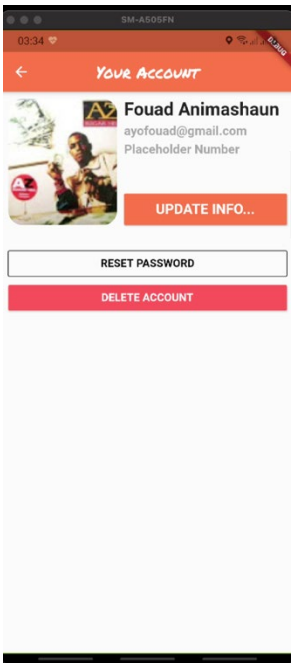


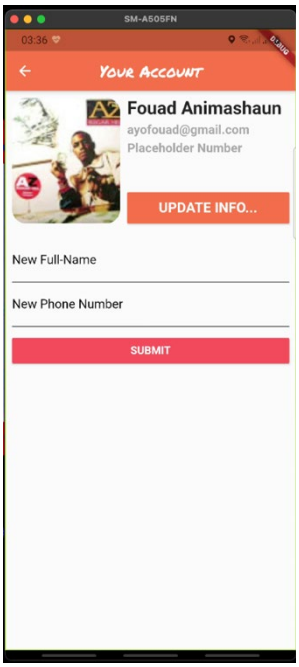
2.4.11. Order Screen

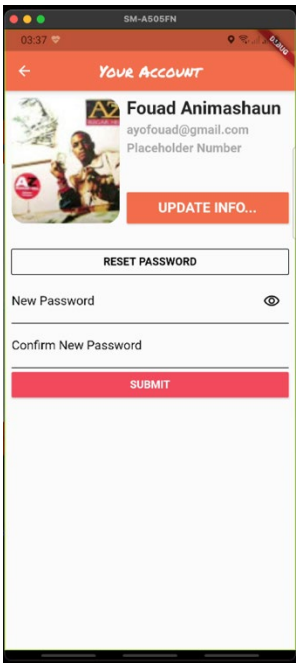
The order screen holds a QR code that identifies the user's order. The order's current status can be checked by tapping on the icon present on the screen.

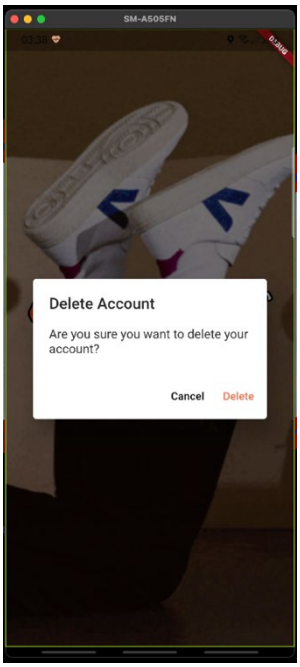
2.4.12.Settings Screen

The Setting screen is home to a great deal of functionality. This includes updating information, resetting the password, deleting an account and uploading a profile picture.









2.4.13.Profile Upload

Here, a user can choose an image from either their gallery or camera as an image to upload. This image can also be cropped.

2.5. Testing

2.5.1. Security Testing

2.5.1.1. Mobile-Security-Framework

Mobile Security Framework (MobSF) is a suite of automated penetration-testing tools for iOS and Android apps to conduct static and dynamic analysis on mobile binary files. Here is an example of the type of output that MobSF is able to generate from the debug build of the android application.

STATUS	DESCRIPTION
secure	Application is signed with a code signing certificate
warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android <7.0
bad	Application signed with a debug certificate. Production application must not be shipped with a debug certificate.
warning	Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues. The manifest file indicates SHA256withRSA is in use.

As can be seen here, I was sent details about the application's signing certificate and a warning about the Android minimum SDK. A developer may extrapolate from what this tooling indicates and conclude that the Android minimum SDK must be increased to defend against the Janus vulnerability and that the application's signing certificate should be converted from debug to release/production.

When recommendations are adhered to, the response from MobSF should look like this.

STATUS	DESCRIPTION
secure	Application is signed with a code signing certificate

The suite is also able to identify areas that are secured in my application, such as these classes that perform root detection.

NO	ISSUE	SEVERITY	STANDARDS	FILES
4	This App may have root detection capabilities.	secure	CVSS V2: 0 (info) OWASP MASVS: MSTG-RESILIENCE-1	c/b/a/d/b.java c/e/a/b/e/j/l2.java c/b/a/d/c.java com/scottyab/rootbeer/b.java

As well as areas that need improvement.

9	SHA-1 is a weak hash known to have hash collisions.	high	CVSS V2: 5.9 (medium) CWE: CWE-327 Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	c/h/a/c1.java

Another point to mention is in the files that have been identified. Normally, classes in applications are defined as "com/nci/nf_kicks/widgets/background_stack.dart," but with the use of code obfuscation, the files and directories are encrypted, making it difficult for

the tool to find more information about the issue, which is why it identifies files with a single letter instead of listing them. This tool is incredibly useful as it is able to identify the flaws and areas that need to be fixed in the application.

2.5.1.2. Testing Firebase Security Rules with Postman

When a developer forgets to implement security rules, an attacker is able to make arbitrary requests to the database that can result in either a data breach or the loss of vital application-related data. For example, when a get request is sent to the following endpoint https://firestore.googleapis.com/v1/projects/nfkicks-35620/databases/(default)/documents/users the database will not perform any validation before authorizing the reading of data. Here is what a valid response would look like:


```
{
  "documents": [
    {
      "name": "projects/nfkicks-35620/databases/(default)/documents/users/PrYKCIuAiBa8c6ZOENALJ9V2fWn2",
      "fields": {
        "image": {
          "stringValue": "7NXarghdYTOfyP08gV8wrNhhMucE33/
            HacQM6wCy2T1lH3QP1NLpj9H5DbJapMebXPvMyQZvTNvkKppXkfEziMPa6zDBHPwnuEDGcnGgB93l20FkPt8iTiBQCqSRiByYoXj07IBQDq8se5PAB96F5L5jdr7103CHDXnypgCIXCEMpGFYSyslzdy92b6pO4s6"
        },
        "has2FA": {
          "booleanValue": false
        },
        "email": {
          "stringValue": "48nHtBEPIXe5xuI4ilJtqsR4U5trthTnAKVtgmLZtkM="
        },
        "fullName": {
          "stringValue": "zs7GsFsjIXnxqYdR6zZLwQ=="
        },
        "phoneNumber": {
          "stringValue": "1M3PvR4PIXCdxP15rUsuq85nU5trthTnAKVtgmLZtkM="
        }
      },
      "createTime": "2021-02-27T12:10:27.080514Z",
      "updateTime": "2021-02-27T12:10:27.080514Z"
    },
```


Even though user information is encrypted, the attacker is able to read and potential crack this information in their own time. This security flaw can be taken to the next level because if the attacker would like to delete all the information in the database, the request would be executed due to there not being any protections in place.

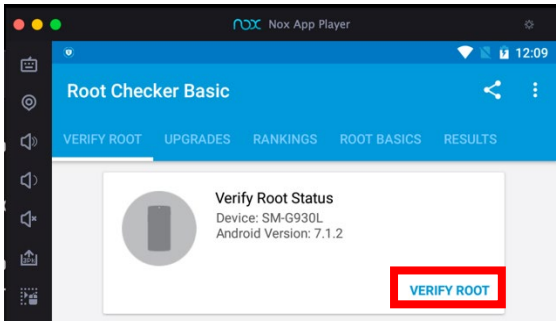
However, when security rules are implemented, the server will send the following response to the same request:

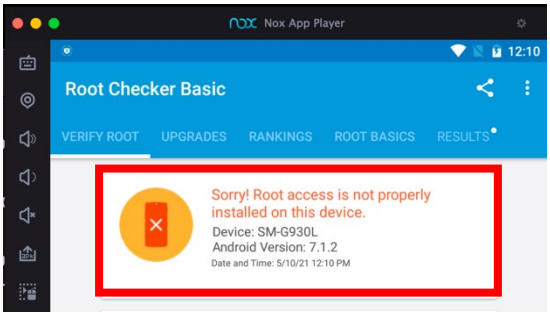

```
{
    "error": {
        "code": 403,
        "message": "Missing or insufficient permissions.",
        "status": "PERMISSION_DENIED"
    }
}
```


This is the desired response because when a request is made to the endpoint mentioned previously, we don't want it to read all the information from the user's collection. The only time a read request is allowed on the user's collection is when a logged-in user makes a request to the database with their UserId to retrieve their own information and only their own.

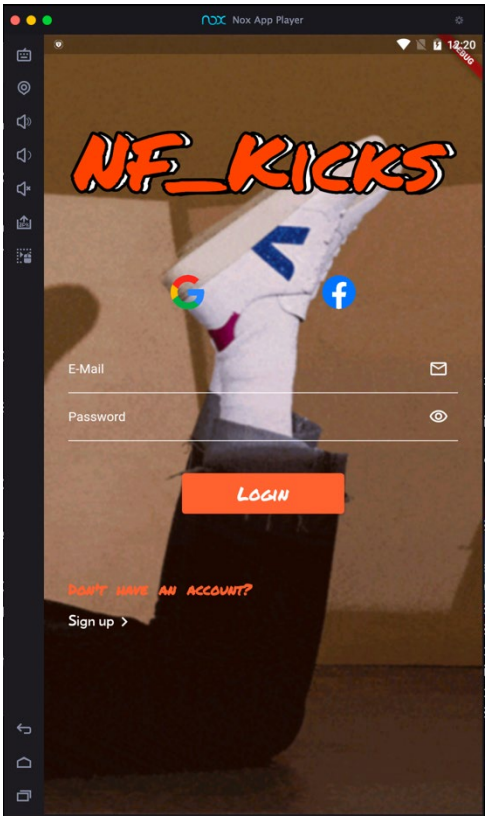
2.5.1.3. Testing rooted Android device detection

To verify the Android root detection capability, I used a tool called Nox Player, which is an Android emulator with toggleable root capabilities. Using a root checker application, I will first show that the device is indeed not rooted. Root Checker is the application I'm using.



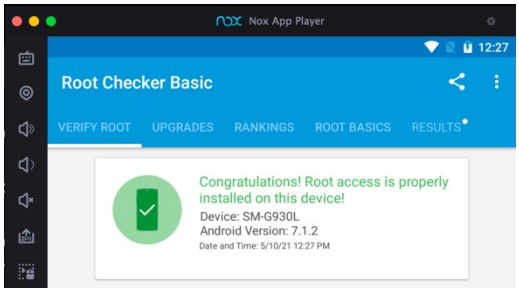


Because rooting-related binary are missing, the application works as intended on a non-rooted Android device, as evidenced by the console output on the right.




```
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /sbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /su/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/.ext/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/failsafe/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/sd/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/usr/we-need-root/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /cachesu Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /datasu Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /devsu Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /data/local/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /sbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /su/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/.ext/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/bin/failsafe/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/sd/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/usr/we-need-root/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /system/xbin/su Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /cachesu Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /datasu Absent :(
I/RootBeer( 5698): LOOKING FOR BINARY: /devsu Absent :(
```

Now I'm going to root the device and verify that the application works as intended. As seen in the root checker application, the device is indeed rooted, resulting in the application ceasing to function. The application recognizes that the device is rooted due to the discovery of the following binary.



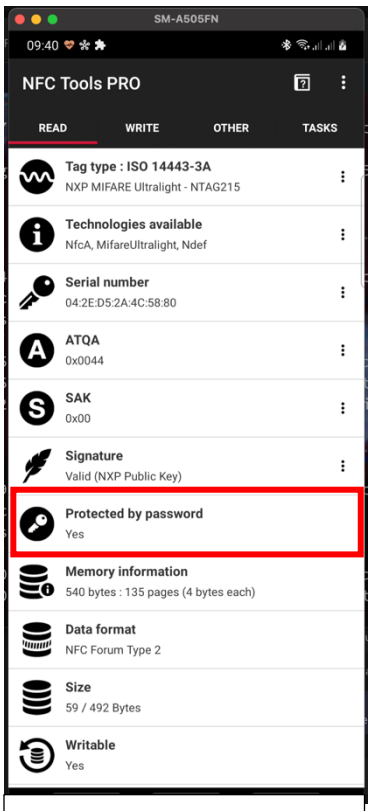

```
V/RootBeer( 3215): b: a() [184] - /system/xbin/busybox binary detected!
V/RootBeer( 3215): b: a() [184] - /system/xbin/busybox binary detected!
```

The appearance of the /system/xbin/busybox informs the application that the device it is running on is indeed rooted and will cease to function by displaying the following page to the user:

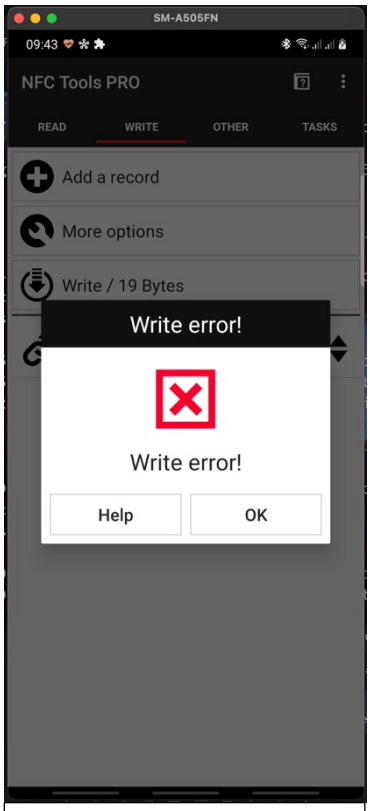
This test has demonstrated that the root detection functionality is working as intended.

2.5.1.4. Test password protected NFC tag

Once again, I'll be using NFC Tool Pro to determine if an attacker can modify the contents of an NFC tag that is password protected.



Tag is password protected

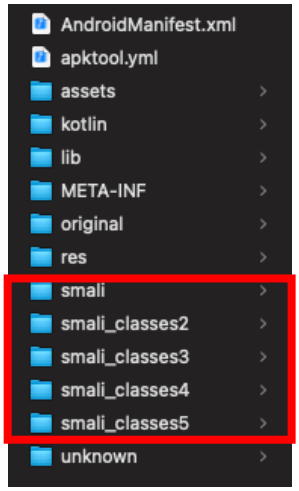


Now try to write to the tag

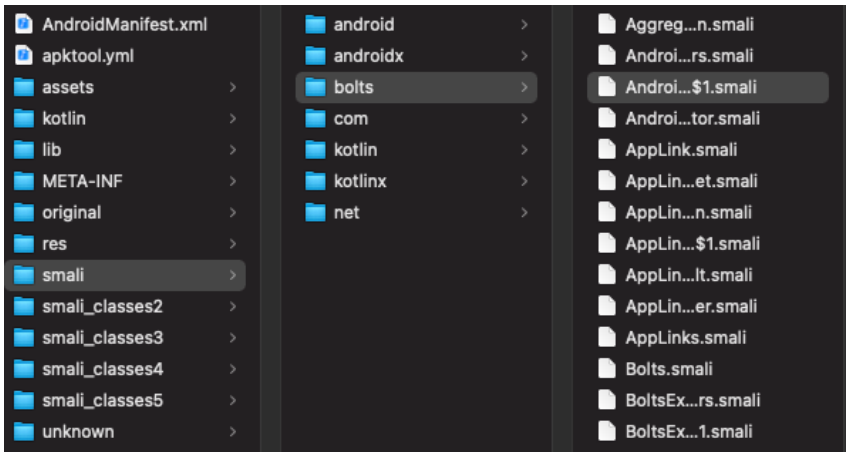
As can be seen here, the NFC tag can not be written on, resulting in it being secure against attackers.

2.5.1.5. Test for code obfuscation

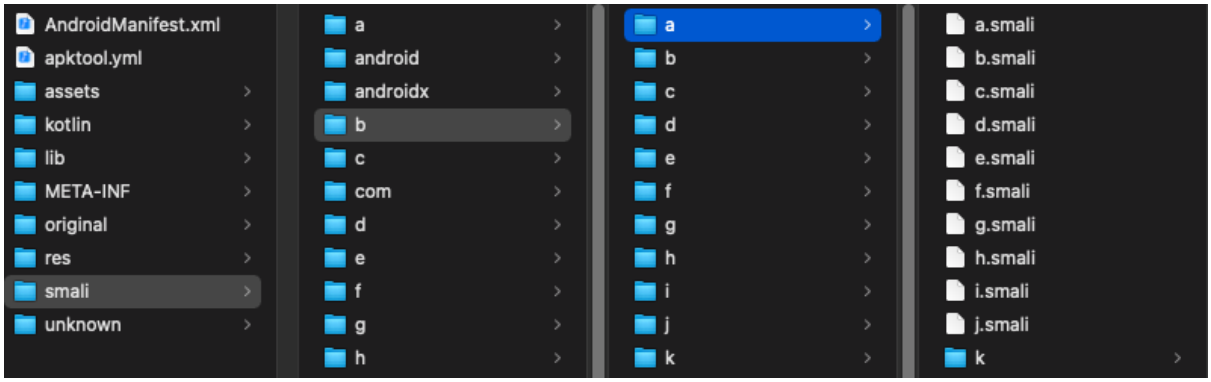
I will now begin to evaluate the presence of code obfuscation by comparing it to an un-obfuscated version of the application. To decompile the android application, I used an open-source tool named apktool that allowed me to view the contents of my application. To decompile, I used the following command: "apktool d app-debug.apk". Once decompiled, we will get the following files:



We can detect code obfuscation by inspecting one of the smali folders and determining whether the names of files and directories are human-readable and, in the case of our un-obfuscated decompiled app, is easily readable.



In an obfuscated application, the files look like this:



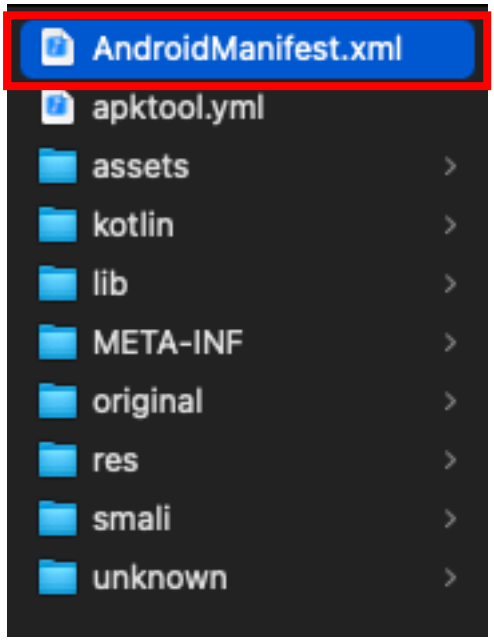
In this case, directories and files are not readable, rendering it more difficult for a hacker to comprehend and reverse engineer the application for harmful purposes. The application is able to interpret this obfuscated code through the use of mappings created by ProGuard.

```
# compiler: R8
# compiler_version: 1.5.64
# min_api: 21
# pg_map_id: 8cc8f7d
android.support.customtabs.ICustomTabsCallback -> a.a.a.a:
    void onRelationshipValidationResult(int,android.net.Uri,boolean,android.os.Bundle) -> a
    void extraCallback(java.lang.String,android.os.Bundle) -> b
    void onNavigationEvent(int,android.os.Bundle) -> b
    void onMessageChannelReady(android.os.Bundle) -> c
    void onPostMessage(java.lang.String,android.os.Bundle) -> c
android.support.customtabs.ICustomTabsCallback$Stub -> a.a.a.a$a:
android.support.customtabs.ICustomTabsService -> a.a.a.b:
    boolean mayLaunchUrl(android.support.customtabs.ICustomTabsCallback,android.net.Uri,android.os.Bundle,java.util.List) -> a
    boolean newSession(android.support.customtabs.ICustomTabsCallback) -> a
    boolean warmup(long) -> a
android.support.customtabs.ICustomTabsService$Stub -> a.a.a.b$a:
    android.support.customtabs.ICustomTabsService asInterface(android.os.IBinder) -> a
android.support.customtabs.ICustomTabsService$Stub$Proxy -> a.a.a.b$a$a:
    android.os.IBinder mRemote -> a
    boolean mayLaunchUrl(android.support.customtabs.ICustomTabsCallback,android.net.Uri,android.os.Bundle,java.util.List) -> a
    boolean newSession(android.support.customtabs.ICustomTabsCallback) -> a
    boolean warmup(long) -> a
```

These are mappings for the obfuscated code. This test establishes that the Nf_Kicks application is indeed obfuscated.

2.5.1.6. Check that no sensitive data is present in the Android Manifest file

When an attacker successfully decompiles the application, they may check the AndroidManifest.xml file for API keys that the developer forgot to hide. I will now visit the AndroidManifest.xml file to check if an API key is present. First, we open the folder to our decompiled application and open the AndroidManifest.xml file.

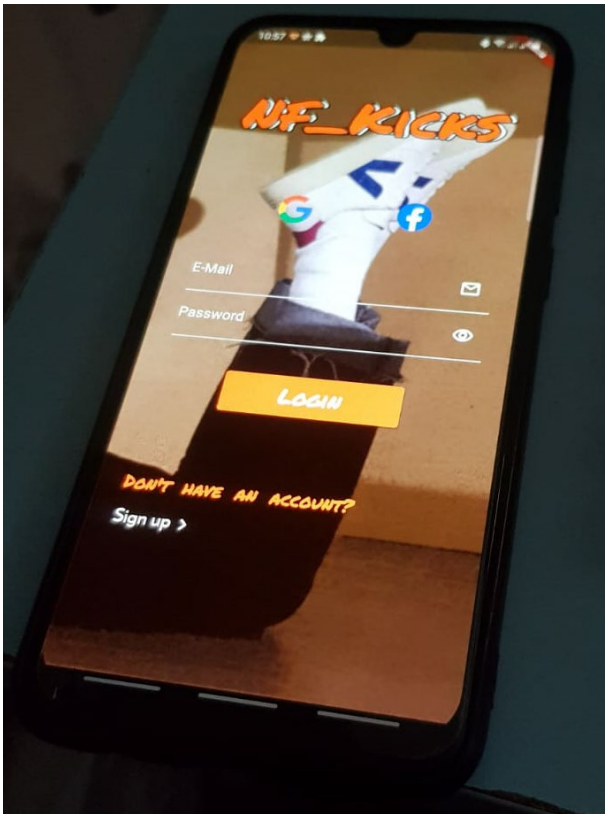


As shown in the following image below, rather than storing confidential data such as the Google Maps API Key in the Android Manifest file, the API key is protected by storing it in a location that is not readily accessible to an attacker, rendering it harder to find.

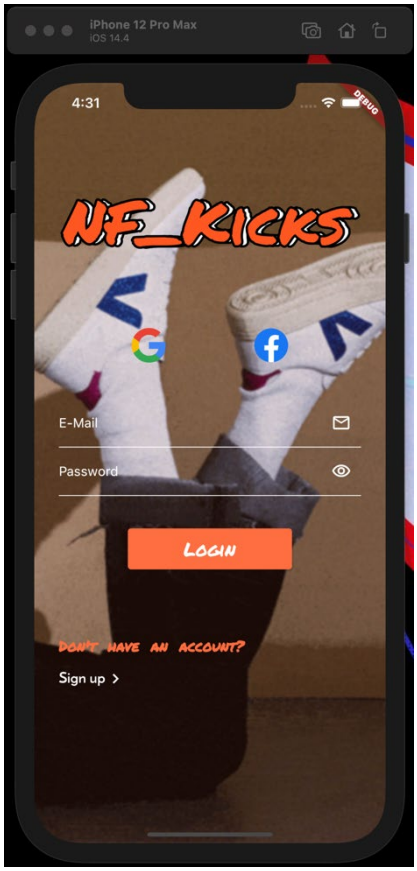
```
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:host="cct.com.nci.nf_kicks" android:scheme="fbconnect"/>
    </intent-filter>
</activity>
<meta-data android:name="flutterEmbedding" android:value="2"/>
<meta-data android:name="com.google.android.geo.API_KEY" android:value="@string/GOOGLE_MAPS_KEY"/>
<service android:directBootAware="true" android:exported="false" android:name="com.google.firebase.c
    <meta-data android:name="com.google.firebase.components:io.flutter.plugins.firebase.storage.Flut
    <meta-data android:name="com.google.firebase.components:io.flutter.plugins.firebase.firestore.Fl
```

2.5.2. System Testing

A system test checks to see if the application performs as expected and as planned. The purpose of the application is to run on both IOS and Android devices. In addition, said devices must be NFC enabled. Android 7.0 (Nougat) is the minimum SDK required to execute this application; any device running an SDK lower than Android 7.0 would be unable to run the application. The device that was used to test the application was a Samsung Galaxy A50 with the latest version of Android (Android 10).



Due to me not having access to an Apple developer account, I was, unfortunately, unable to test the application on a real apple device. However, the application is fully functional on an IOS simulator running IOS 13.



2.5.3. Integration Testing

Integration testing is a process in the software testing phase wherein the application's different features are integrated and evaluated as a cohesive unit. Integration testing was conducted during the implementation of the Firebase security rules that restricted access to data that the application and every other client may request from the database. When the application requested details about the current customer, integration checks verified that the requests were delivered to Cloud Firestore and that the responses were correct and compliant with security rules.

2.5.4. Unit Testing

The majority of unit tests that have been written mainly touch on the model classes present in the application because they are responsible for ensuring that the right data is received and transmitted to and from the database. For the Object.fromMap and Object.toMap functions, the tests that have been written are quite repetitive and mainly test the following

2.5.4.1. Test fromMap the function with null data:

```
test('test with null data', () {
  final product = Product.fromMap(null, 'documentId');
  expect(product, null);
});
```

2.5.4.2. Test fromMap function with a filled out object:

```
test('test with store object', () {
  final store = Store.fromMap({
    "name": "Nike Shoes",
    "description": "Lorem Ipsum",
    "address": "123 Main Street",
    "image": "URL Here",
    "storeImage": "URL Here",
    "inStorePickup": true,
    "inStoreShopping": true,
    "latLong": const GeoPoint(0, 0),
  }, 'documentId');  // Store.fromMap
  expect(
      store,
      Store(
        id: 'documentId',
        name: "Nike Shoes",
        description: "Lorem Ipsum",
        address: "123 Main Street",
        image: "URL Here",
        storeImage: "URL Here",
        inStorePickup: true,
        inStoreShopping: true,
        latLong: const GeoPoint(0, 0),
      ));  // Store
});
```

2.5.4.3. Test fromMap function with nulls in the object

```
test('test store object with nulls', () {
  final store = Store.fromMap({
    "storeId": null,
    "name": null,
    "description": null,
    "price": null,
    "stock": null,
    "image": null,
    "inStock": null,
    "latLong": null,
  }, 'documentId');
  expect(
      store,
      Store(
        id: 'documentId',
        name: "",
        description: "",
        address: "",
        image: "",
        storeImage: "",
        inStorePickup: false,
        inStoreShopping: false,
        latLong: const GeoPoint(0, 0),
      ));  // Store
});
```

2.5.4.4. Test fromMap function with missing key and value

```
test('test store object with missing key(s) and value(s)', () {
  final store = Store.fromMap({
    "name": "Nike Shoes",
    "description": "Lorem Ipsum",
    "address": "123 Main Street",
    "storeImage": "URL Here",
    "inStorePickup": true,
    "inStoreShopping": true,
    "latLong": const GeoPoint(0, 0),
  }, 'documentId');  // Store.fromMap
  expect(
      store,
      Store(
        id: 'documentId',
        name: "Nike Shoes",
        description: "Lorem Ipsum",
        address: "123 Main Street",
        image: "",
        storeImage: "URL Here",
        inStorePickup: true,
        inStoreShopping: true,
        latLong: const GeoPoint(0, 0),
      ));  // Store
});
```

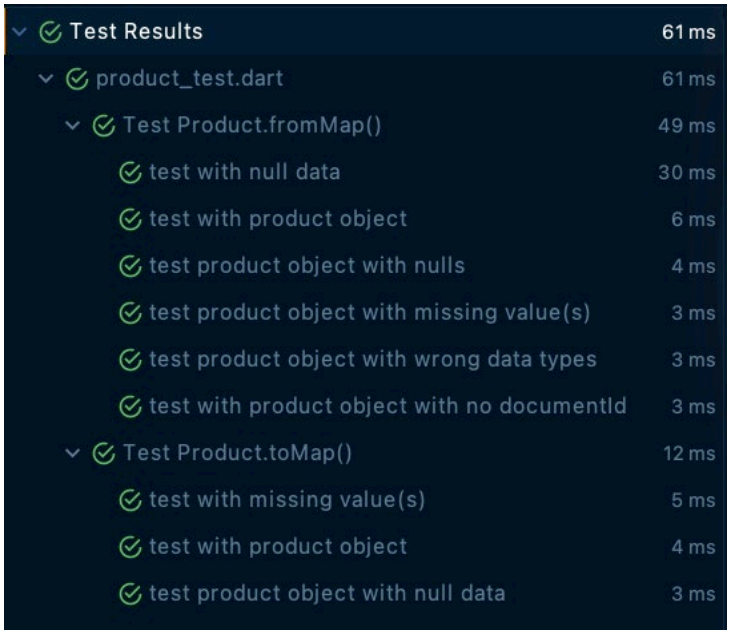
2.5.4.5. Test fromMap function with incorrect data types

```
test('test product object with wrong data types', () {
  final product = Product.fromMap({
    "storeId": 0.0,
    "name": 0.0,
    "description": 0.0,
    "price": "Twenty",
    "stock": "Four",
    "inStock": "might be true",
  }, 'documentId');
  expect(
      product,
      Product(
          id: 'documentId',
          storeId: "",
          name: "",
          description: "",
          price: 0.0,
          stock: 0,
          image: "",
          inStock: false));
});
```

2.5.4.6. Test fromMap function with no documentId

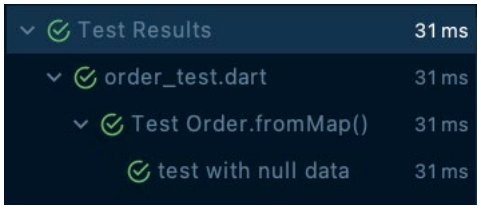
```
test('test with product object with no documentId', () {
  final product = Product.fromMap({
    "storeId": "abc",
    "name": "Nike Shoes",
    "description": "Lorem Ipsum",
    "price": 20.00,
    "stock": 4,
    "image": "url here",
    "inStock": true,
  }, null);
  expect(
      product,
      Product(
          id: null,
          storeId: "abc",
          name: "Nike Shoes",
          description: "Lorem Ipsum",
          price: 20.00,
          stock: 4,
          image: "url here",
          inStock: true));
});
```

2.5.4.7. Product model - Results

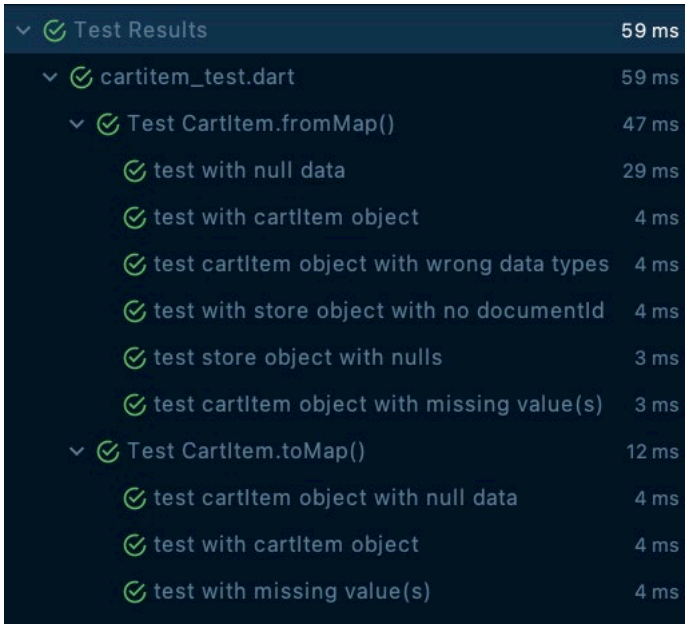


2.5.4.8. Store model - Results

2.5.4.9. Order model - Results



2.5.4.10. CartItem model - Results



2.6. Evaluation

Looking back on the testing completed to evaluate the application, it is safe to say that the goal of developing a secure e-commerce application was reached. Using the comprehensive road map used in the aims section, we are able to come to a conclusion on whether the four attributes listed were indeed secure by the application.

2.6.1. How the device is secured?

We ensured device security by first writing code that would detect the presence of a rooted/jailbroken mobile device running the application. This feature was then tested to ensure that the functionality was working as intended by taking note of the rooted device was in-capable of using the application.

To defend against vulnerabilities associated with older Android and IOS devices such as the Janus, the application was made to only be available on newer devices with the latest Android and IOS security patches.

2.6.2. How is content secured?

Content, such as a user's private information, was transmitted and stored in the database using cryptography to render it unreadable. Web requests including the user's private information were also encrypted using Secure Socket Layer (SSL) which was provided by default in the Flutter framework.

Credit card details were protected by the application's usage of the PCI-compliant Stripe Payments API; the application secured this card information not only by the use of Stripe

but also through ensuring that the data was not stored on the device or transferred to any server other than Stripe.

Additionally, content held in the cloud was demonstrated to be unavailable without the appropriate permissions by using Firebase Security rules to restrict access to unique resources on both unauthenticated and authenticated user accounts.

Finally, a password was also used to protect critical business content contained in NFC tags.

2.6.3. How is the application secured?

Unit tests were written to assure that model classes would function as intended when incorrect data and data types were used. Integration testing was done to check that the application worked with Firebase Authentication, Cloud Storage and Cloud Firestore along with testing that the application functioned on both IOS and Android devices.

The existence of code obfuscation to defend against reverse engineering attacks was demonstrated through the use of apktool to decompile the Android build of the application.

Secret keys were secured by not storing them directly in the application, but rather in a separate file to which they could be referred, making it difficult for an attacker to access them.

A linter was used and followed to assist in writing secure code in accordance with the Flutter development team's recommendations for preventing bugs and errors from occurring during application use.

2.6.4. How are identity and access secured?

When a customer chooses to create an account with Nf_Kicks, customer account security is prioritized from the beginning. This is in part due to the application's use of Firebase Authentication, an often audited account creation and management service owned by Google. The application itself ensures security by constantly validating the customer's chosen password and determining if the password has been used in previous data breaches. Additionally, as the password is transferred to a database, it is hashed using SHA 512 before transmission to preclude the password from being saved in plain text and to protect the customer's password in the event of it being intercepted. Lastly, authorization tests using Firebase security rules are used to protect access to consumer details.

3.0 Conclusion

The Nf_Kicks application was developed with an understanding of how to secure e-commerce applications, as shown by the application's use of a standardised password policy to ensure customers create an account with a safe and secure password. Additionally, the program protects the user's private details, such as passwords, email addresses, full names, and phone numbers, when in transit to the servers and while present on the mobile application. Additionally, the project adopts the use of security rules to ensure that proper

permission protocols are followed and that customers only have access to their data. However, the application has a few drawbacks. The first is that the API keys stored in the ENV file should not be stored in the application. This may have secured the application against an attacker with less experience, but a skilled hacker would be able to extract the given enough time.

Another disadvantage is that the application is an e-commerce application that required the customer to re-enter their credit card information every they choose to checkout though the reasons for doing are well-founded, a better solution would be to use Stripe's Customer Ids to help them pay for product quickly while ensuring that the credit card data cannot be stolen.

Thirdly, there is no two-factor authentication in the application. This is needed because in the event that a customer does have their password stolen, another form of authentication that serves as a second step for gaining access to a customer's account will further protect customers from attackers.

Finally, there was a huge focus on Android security even though the application is cross-platform. This was due to a lack of freely available tools to take advantage of when securing the IOS build of the application further than the in-app security.

4.0 Further Development or Research

With additional time and resources, I would have performed more research into IOS security to grant protection features exclusive to the iOS environment, such as the usage of iXGuard to protect against reverse engineering of iOS apps similar to the use of ProGuard for Android applications.

Additionally, I would create a RESTful web service for my application that would allow it to communicate directly with Stripe rather than make requests from inside the application. This is mostly because, while I'm concealing my Stripe API keys with code obfuscation, they can still be found by a highly qualified attacker, and the only way to combat this is to create a REST API that manages Stripe payments.

5.0 References

IBM, 2015. Secure mobile enterprise infographic. [image] Available at: <https://www.ibm.com/cloud-computing/images/secure_mobile_enterprise_infographic.jpg> [Accessed 28 April 2021].

Varghese, J., 2020. 10 E-commerce Security Threats That Are Getting Stronger By The Day!. [online] Get Astra. Available at: <https://www.getastra.com/blog/knowledge-base/ecommerce-security-threats/> [Accessed 28 April 2021].

Security, R., 2020. Basic Guide to Cybersecurity in E-commerce. [online] RSI Security. Available at: <https://blog.rsisecurity.com/basic-guide-to-cybersecurity-in-e-commerce/> [Accessed 28 April 2021].

Zinghini, F., 2018. Protect Your Ecommerce Customers with Mobile App Security. [online] Applied Visions Inc. Available at: <https://www.avi.com/blog/how-to-protect-your-ecommerce-customers-with-proper-mobile-app-security/> [Accessed 28 April 2021].

Wilczek, K., 2020. How to Secure Your Mobile Commerce App in 2021. [online] Droids On Roids. Available at: <https://www.thedroidsonroids.com/blog/mobile-commerce-security-tips> [Accessed 28 April 2021].

Guardsquare.com. 2020. Retail & E-Commerce Mobile Application Security | GuardSquare. [online] Available at: <https://www.guardsquare.com/blog/e-commerce-app-security-crisis-how-retailers-can-do-better> [Accessed 28 April 2021].

Signé, L. and Signé, K., 2018. Cybersecurity in Africa: Securing businesses with a local approach with global standards. [online] Brookings. Available at: <https://www.brookings.edu/blog/africa-in-focus/2018/06/04/cybersecurity-in-africa-securing-businesses-with-a-local-approach-with-global-standards/> [Accessed 28 April 2021].

Signé, L. and Signé, K., 2018. Global cybercrimes and weak cybersecurity threaten businesses in Africa. [online] Brookings. Available at: <https://www.brookings.edu/blog/africa-in-focus/2018/05/30/global-cybercrimes-and-weak-cybersecurity-threaten-businesses-in-africa/> [Accessed 28 April 2021].

Singh, S., 2019. Cyber Insecurity is Harming Emerging Markets. [online] Global Security Review. Available at: <https://globalsecurityreview.com/cyber-insecurity-harming-emerging-markets/> [Accessed 28 April 2021].

Blazevic, A., 2019. E-Commerce, Cyber Security and Governance in Africa. [online] Legaltechnologist.co.uk. Available at: <https://www.legaltechnologist.co.uk/e-commerce-cyber-security-and-governance-in-africa/> [Accessed 28 April 2021].

Odonkor, A., 2020. Unveiling the cost of cybercrime in Africa. [online] News.cgtn.com. Available at: <https://news.cgtn.com/news/2020-10-27/Unveiling-the-cost-of-cybercrime-in-Africa-UVhmu1PJeM/index.html> [Accessed 28 April 2021].

Masekesa, F., 2020. Nigeria, South Africa and Kenya dominate the e-commerce industry in Sub-Saharan Africa. [online] Theasianbanker.com. Available at: <https://www.theasianbanker.com/updates-and-articles/nigeria,-south-africa-and-kenya-dominate-the-e-commerce-industry-in-sub-saharan-africa> [Accessed 28 April 2021].

Głuszczyk, D., 2020. Mobile App Security Testing - Introduction & Tips for 2020. [online] Droids On Roids. Available at: <https://www.thedroidsonroids.com/blog/mobile-security-testing-introduction> [Accessed 28 April 2021].

Chhetri, C., 2020. Firebase Security Rules. [online] DEV Community. Available at: <https://dev.to/chandrapantachhetri/firebase-security-rules-43kn> [Accessed 2 May 2021].

General Data Protection Regulation (GDPR). 2021. Encryption | General Data Protection Regulation (GDPR). [online] Available at: <https://gdpr-info.eu/issues/encryption/> [Accessed 2 May 2021].

Stripe.com. 2021. Security at Stripe. [online] Available at: <https://stripe.com/docs/security/stripe> [Accessed 8 May 2021].

Kshetri, N., 2019. Cybercrime and Cybersecurity in Africa. Journal of Global Information Technology Management, [online] 22(2), pp.77-81. Available at: <https://www.tandfonline.com/doi/full/10.1080/1097198X.2019.1603527> [Accessed 7 May 2021].

Ololuo, F., 2020. Nigeria: Understanding Nigerian Data Protection Compliance Requirements And Managing Breach. [online] Mondaq.com. Available at: <https://www.mondaq.com/nigeria/data-protection/984628/understanding-nigerian-data-protection-compliance-requirements-and-managing-breach> [Accessed 7 May 2021].

Staff, A., 2016. Durban shuts e-services after data leak. [online] TechCentral. Available at: <https://techcentral.co.za/durban-shuts-e-services-after-data-leak/68304/> [Accessed 7 May 2021].

Specops Software. 2020. NIST Password Standards. [online] Available at: <https://specopssoft.com/blog/nist-password-standards/> [Accessed 2 May 2021].

Flutter. 2021. Obfuscating Dart code. Flutter.dev [online] Available at: <https://flutter.dev/docs/deployment/obfuscate> [Accessed 2 May 2021].

Van Niekerk, B., 2017. An Analysis of Cyber-Incidents in South Africa. The African Journal of Information and Communication, [online] (20), pp.113-132. Available at: <http://www.scielo.org.za/pdf/ajic/v20/06.pdf> [Accessed 6 May 2021].

Dart Team, 2021. Linter for Dart. [Online] Available at: https://dart-lang.github.io/linter/lints/ [Accessed 6 May 2021].

6.0 Appendices

6.1. Project Plan

Software Development Plan.mpp

Link to project plan

6.2. Reflective Journals

6.2.1. September & October

My Achievements

Over the past few months, I was able to purchase the equipment needed for my project to function according to my specification, the tools that were purchased were NFC tags that serve as the middlemen for the viewing of products from a mobile device.

Since my project is going to be developed using the google framework Flutter, I have been taking a course on it to aid in the development of my project, as of writing I have currently completed 40% of the course.

My contributions to the projects included determining a design to be followed for the user interface. The designs that have been chosen are for the landing & product pages, login, cart and waiting page. The designs can be view here: https://dribbble.com/theyoungceo/likes.

Other contributions include researching into encrypting data on an NFC tag to preserve the integrity of the product, prevent an attacker from editing the data within a tag or to read the contents of the tag without the use of the application. A tool that I discovered that allows me to do all of this and more is a mobile application named NFC Tools Pro. This application provides a user with the ability to lock an NFC tag to make sure that it can no longer be written to, updated or deleted by a user with malicious intent.

A meaningful contribution is discovering a content management system that does not require me to develop a backend for the application. The CMS that I will be using for the development of this project is Strapi; I chose this because it is open-sourced, uses very little memory overhead and has consumable APIs that are straight forward since the plan is packed with UI components it will enable me to have multiple product pages that can serve content to the user quickly.

My Reflection

I originally planned to have a login system developed for this month. Still, I was struggling quite a bit getting Firebase up and running so I have decided to delay its development and schedule it to be completed before the end of November.

I planned to get a start on the creation of my requirement specification document since it is due at the end of November. Still, after the discussion of my project idea to my supervisor, it seems that I might need to come up with a new project idea that will allow me to receive the best grade. I'm slightly still not sold on the idea of my project not being feasible since it is less demanding to develop complexity wise so there is still a chance for me to continue with the current idea. I mainly do not want to drop it due to it being something I personally want to make and that I have poured far too much time and effort into researching the development of the project.

Intended Changes

Next month, I will try to rethink my original project idea my project idea to hopefully choose something that is not only feasible but also will get me the best grade possible and will push me into wanting to create it.

Supervisor Meetings:

Meeting 1:

Date of Meeting: 21/10/2019

Items discussed:

Project idea and details that I know in relation to the project.

Action Items:

Write up a report on the latest trend within the e-commerce space and list products currently out there that resembling your project idea.

Meeting 2:

Date of Meeting: 24/10/2019

Items discussed:

Research report for project and complications with the project

Action Items:

Read and performance research on the project ideas given by the supervisor to determine an actionable project idea.

Dr Keith Maycock – Lecturer

6.2.2. November

My Achievements:

This month, I was able to begin development on the project and set up a GitHub repository.

Due to the requirement specification being due this week, I was mainly working on my use case diagrams and class diagrams to describe the architecture behind my project further. Upon reflection doing these diagrams has made me think deeply about the functionality behead my project and the features that potentially unnecessary or need to be considered.

My contributions to the projects included developing the landing, product, cart pages of my application.

Intended Changes

Next month, I will try to develop a working prototype that will get my project idea across in the best way possible, and I will have to achieve this promptly due to their being multiple other submissions due. With this in mind, I will not be adding in the back end of my application for my mid-point presentation but instead, I will develop merely the user interfaces that will flow to the correlating pages and leave it at this for the time being.

Supervisor Meetings

Meeting 1:

Date of Meeting: 13/11/2019

Items discussed:

I clarified further what the premise behind my project was and explained my reasoning for why I wanted to continue working on the idea. I explained that instead of designing the application for merely one store, I would develop it as a single source for multiple stores. Keith Maycock then provided a suggestion due to my project lacking complexity. He suggests that I add bidding zones where when a customer is near to a store using the service the business then proceeds to place a bid on how much they'd like to pay to get the customer into the store.

I pitched my reasoning for wanting to use NFC tags in my project, and he had many concerns regarding customers being afraid that they might be tracked.

Action Items:

Perform research on how to implement KNN into the project.

Rethink the usage of NFC tags with the project.

Meeting 2:

Date of Meeting: 25/11/2019

Items discussed:

Due to Keith being unavailable this week, I attend a meeting with the standing in supervisor Gustavo where he and I discussed the architecture of my application and brought up any problems with the project that Keith discussed. He helped me out big time by diagramming the project architecture for and describing how KNN will fit into the project.

Action Items:

Read resources sent on KNN.

Dr Keith Maycock - Lecturer

6.2.3. December

Project Presentation

Date of Meeting: 17/12/2019

What was discussed here was just what my project is about repeated in a much more consumable format presented to Keith and another lecturer. In my slides, I mainly discussed whether or not there was a market for my project and literature review of ideas similar to my project. From here, I started to discuss the security features of my application and the frameworks that I was using, following this, I demonstrated my application's prototype and the features that I developed.

Once my presentation was completed my supervisor seemed quite pleased at the amount of research that I conducted to back up my idea and didn't have much to say in terms of feedback but the second examiner had only one mistake to point out and that was my

explanation of the use of weka which is a data analysis tool. I believe that this was the only thing I got incorrect in regards to the presentation as a whole. Aside from that, I was then given feedback on the format of my slides and what I could have done to make the better.

Action Items:

Research more in-depth into the use of weka specifically for my project.

Intended Changes

Next month, serious development time must be undertaken to ensure the completion of my software project, I have currently decided to the change the framework that I intend on using for the development of the store CMS/CRM of my project to a more trustworthy framework in angular due to flutter hummingbird being only in beta and is most likely not in a secure enough development stage to be used in production.

Dr Keith Maycock – Lecturer

6.2.4. January

My Achievements

Throughout the year I was able the complete the main functionality of my final year project to a presentable and fully functional state. Currently, my application can register users with either Google, Facebook or basic email and password fields. This functionality is accomplished with the use of Firebase which handles the sign-in and registration SDKs for both IOS and Android. When choosing a password the application requires the user to use the latest password security practises and along with this the password is checked via an API (HaveIBeenPwned) to discover whether the chosen password has been compromised or not.

Upon logging in or registering an account, if the user has not verified their email address via the link sent to their email address they will be required too in order to login to their account and begin shopping. Once verified the user is presented with a map with stores close to them. These stores are represented in the form of a pin on a map and upon being taped the user will be directed to a store page where a list of product is shown. Each product page has basic e-commerce functionality such as increasing product quantity and adding products to the cart for an individual store.

Once a product has been added to the cart it can be seen within the tab for specific stores. When a user is ready to checkout they are provided with a total and they can enter their credit card information to complete the order. If the credit card is valid a transaction is made and an order is generated with a QR code used to confirm and collect the order.

My Reflection

When writing my code didn't take care in writing it as clean as possible which resulted in redundant being written or code being duplicated as opposed to writing a function that could be reused throughout the codebase. In future, I will have to rewrite multiple lines of code to conform to both the dry principle but also secure programming.

Very few security measured are put in place throughout the application making it very open to reverse engineering and external requests to the firebase database.

My choice of user interface design on the cart and orders page may be a problem if the project is to have multiple stores because it will force the user to look through multiple stores to view their cart. This problem can be fixed by simply implementing a search feature for both the cart and order pages.

To follow more conventional e-commerce features, it would have been desirable if I had created an invoice that was sent to the user's email once an order was paid for.

Intended Changes

My next steps for the project or to implement the use of NFC within the project to allow users to scan tags and have them appear on the device. Unit testing is another task that is vital to the completion of my project. I will mainly be testing database and authentication calls to firebase.

Supervisor Meetings

Meeting 1:

Date of Meeting: 28/01/2021

Items discussed:

Throughout the meeting with Keith I presented what I have completed up to now with the project. I demonstrated the main functionality of the mobile application such as login & registration, cart functionality and making orders. Extra items that were discussed included focusing in on the security aspects of the mobile app such as protecting again reverse-engineering attacks, write firebase security rules and unit testing the application. Keith's other recommendations included design updates on the login & signup pages to make them more attractive to a marker.

Action Items:

Continue with the development of the application and double down on the security aspects of the project rather than writing software code. Return in two weeks with a security feature implement and explain the reasoning behind its application.

6.2.5. February

My Achievements

I accomplished the majority of my goals for this month with regards to the project, I implemented the use of NFC technologies in the project which allows a user to scan a tag that opens up the application and display the product stored in the NFC tag for the user. These NFC tags have been secured with the use of locking which prevents an attacker from editing an NFC tag.

Continuing I secured the project's API keys by placing them in a .ENV file that can't be accessed unless pushed to GitHub With this reverse engineering protection has been implemented for Android with the use of ProGuard to obfuscate the code used in the application.

My Reflection

The password used for the NFC lock must be changed to something more secure for the password is very common and easily be brute-forced by an attacker.

Reverse engineering protection must be implemented on the IOS version of the application.

API keys were pushed to GitHub and must be removed from the git commits.

Intended Changes

My next steps for the project is to focus on writing unit and integration tests to show that the project is fully functional and works as intended. Aside from project work intend on mainly putting my attention into the writing of my project report to further explain the security features and functionality of my project.

Supervisor Meetings

Meeting 1:

Date of Meeting: 25/02/2021

Items discussed:

Throughout the meeting with Keith I presented what I have completed up to now with the project. Demonstrated the new products added to each store, this was shown because previously I had stored one to two products for each store for mainly testing purposes but now the project is capable of store hundreds of products for each store.

The next item that was discussed was the use of firebase security rules to prevent bad actors from being able to read sensitive data from the database. I demonstrated how security rules functioned with the use of Postman to make requests to the users table. The first request that was made was with security rules turned off, doing this allowed me to request my users table to retrieve information about all my users. The second request that was made had security rules turned on and when the request was made I was sent a 403 forbidden, demonstrating that with the use of security rules the project can reject requests to the database from unauthorized users.

Continuing on I displayed the use of end to end encryption in the project. This secures both the user information that leaves the device but also data the device requests. The algorithm that is being used for end to end encryption is AES.

Action Items:

Cease development of the project for the time being and focus on writing about the security aspects of the project in the project report. Keith Maycock provided feedback on the use of AES encryption by recommending the use of salt to append to an encrypted string for extra security.

Dr Keith Maycock - Lecturer

6.2.6. March

My Achievements

This month has been very busy in regards to course work and I, unfortunately, did not complete a whole lot of work related to the final-year project, however, I managed to complete a couple of sections with my technical report such as the Aims and Technology sections in the technical report

My Reflection

Upon testing features in my application I realized that I was missing other basic e-commerce and mobile application functionality such as a forgotten password system that would allow a user to recover their accounts if they've managed to forget their password or user credentials. Other functionality includes generating invoices for purchased goods in the application because the customer usually prefers some form of confirmation to show that they have made an exact order for these exact items. Finally, missing functionality includes extra code obfuscation for the IOS build of the application to make it difficult for attackers to reverse engineering and learn how the application's functionality works.

The thing I probably would have done to make my application much more secure is to create a web API that would make a request to the stripe payments endpoint for extra security because the software performing the payment would not be on the application.

Intended Changes

My intended changes for this month are to complete my technical report and receive feedback from Keith Maycock next month in preparation for submission and the project presentation.

Supervisor Meetings

Meeting 1:

Date of Meeting: 01/04/2021

Items discussed:

Throughout the meeting Keith and I went through various sections of the technical report to discuss what he expects from each section. Keith's first recommendation was to write about what challenges are present in securing an e-commerce application, this includes what money is lost, how much time is wasted either recovering from the attack or fixing vulnerabilities and how big the impact is. Keith continues this by discussing the digital unaversive infographic which goes into detail on how older methods of attacking are making a come-back in emerging markets such as India and China. Overall I found this meeting very insightful and has details that I intend on writing down in my technical report this month.

Action Items:

Complete technical report and have it reviewed and critiqued by Keith before the submission date. Doing this will allow Keith to provide me with feedback on how well I did and what information I need going into a bit more detail on the gain extra marks.

Dr Keith Maycock - Lecturer

6.2.7. April

My Achievements

This month's work was mostly focused on the technical report in order to complete it by the May submission deadline. The document is 90% complete, with the exception of the system design architecture and evaluation sections. I conducted analysis for the technical report about how cyber-attacks impact emerging the e-commerce markets; I addressed the money lost and time spent attempting to rebound from said attack. The functional requirements took a long time to write but are now complete; the functional requirements even include the server-side firebase security rules. Although this is not code that can be submitted, it can be discussed in detail in the implementation section of the document to illustrate the presence of security in my application and on the server-side.

My Reflection

References will be used to amend items in the current version of the technical study. GDPR recommendations in the section on data requirements ti emphasize the importance of privacy as a component of cyber-security is important. Although the report is very lengthy, it will need additional detail in my introductory sections, where I will address the consequences of not applying security in application from emerging market.

Intended Changes

My target for this month is to complete my technical report and obtain input from Keith Maycock prior to the submission deadline in one week. Additionally, the project must be in a condition that allows it to be submitted with all of its functionality and security features intact. From now on, implementation of new features in the application will be limited to bug fixing and error handling.

Supervisor Meetings

Meeting 1:

Date of Meeting: 06/05/2021

Items discussed:

What I've accomplished in terms of the technical report. Keith reviewed the report and highlighted sections that needed more clarity, such as my executive summary and functional requirement. He suggests that I rewrite my report to concentrate exclusively on how to apply cyber security in modern applications, rather than discussing how my project relates to e-commerce.

Additionally, there were problems with my use case diagrams, as some of them did not adhere to UML standard notation and needed to be edited, and the actors in each diagram needed to be modified from an image to the standard actor image.

Action Items:

Complete the technical report and have it read over by Keith once more before submission.

Dr Keith Maycock - Lecturer