Design and Development of a Platform Agnostic Supply Chain Tracking Application

Danijel Dordevic
Student ID: 17-717-778
Zurich, Switzerland

Supervisor: Sina Rafati Niya, Prof. Dr. Burkhard Stiller
Date of Submission: August 27, 2019

University of Zurich
Department of Informatics (IFI)
Binzmuhlestrasse 14, CH-8050 Zurich, Switzerland

INDEPENDENT STUDY MODULE – Communication Systems Group, Prof. Dr. Burkhard Stiller

Independent Study Module
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuhlestrasse 14, CH-8050 Zurich Switzerland
URL: http://www.csg.uzh.ch/

Abstract

While traditional supply chain (SC) systems met their requirements, they eventually reached their limits, especially in a sense they do not provide enough transparency for the final consumers. They are also highly specialized in supplying just specific goods, meaning they are not generic enough to be reused for other types of products. This is especially true for food, which safety and transparency are paramount to the final consumers. Traditional SC systems require a lot of manual work which leaves room for errors and manipulating with goods. Also, traditional systems lack integration between different systems in the supply chain, which requires more work to make those systems work together. The commence of the Internet changed the way how traditional SC systems functioned. The more agile and dynamic way of working was introduced. That allowed the consumers to interact with their products at any of its stages. While the traditional SC systems have been improved over time, there is room to advance the existing SC systems to the next level, especially when speaking about the transparency and making them more user-friendly for the final consumers. In this independent study, the requirements mentioned above are addressed by developing the application that can be used by both consumers and producers.

Acknowledgments

I want to take the opportunity and thank my supervisors Sina Rafati, Prof. Dr. Burkhard Stiller, and other members of the Communication Systems Group at the University of Zurich for allowing me to pursue this Independent Study Module.

Contents

Abstract **i**

Acknowledgments **ii**

Contents **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 2

2 Design and Implementation **4**

2.1 System Design 4

2.2 Implementation 7

2.2.1 React Native 8

2.2.2 State Management 9

2.2.3 Multi-Language Support 10

2.2.4 Development Process 10

2.3 Application Modes 11

2.3.1 Producer Mode 11

2.3.2 Consumer Mode 18

3 Evaluation and System Analysis **21**

4 Summary and Future Considerations **24**

Bibliography **25**

Abbreviations **27**

List of Figures **27**

List of Figures **28**

A Installation Guidelines **30**

B Contents of the CD **32**

Chapter 1

Introduction

1.1 Motivation

A supply chain (SC) represents a network of parties responsible for creating and selling the products, from the source (raw) materials through its processing in multiple steps and delivery to the final consumers. This process is known as the distribution channel.

Each of these middlemen represents both a producer and consumer in that chain. It is crucial that each of them can verify the full history of the product. If any of the stages in this distribution channel gets compromised, the whole SC is in jeopardy. This is especially important when dealing with food, which safety is paramount for the final consumers, and any issue with it could result in fatal consequences for all the parties within the channel.

[1] points out the five major problems in the current SC systems. Each of them is briefly explained with the proposed solutions and how the SC tracking application developed as part of this IS tries to tackle some of these problems:

1. **Lack of traceability**

   Even in the Internet era, many products are untraceable, and this is the main concern for many consumers nowadays because they want to know where each ingredient of their product came from. Traceability would straighten the brand integrity and increase customer loyalty.

   Our application tackles this problem by introducing labels being stuck on the product in each step of the channel. The labels hold just a hash, stored on the blockchain, that can not be forged, and which contains the reference to the data about the product. This enables the full traceability and transparency not only for the final consumers but also for each party of the SC system.

2. **Inability to maintain the safety and quality of the products**

   Today's SC systems are more complex than ever before. The manufactures are under the pressure to produce and deliver high-quality products in ever limited time. There are several main challenges like climate changes, delays in transportation, etc. The proposed solution for these few issues is that everything starts from the high-quality raw materials, also the packaging and storing should be taken into account.

3. **Inadequate communication between parties**

   In ever-changing and more complex SC, communication becomes more difficult. There is little to no knowledge of one another's actions. Poor communication causes issues with delays, increase costs, less quality final products.

   Our application partly solves these problems by providing valid information about each product. The information contains the previous actor in the channel, actions being performed on the product, date, etc.

4. **Rising supply chain costs**

   To run an SC system is ever more expensive mainly due to the high energy costs, manpower, investing in new technologies.

5. **Failure to track and control inventory in warehouses and stores**

   There has to be a trade-off between too strict and too lax controlling of inventory. To strict control would incur higher costs without bringing benefits while too lax control would result in low-quality products.

   The SC tracking application again helps to tackle this issue by enabling easy verification of each product in each of its stages.

There are several other mechanisms that would enable tracking the product.

RFID [2] is one of them but it would require the usage of expensive chips that would have to be implemented in each product, also it would require specialized scanners for validating the products.

GIS [3] uses GPS for tracking the products, but this would introduce even more expensive equipment. There are use-cases where GIS makes sense, like tracking the products in real-time.

In our use-case, combining the QR labels that contain the hash data, and immutable nature of the blockchain technology is the best candidate for implementing cost-effective, robust and reliable SC tracking system that is easy to use by all parties.

1.2 Description of Work

The aim of this IS is to develop a mobile application that is an integral part of the SC Tracking System. It is meant to be used by both producers and final consumers. It describes the main architectural and implementation decisions. The focus is on seamless integration of the mobile application into the system, allowing all the parties of the SC easy, reliable, and cost-effective usage of the system.

Chapter 2 explains the technical and implementation details about the application. The reasons for choosing certain technologies were given.

The section 2.3 explains the flow of the application and kind of a user-guide for both modes, consumer and producer.

Chapter 3 provides the evaluation details of the application while finally chapter 4 sums up the overall project and provides some future considerations of the application.

Chapter 2

Design and Implementation

In this section, the architectural decisions and implementation will be explained in details. The first part describes the overall architecture, system design, high-level overview. Later, the back-end system will be explained that this application relies on. Lastly, the application itself will be explained in details starting from the technologies/frameworks being used.

2.1 System Design

System design is the process of defining the elements of a system with all its components, modules and interfaces. It can take a bottom-up or top-down approach [5]. A system diagram is a high-level diagram that depicts the boundaries between the system and its environment. The figure 2.1 shows the high-level overview of the components and actors within the system. It can be seen that mobile application plays a central role. It is a mediator between the producers and consumers, and the back-end system. It shows that the producers can generate QR codes, store the data on the back-end system and that the consumers can later validate products by scanning the products/QR codes. This means that this application can be used both by consumers and producers which will be explained in details in the following section.

Figure 2.2 shows the use-case diagram of the mobile application. The purpose of the use-case diagram is to show the interactions between the elements of the system.

The given use case diagram can be breakdown into four main components:

1. **System:** The application developed. In this case, it represents the food chain tracking mobile application. It is a rectangle encompassing the use-cases and relationships. All components within this rectangle are part of the system.

2. **Actors:** Represent the external entities that use the functionalities provided by the system. In our case, the actors are the producers and consumers.

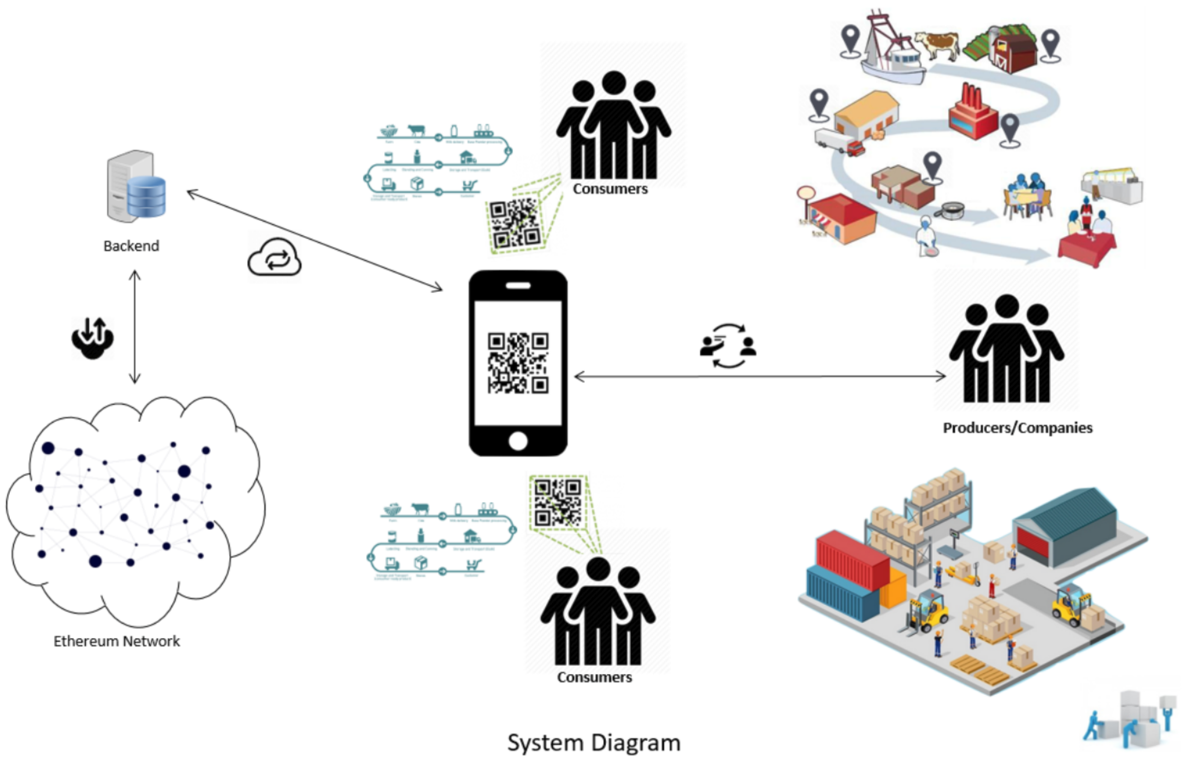


Figure 2.1: System Design Diagram [6]

3. **Use-Cases:** The oval-shaped elements representing the functionalities provided by the system, and used by the actors.

4. **Relationships:** There are several types of relations.

   - **Association:** Represented as a solid arrow. In this case, there are associations between the system and users, and consumers. A producer can generate QR code, sign in/up and update its data. On the other side, consumers can only scan products/QR codes and show the product's data.
   - **Extend:** Represented as a dotted arrow, describes the use case that is extended by some additional functionalities. In this case, generating a QR code can be extended by the functionality that prints that newly generated QR code.

Container Diagram

Figure 2.3 shows the container diagram. Essentially, every application represents one container that is runnable/deployable unit that process, stores or serves data. In that way, the benefits of using the micro-service architecture are leveraged. It represents the high-level view of the system architecture and how the responsibilities are distributed across containers. It also shows the major technology choices and how the containers communicate with each other.

The next part shortly explains each of the components.

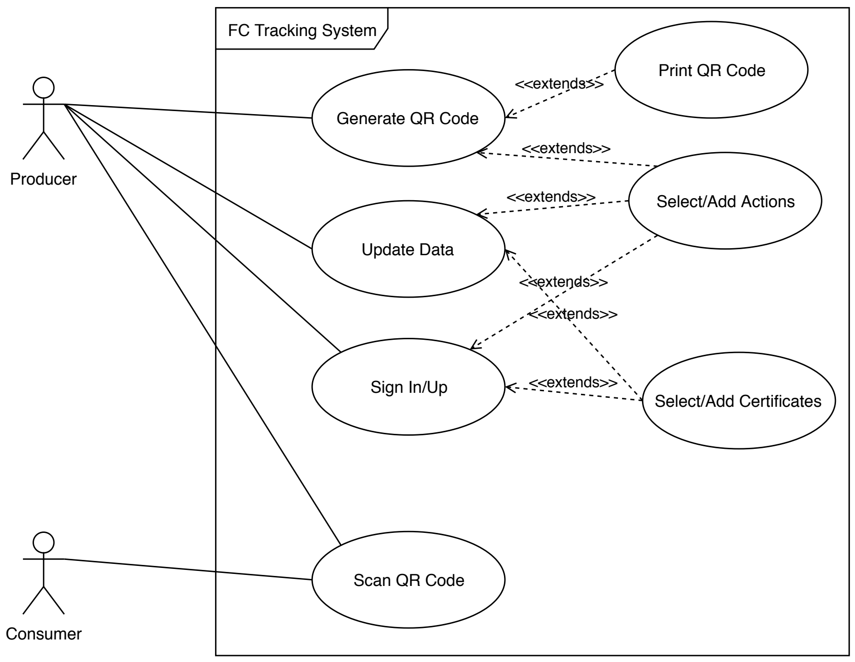


Figure 2.2: Use Case Diagram

Database Food Chain Tracking System stores all its data in the SQL database. All kind of SQL-like databases supported by Hibernate can be used to store the data.

RESTFul Web Service Backend application exposing RESTFul APIs that other application can call.

Ethereum Node In order to call the smart contract and validate product tag's hash, the backend application calls the Ethereum node which can call the smart contract and validate the given hash value. The backend can easily switch to use a remote Ethereum node if e.g. the machine that the machine runs on doesn't have enough resources to run a node.

Web Application The application used by consumers. It is built as a single-page web application and its production build is served by Nginx web server.

Reverse Proxy Used in order to hide the internal structure of the system and increase the overall security. It exposes the server's and backend's APIs under the same origin.

Android Appication Used by producers that can create and validate product tags. It can also be used by consumers for checking the validity and details of product tags.

React Native Application A mobile application builds with React Native framework that provides the same look and feel on both Android and iOS platforms. It can also be used by both producers and consumers and it is intended to provide the same functionalities as the aforementioned Android application.

Ethereum Network A distributed-computing platform and operating system featuring smart contract functionality. It runs the smart contract written for this platform and executes the transactions against the contract.

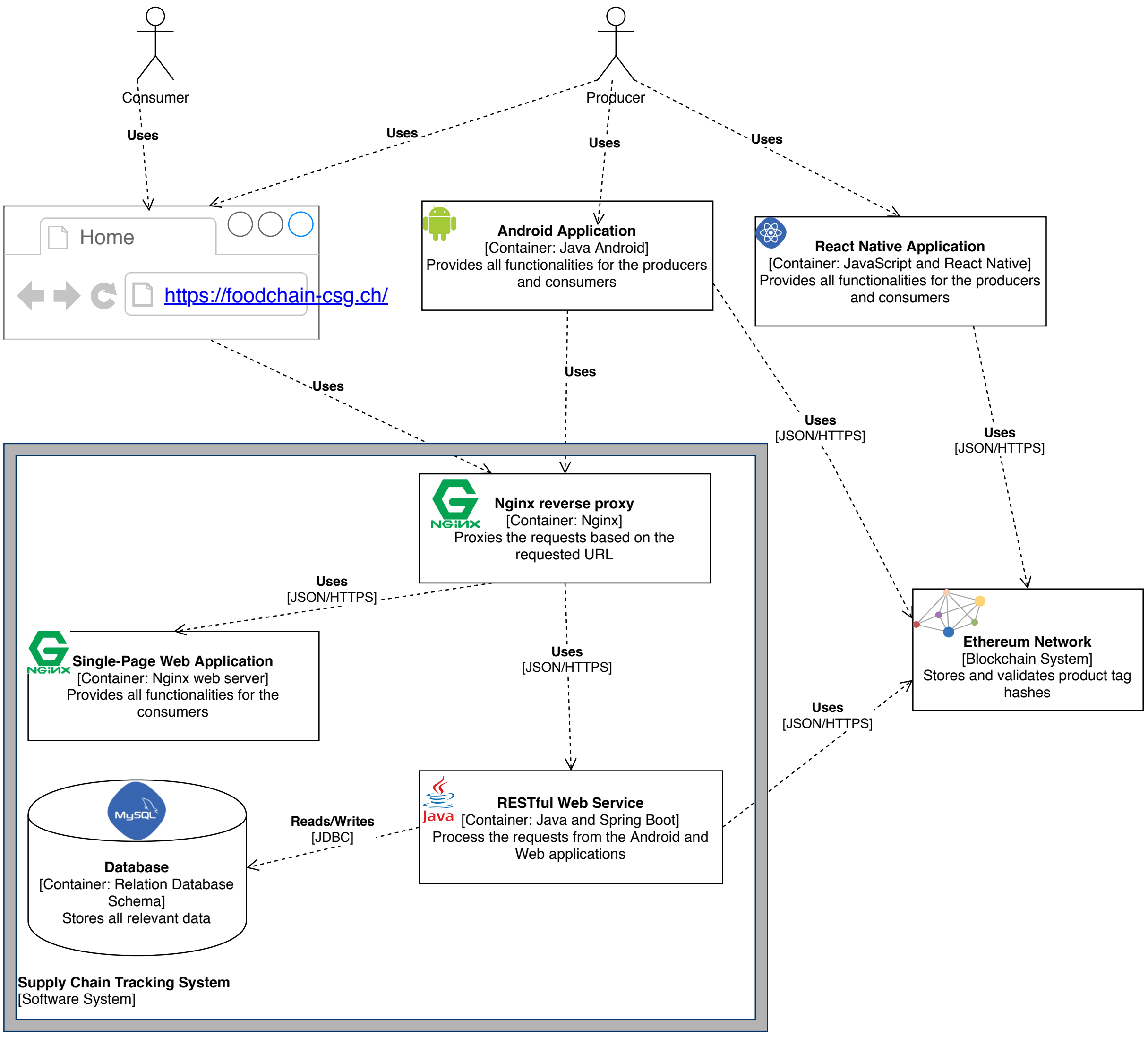

Consumer
Producer
Uses
Uses
Uses
Uses
Home
https://foodchain-csg.ch/
Android Application
[Container: Java Android]
Provides all functionalities for the producers and consumers
React Native Application
[Container: JavaScript and React Native]
Provides all functionalities for the producers and consumers
Uses
Uses
Uses [JSON/HTTPS]
Uses [JSON/HTTPS]
Nginx reverse proxy
[Container: Nginx]
Proxies the requests based on the requested URL
Uses [JSON/HTTPS]
Single-Page Web Application
[Container: Nginx web server]
Provides all functionalities for the consumers
Uses [JSON/HTTPS]
Ethereum Network
[Blockchain System]
Stores and validates product tag hashes
Uses [JSON/HTTPS]
Database
[Container: Relation Database Schema]
Stores all relevant data
Reads/Writes [JDBC]
RESTful Web Service
[Container: Java and Spring Boot]
Process the requests from the Android and Web applications
Supply Chain Tracking System
[Software System]

Figure 2.3: Container Diagram for Food Chain Tracking System.

This overview of the whole ecosystem is given in order to show the role of the Application being built. In the next section, the implementation details will be explained.

2.2 Implementation

This section covers the implementation details for the Food Chain Mobile Application. The main goal of this mobile application is to bring the supply chain platform to the producers and consumers using the iOS platform. Therefore, the requirement was to build a mobile application that can run on the iOS platform. There are several possible technologies/frameworks available for building the iOS applications such as React Native[7] Flutter[8] and Ionic[9]. In the end, the React Native framework is utilized. This section will explain the React Native internals, reasons why the React Native was used, and also the implementation details will be given.

2.2.1 React Native

React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It is based on React [10], Facebook's JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. React Native currently supports both iOS and Android and has the potential to expand to future platforms as well [7].

The fact that by writing JavaScript code and running it natively on both Android and iOS platforms brings clear benefits. Only one code base should be maintained while utilizing well established frameworks/tools such as React [10], Redux[11], Material-UI [16] etc.

As mentioned, React Native provides almost native performances. The reason is that the code written in Java Script gets compiled natively to the target platforms where currently Android and iOS are supported. Also, the support for a few other platforms is under development [18]. Figure 2.4 illustrates the flow of the compiling process.

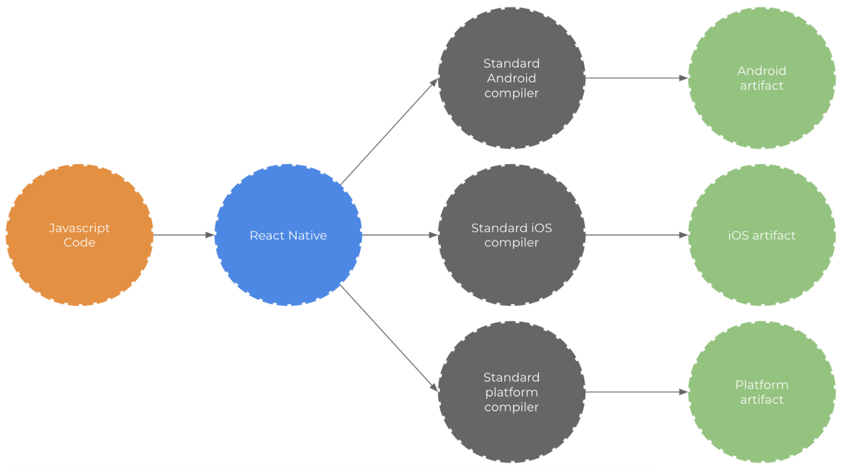


Figure 2.4: React Native compiling flow [17]

The heart of the React Native architecture is the bridge that allows the communication between JavaScript and a native/target platform. The bridge provides bidirectional and asynchronous communication between these two different technologies. Figure 2.5 illustrates the communication between two different technologies using the bridge.

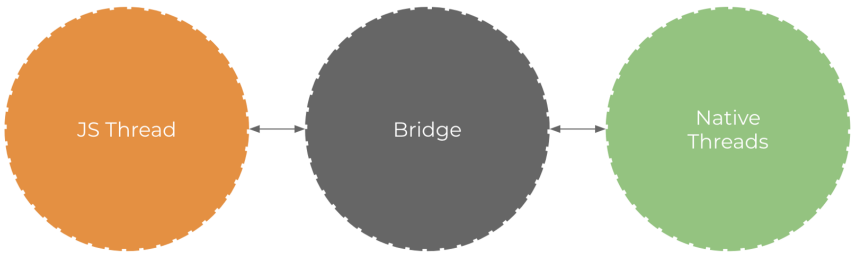


Figure 2.5: Bridge sitting between JavaScript and Native Threads [17]

2.2.2 State Management

Managing application state is a crucial component in any software development process. React itself provides some useful methods for setting component's state using `setState()` and adding a 'local state' to a class. The method `setState()` will set the state in the corresponding class, which means that the changed state will be accessible just inside that class and its children classes and components. Managing states just with React is possible but very ineffective due to growing complexity and performance issues, especially as the application grows.

Redux [11] is a library that helps to manage application state in a consistent and predictable manner. The Redux Store is basically the single source of truth for the application and it can be changed only with pure functions [12]. The React component can connect to the Redux store by using the `react-redux` library[13]. That allows the components to subscribe to the Store changes, and also trigger the update on the Store.

The figure 2.6 shows the Redux flow in React applications. It is important to mention that when making the asynchronous calls, the additional middleware has to be used that will update the Redux Store on the call completion. In this application, for that purpose Redux Thunk [15] was utilized.

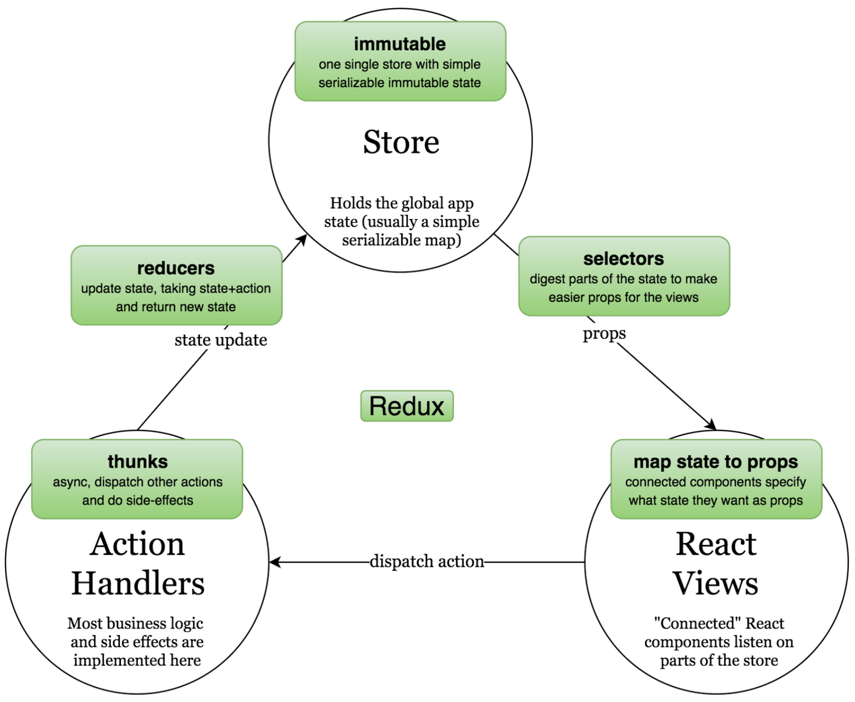

immutable
one single store with simple serializable immutable state
Store
Holds the global app state (usually a simple serializable map)
reducers
update state, taking state+action and return new state
state update
selectors
digest parts of the state to make easier props for the views
props
Redux
thunks
async, dispatch other actions and do side-effects
Action Handlers
Most business logic and side effects are implemented here
dispatch action
map state to props
connected components specify what state they want as props
React Views
"Connected" React components listen on parts of the store

Figure 2.6: Redux Flow[14]

Section 2.2.3 explains how the Redux library was utilized to implement multi-language support.

2.2.3 Multi-Language Support

As the global application state is handled with Redux [11], the multi-language support is implemented with the same library. The language can be changed from the home screen drawer. Currently, German, English, French, and Italian are implemented. Figure 2.7 shows the open drawer where the buttons for the implemented languages are shown. The button for the current language is in blue color. By changing the language, the Redux Store gets updated which triggers the text to change in the whole application. This allows the managing of all the translations of the application from one central place. All components that have to display some of the translations, just have to subscribe to the global Store using the aforementioned `react-redux` library and select the key of the corresponding text.

Figure 2.8 shows the home screen in German language while 2.10 shows the same screen in English translation.

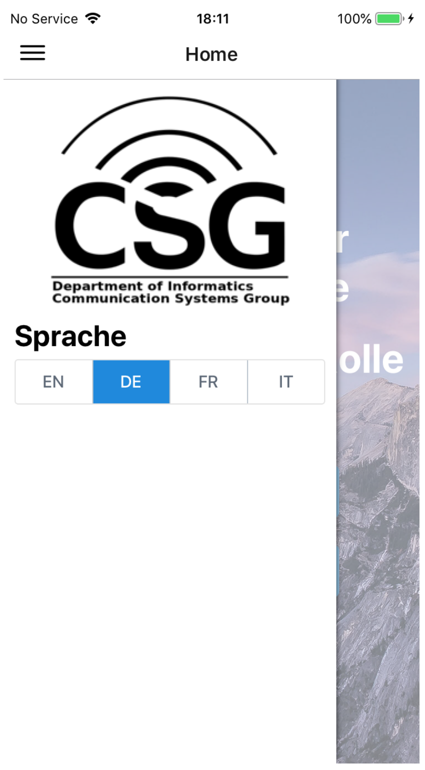


Figure 2.7: Home screen drawer

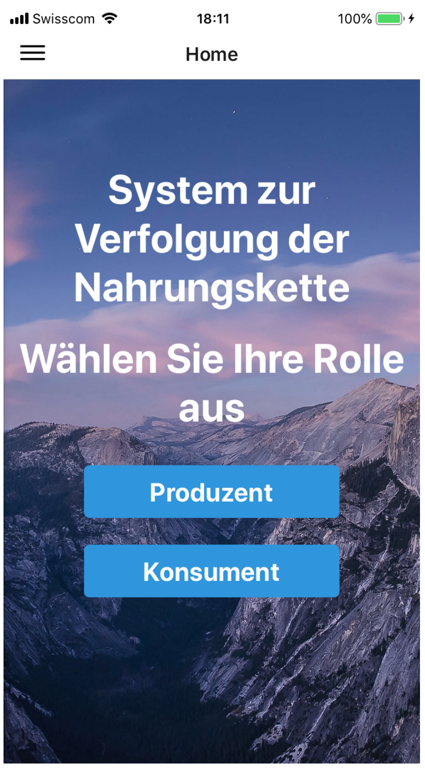

Swisscom
18:11
100%
Home
System zur Verfolgung der Nahrungskette
Wählen Sie Ihre Rolle aus
Produzent
Konsument

Figure 2.8: Home screen - German translation

2.2.4 Development Process

There are two main ways of building a React Native application. Using the React Native CLI [19] or Expo CLI [20].

React Native is a default tool. It is a more flexible way of developing because it allows developers to link native modules written in other languages (Java, Kotlin, Swift). It even allows integrating React Native into an existing native application. The drawback is that it requires XCode for iOS and Android Studio for Android application. Also, the initial setup is more complicated.

Expo tries to solve these problems. It provides a set of tools that simplify the development and testing of React Native applications. It integrates the components of users interface and services that are usually available in third-party native React Native components. This significantly simplifies the development process and allows the developers to focus on building the functionalities of their applications, instead of spending time on setting up the development environment. Expo also manages the application update when a new version of React Native is released. Also, building an iOS application without a MAC machine brings clear benefits. By taking all that into account, Expo was a clear choice.

The typical way to use Expo tool is the managed workflow, figure 2.9.

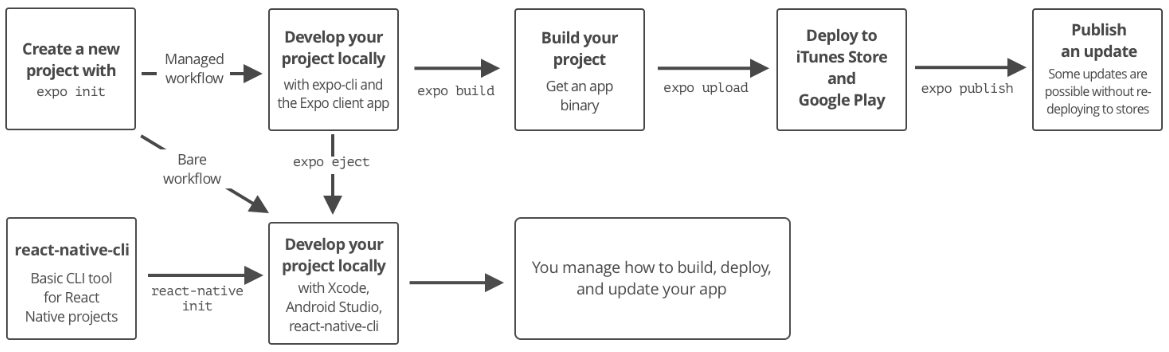

Create a new project with
expo init
Managed workflow
Develop your project locally
with expo-cli and the Expo client app
expo build
Build your project
Get an app binary
expo upload
Deploy to iTunes Store and Google Play
expo publish
Publish an update
Some updates are possible without re-deploying to stores
Bare workflow
expo eject
react-native-cli
Basic CLI tool for React Native projects
react-native init
Develop your project locally
with Xcode, Android Studio, react-native-cli
You manage how to build, deploy, and update your app

Figure 2.9: Expo Managed Workflows [21]

Expo allows a very convenient way for testing and debugging applications while using the managed workflow. It provides a development server and applications for both iOS and Android platforms. When the development server is up, it will generate a QR code. All we have to do is to scan the QR code from the Expo application, and the application we are working on will run on the phone. Hot reload is also available which speeds up the development process.

2.3 Application Modes

As mentioned, the Food Chain Mobile Application provides the functionalities for both producers and consumers.

2.3.1 Producer Mode

The producer mode of the application provides a wide set of functionalities that help the producers to successfully use the Food Chan Tracking system. As shown in 2.2, producers can log in, register, add or create actions and certificates, generate QR code, see the QR codes they have generated, update their data. Producer's authentication screen can be entered by pressing the Producer button from the home screen.

By entering the producer's authentication screen, two tabs will appear:

- **Login**

  The producer is already registered and it has to enter its username and password to access the application.

- **Sign Up**

  The producer has not registered yet so it has to create a new account. The following fields are required:

  - **Producer Name**
    The registered name of the producer.
  - **Licence Number**
    A number that uniquely identifies each business entities within a country.
  - **Username**
    Producer's username that it can be used for logging into the system. It has to be unique and with a minimum of 5 characters.
  - **Password**
    Required for logging into the system. It has to have a minimum 8 characters.
  - **Ethereum Account**
    64 character hex string that represents the producer's ethereum account number.
  - **Website URL**
    Not required field represents producers URL address if any.
  - **List of Certificates**
    The producer can select the certificates from the provided list, it can also add additional certificates if required as shown in figure 2.13.
  - **List of Default Actions**
    The producer can select the default actions while creating an account so it does not to select them for each product tag generation. New actions can also be created on the fly as shown in figure 2.14.

  Once the producer fills in all the required fields, it has to press the button **SignUp** from the figure 2.14. All the fields will be validated on the client-side. If there is any error, the corresponding error message will be shown. Otherwise, the application will make an HTTP request against the server that will save the producer data in the database. Upon successful account creation, the producer will be redirected on the producer's welcome page, as shown in figure 2.15.

  On successful sign-in/up action, the server will return a JWT token used for authentication. The mobile application will store the token in the Redux store. The application will include the JWT token in the header for each request that requires the authentication. The token will be destroyed on the sign-out action.

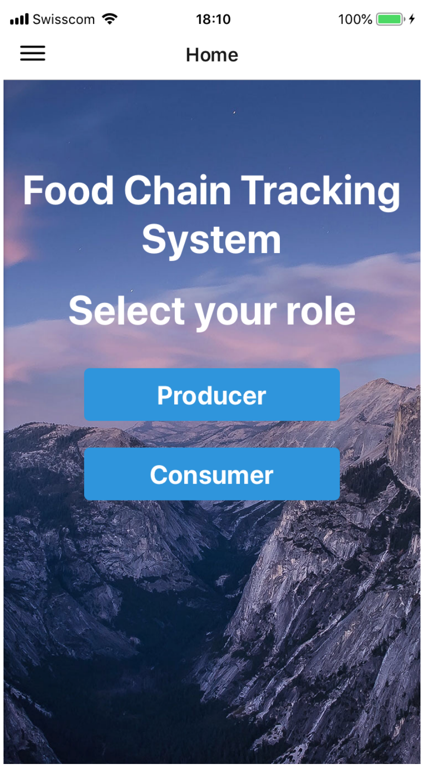


Figure 2.10: Home screen

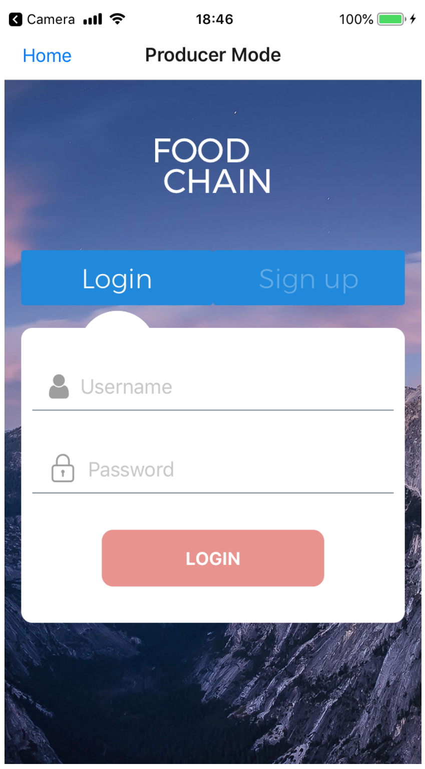


Figure 2.11: Producer Login Screen

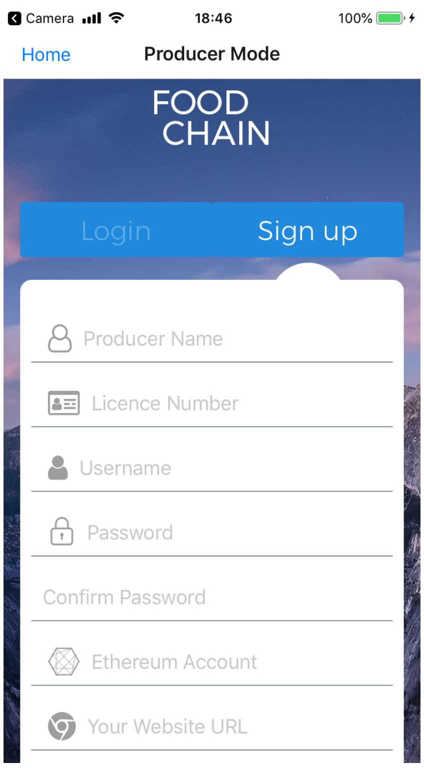


Figure 2.12: Producer Sign-up Screen 1

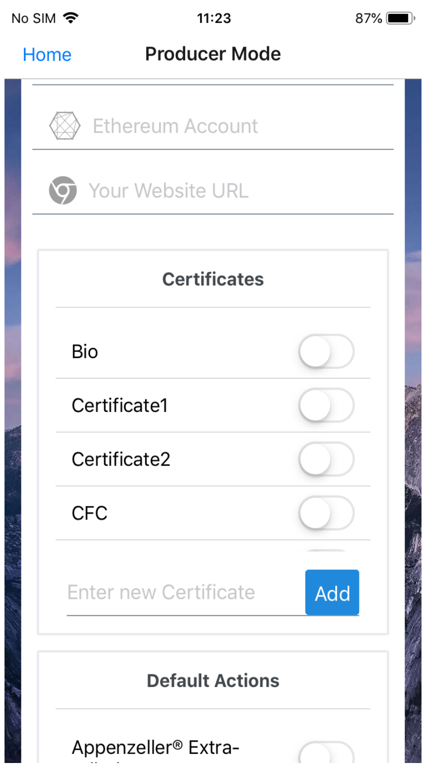


Figure 2.13: Producer Sign-up Screen 2

Product Tag Generation

The main functionality of this mobile application is the product tag generation. Upon successful logging/registration, a producer can generate new product tags. To make it as

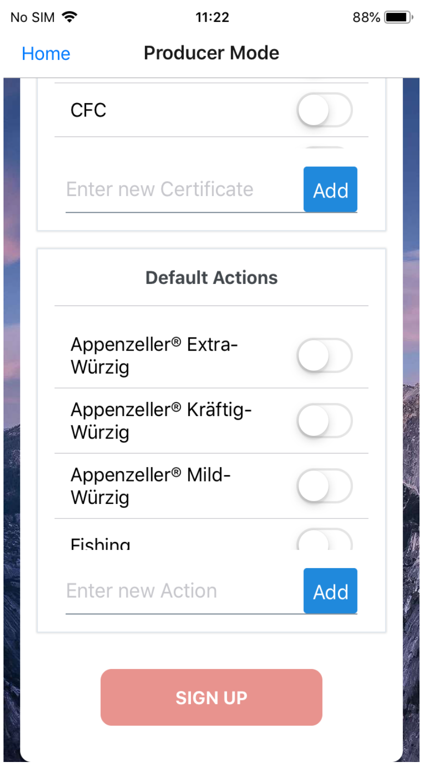


Figure 2.14: Producer Sign-up Screen 3

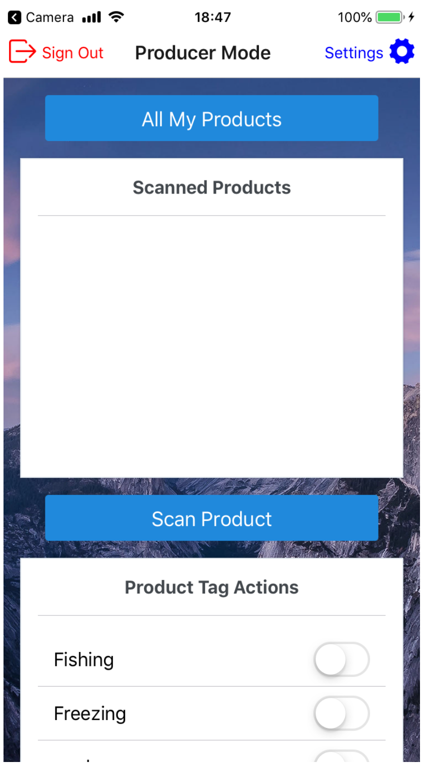


Figure 2.15: Producer Home Screen

easy as possible, the producer can perform the aforementioned action directly from its home screen.

The product tag that is about to be created can be the first one in the chain (genesis product tag), or it can have one or more previous product tags. The application is able to handle both cases.

In case the product tag depends on one or more previous product tags, the producer has an option to scan those product tags and link them with the new product tag. That can be accomplished by pressing the button **Scan Product** from figure 2.15. By doing that, the QR scanner will become active as shown in figure 2.16. All the producer has to do is to scan the QR code. Once the QR code is successfully scanned, the application will decode the QR code in text format and send a request to the server for validation. If the QR code is valid, the server will return the information about the product being scanned and the popup from figure 2.17 will be shown. The producer can continue scanning the products if he wishes, the application will maintain a list of scanned products. Should the scanned QR code is invalid, the appropriate error message will appear.

Once the producer is done with the product scanning, it can close the QR scanner. By doing so, it will be returned to its home screen from figure 2.18. It can be seen that the list of scanned product is shown. The producer can show the details about each of the product and also there is a remove button for each of the product. The next step is to select the actions being performed on the product. The list of actions is just below the **Scan Product** button, figure 2.19. The default actions are preselected. The producer can remove/add actions by toggling the button for each of them. The final step is to press

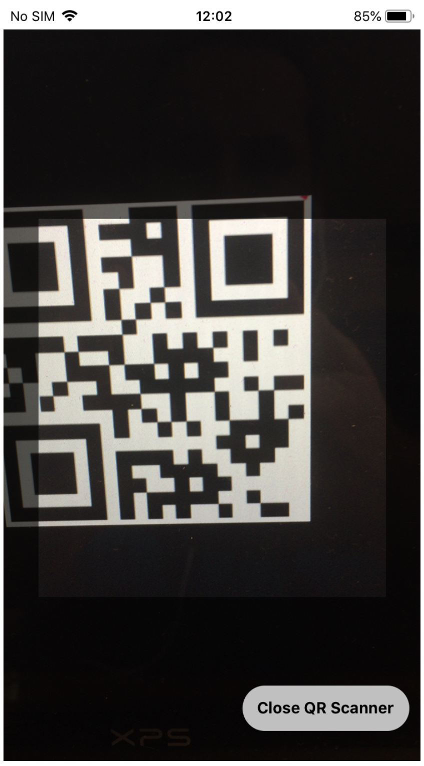


Figure 2.16: QR Scanner

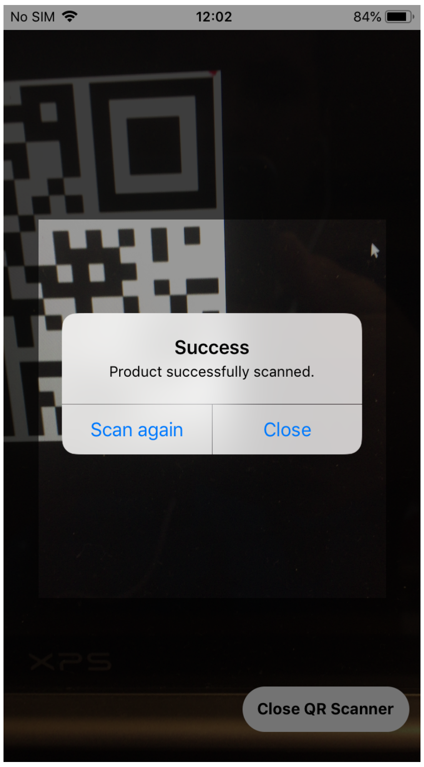


Figure 2.17: QR Scanner - Valid Product Tag

Generate Product Tag from figure 2.19. The application will make an HTTP request against the server and as the payload will contain all the information about the product:

- Producer information
- JWT token for authentication
- List of previous product tags
- List of actions
- Geoinformation

The server will validate the data being sent and if the data is valid, it will generate a new product tag and its hash value. The hash value will be also stored to the Smart Contract that is deployed on the Ethereum network. Once the data is successfully stored in the database, the server will return the success message, the message will include the data about the new product tag. On successful product tag generation, the popup from figure 2.20 will appear.

The popup will say that the product tag is successfully generated, and will show the **Print**, **Show Details** and **Close** buttons. Since the server returned the hash value of the new product tag, the producer can print it in its QR code format. By selecting the **Print** button, the page with printer options will appear as shown in figure 2.21. The producer can select a printer and basic printing options, and it is also able to preview the

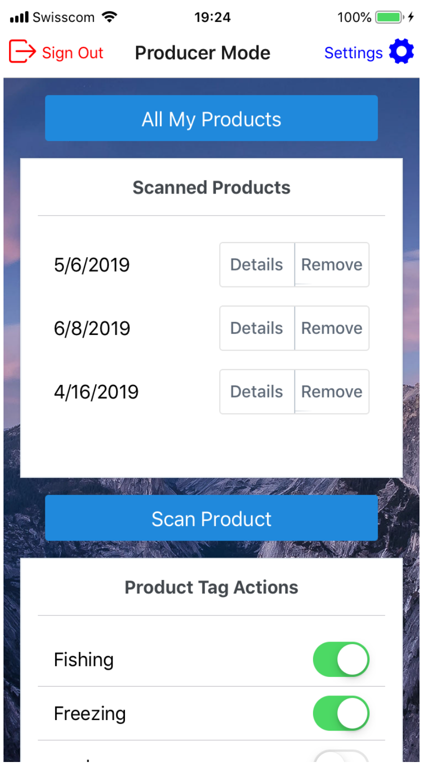


Figure 2.18: List of scanned products

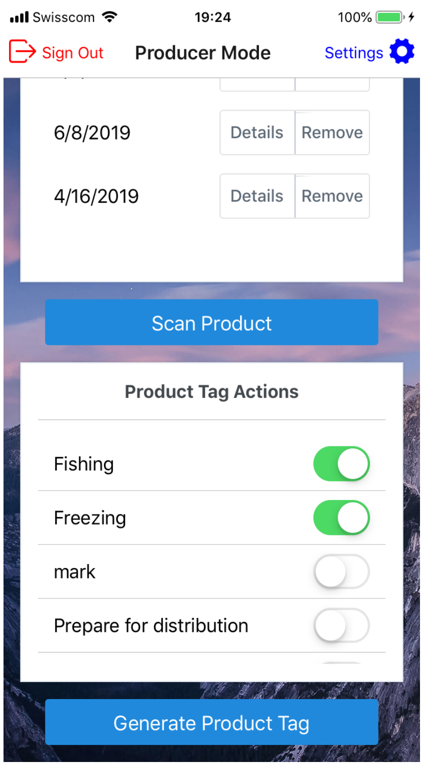


Figure 2.19: Product Tag actions and Product Tag Generation

QR code. Once the producer presses the **Print** button from the top-right corner from figure 2.21, the application will connect to the printer and trigger the printing process.

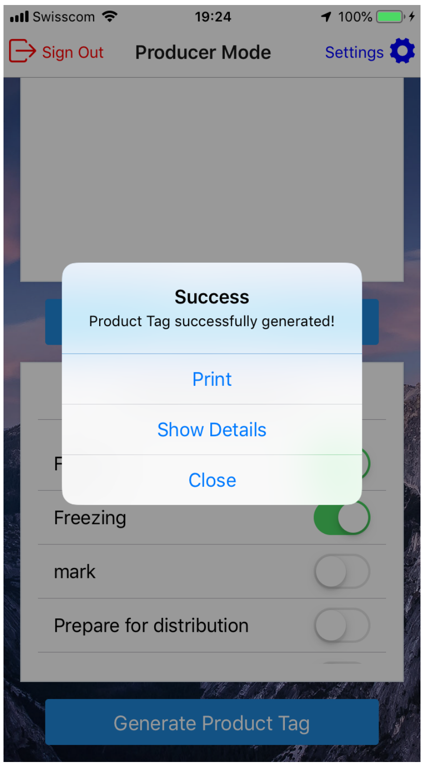
The popup from figure 2.20 offers the option to show the product tag details upon successful generation. This option offers the same functionalities as the one from the Consumer's mode where a consumer wants just to validate a product. It will be covered in section 2.3.2.

In case the producer is creating a product tag that is the first one in the chain, all it has to do is to select the actions being performed on the product, and to click on the **Generate Product Tag** button from the figure 2.22. The producer will be warned that it is going to create a product tag without any previous product tag as shown in the figure 2.23.

In the warning popup from the figure 2.23, the producer can choose to create a product tag or to scan previous product tags. By selecting the option to create a product tag, the application will make a request against the server, which will validate the data being sent. This flow is the same as the case above where there is one or more previous product tags. Upon successful product tag generation, the producer will see the same popup as in figure 2.20.

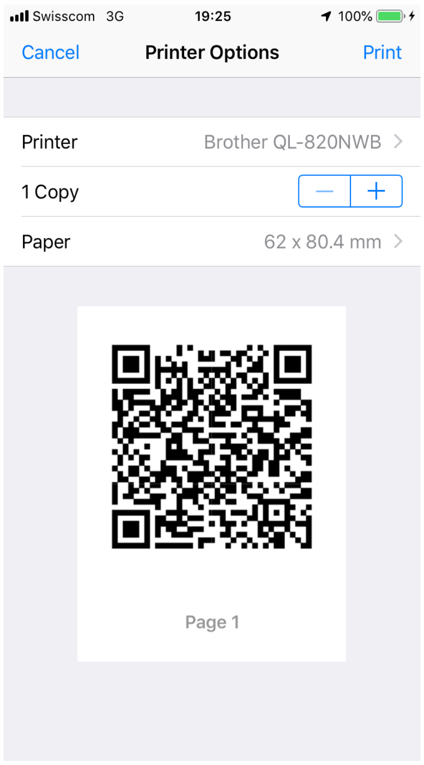
Product History Page

The top button in producer's home screen from figure 2.15 leads to the page that lists all the products ever being generated by that producer. The figure 2.24 shows the list of



Success
Product Tag successfully generated!

Figure 2.20: Product Tag successfully generated



Printer: Brother QL-820NWB
1 Copy
Paper: 62 x 80.4 mm

Figure 2.21: QR Code Print Page

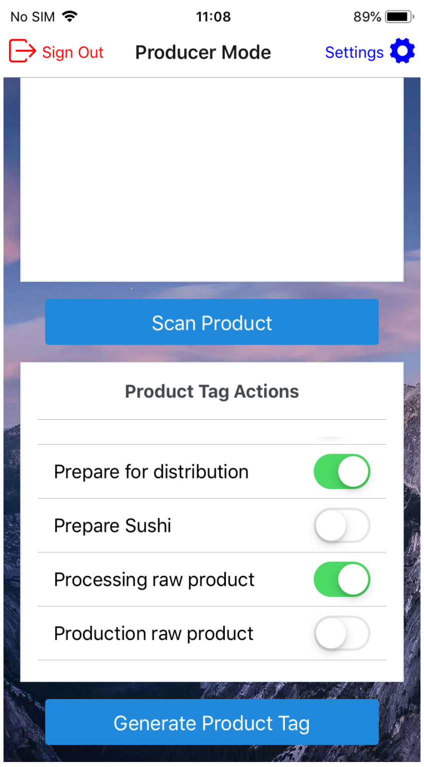
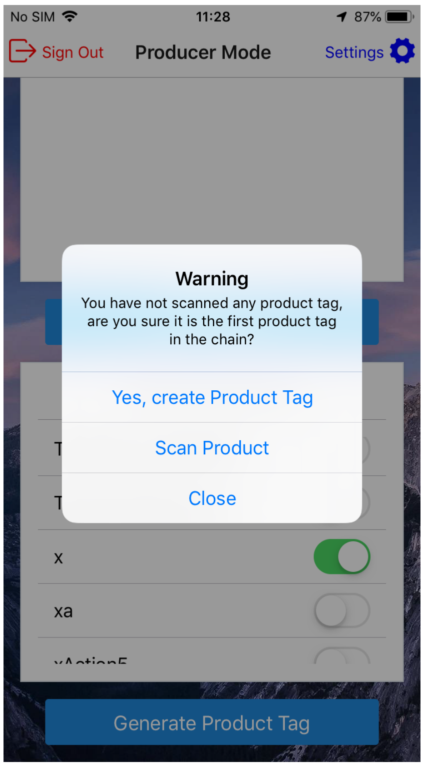


Figure 2.22: Generate Product Tag Button



Warning
You have not scanned any product tag, are you sure it is the first product tag in the chain?

Figure 2.23: No previous Product Tags Warning

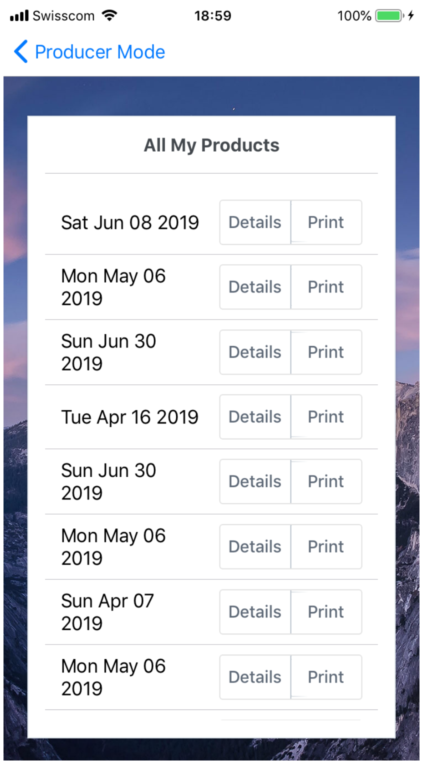
product tags sorted in chronological order. For each product tag there are two option:

- **Details**

  Producer can show the product tag details in a same way as a Consumer. It will open a map view as shown in figure 2.29. More details will be given in section 2.3.2.

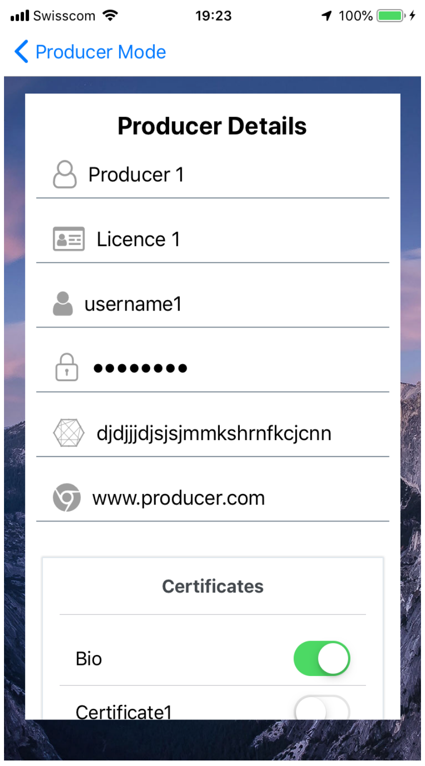
- **Print**

  There is also an option to print again the QR code of the specific product tag if needed.



Sat Jun 08 2019	Details	Print
Mon May 06 2019	Details	Print
Sun Jun 30 2019	Details	Print
Tue Apr 16 2019	Details	Print
Sun Jun 30 2019	Details	Print
Mon May 06 2019	Details	Print
Sun Apr 07 2019	Details	Print
Mon May 06 2019	Details	Print

Figure 2.24: Producer's Product History Screen



Producer Details

Producer 1

Licence 1

username1

••••••••

djdjjjdjsjsjmmkshrnfkcjcnn

www.producer.com

Certificates

Bio ✓

Certificate1

Figure 2.25: Producer Update Screen

Producer Settings Page

Producer can enter the settings page by clicking on the **Settings** button on the top-right corner on the home screen from figure 2.15. This will open the page from figure 2.25. The page contains the same form as the sign up page, just the values are already filed with producer's details. The producer can update each of the values, add/remove certificates and default actions. By pressing the **Update** button, the application will send the data to the server, which will update the producer's data in the database.

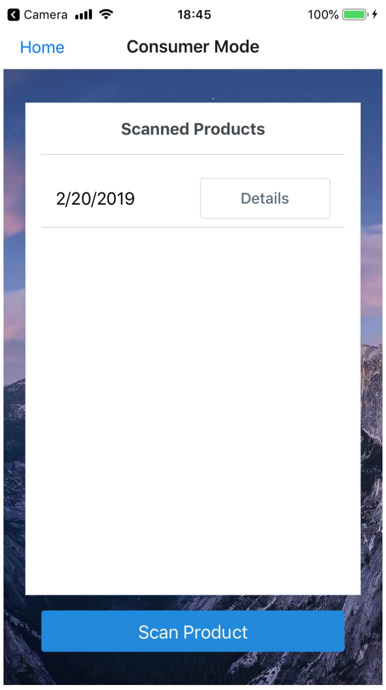
2.3.2 Consumer Mode

Consumer Mode makes it as easy as possible for the final consumers to validate the products. The consumer page can be accessed by clicking on the **Consumer** button from the startup screen from figure 2.10. This will open the page from figure 2.26. This page provides just the list of scanned products and the **Scan Product** button of the bottom of the page.

By clicking on the **Scan Product** button, the QR scanner page will be open, as shown in figure 2.16. If the product is not valid, the error popup will appear, as shown in figure 2.27. Otherwise, the popup from figure 2.28. The success popup 2.28 will be shown. Consumer can right from the popup continue scanning or it can show the product details. By choosing to show the details, the map view will appear as in figure 2.29. This example show three pins on the map. Each pin represents one product tag. The green pin represents a product tag that is scanned, while red pins represent previous product tags. The pins are numbered, where the lowest number represents the oldest product tag in the chain and the highest number represents the latest product tag, namely the one we scanned. This means that all the polylines finaly lead to the latest (scanned) product tag. The polilynes always form a tree. The leafs represent the genesis product tags and the root is the product tag we scanned.

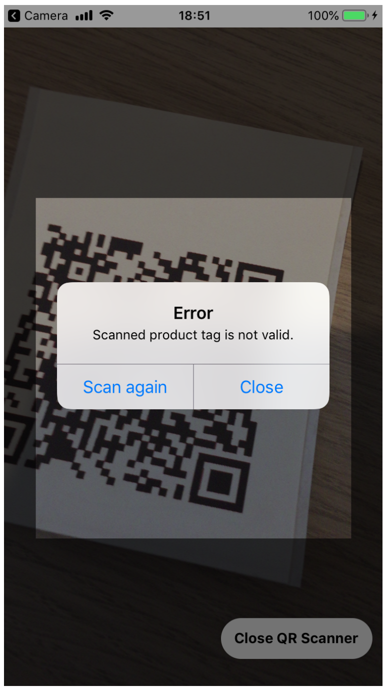
By clicking on each of the pin, the popup with product tag details will appear, as in figure 2.30. The popup show the following information about the product:

- Date when the product tag is generated.
- Product tag hash.
- Product tag actions.
- Product tag producer.
- The list of the previous product tags, if any.



2/20/2019

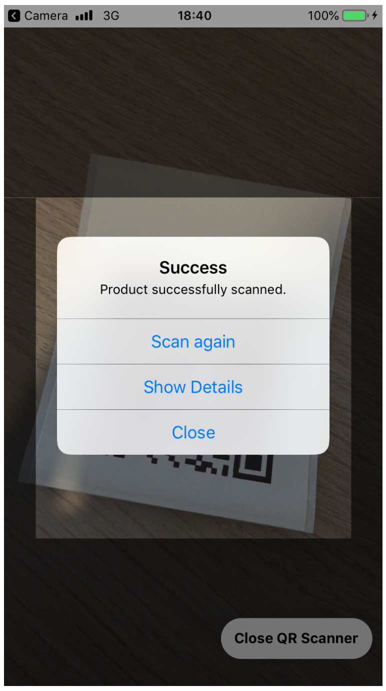
Figure 2.26: Consumer Screen



Error

Scanned product tag is not valid.

Figure 2.27: Invalid QR Code



Success

Product successfully scanned.

Figure 2.28: Valid QR Code

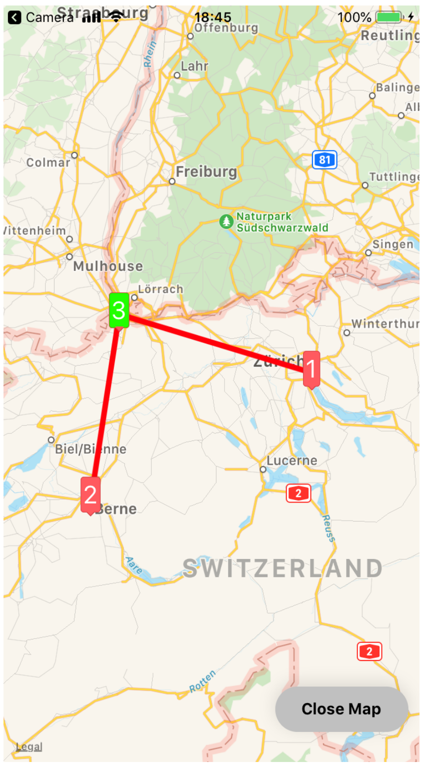
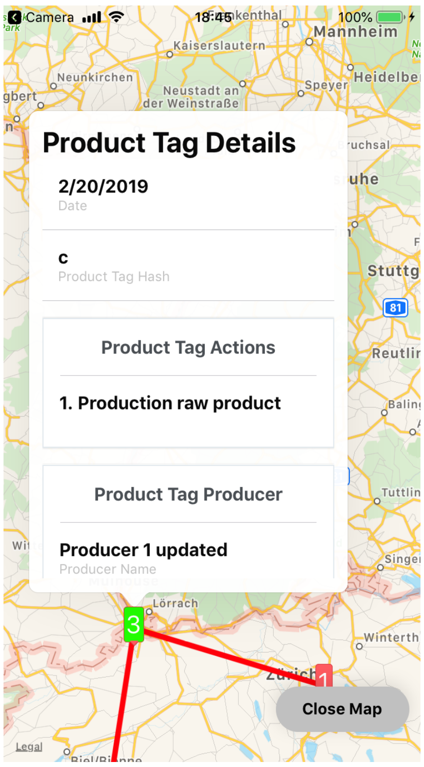


Figure 2.29: Map View



Product Tag Details

2/20/2019

Date

c

Product Tag Hash

Product Tag Actions

1. Production raw product

Product Tag Producer

Producer 1 updated

Producer Name

Figure 2.30: Map View Details

Chapter 3

Evaluation and System Analysis

This chapter describes the quality of the mobile application's source code and its robustness. The focus is put on the characteristics of the application while running on the iOS devices since that was the main intent of this project, to bring the Food Chain Tracking System to the iOS users.

The application was developed on the iPhone 6 device and it is also tested on the iPhone X device. On both devices the application is stable without unexpected crashing even after heavy usage. All the functionalities that are implemented in the Android application are also implemented in the React Native application. The focus was put on the user interface and experience so some UI elements and user flows were improved.

The application was used for demonstrating the FC Tracking Platform capabilities on several events. One of them was the IEEE International Conference on Blockchain and Cryptocurrency (14-17 May 2019, Seoul, South Korea) [22]. The application was demonstrated as a part of the SC Tracking Platform, and the paper was published under the title *A Platform-independent, Generic-purpose, and Blockchain-based Supply Chain Tracking* [23]

The application has been tested and presented on several occasions within the Communication Systems Group [24]. All the inputs were collected and taken into account so the final version meets all the requirements while stays stable.

One of the main issues in the first version of the application was that the application was crashing in an unpredictable manner while scanning the product tags. It turned out that the problem was with the improper handling on the Redux Thunk asynchronous actions. The reason was that the Redux global state was not updated properly on some occasions so the second action for fetching the product data was triggered while the first one had not been resolved. This issue was solved after heavy debugging and the solution was to move the handling of product data fetching locally within the QrScanner component class state instead handling it with Redux.

Other relevant suggestions are:

- Add proper test cases. One of the possible test case would be to test the user registration process and obtaining the JWT token. The test case could be implemented in a way that all the form fields are populated in an automatic manner and it is validated that the JWT token is obtained and stored properly upon successful registration.

- Send the Product Tag hash values to the smart contract directly from the mobile application instead of delegating it to the server. Figure 3.1 shows the current approach where the server application sends the new hash value to the smart contract. The better approach would be that the mobile application should send the new hash value to the smart contract, which would enforce the producers to pay for the transactions. The problem is that at the time of developing the application, the Web3j library did not work well with React Native.

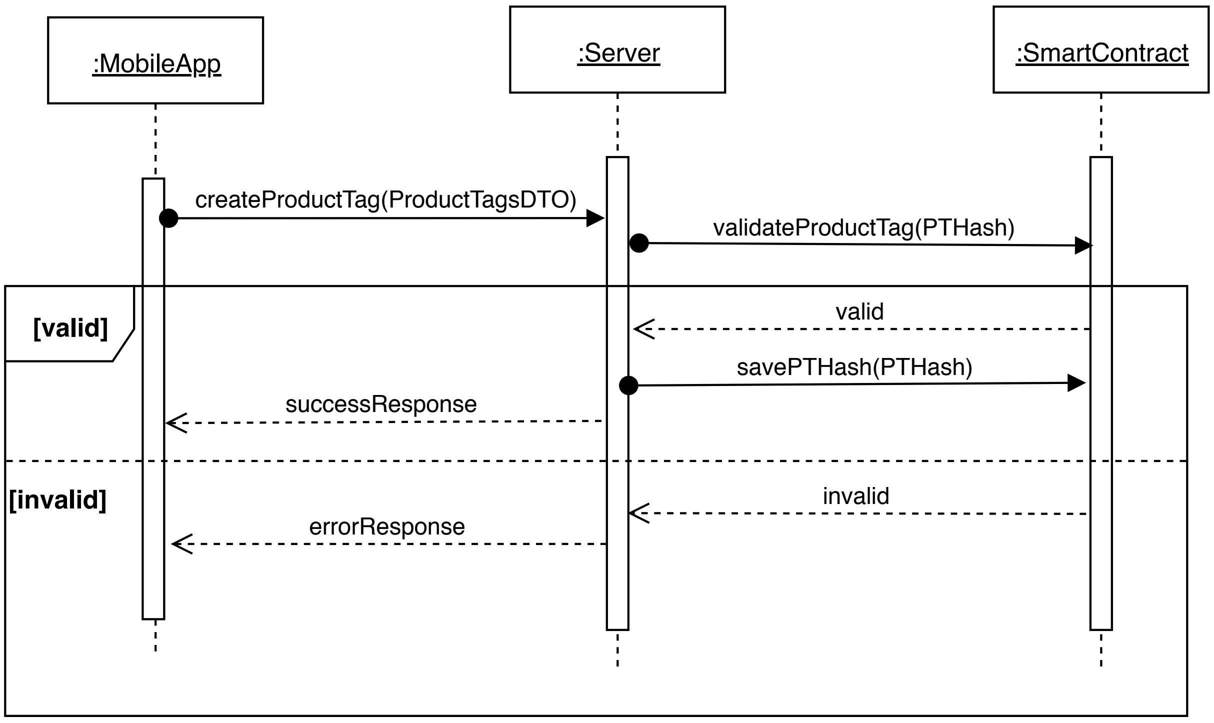


Figure 3.1: Product Tag Generation - Server sends the new hash value to the smart contract

- Improve form validation. Both Login and Sign UP forms could have better validation for their fields. E.g. the password fields could require more strict passwords (minimum length, symbols, uppercase/lowercase letters).

- Store the JWT token in the phone's local storage instead of in Redux store. This would allow automatic login even if the application had been closed and reopened.

Even though the main goal of this IS was to make an iOS application, the React Native framework builds both Android and iOS applications from the same code base. It is important to mention that the application behaves much better on the iOS platform because the React Native framework is much better optimized to work on iOS. The main issue with the Android platform is the nested scroll view. If there is a nested scroll view,

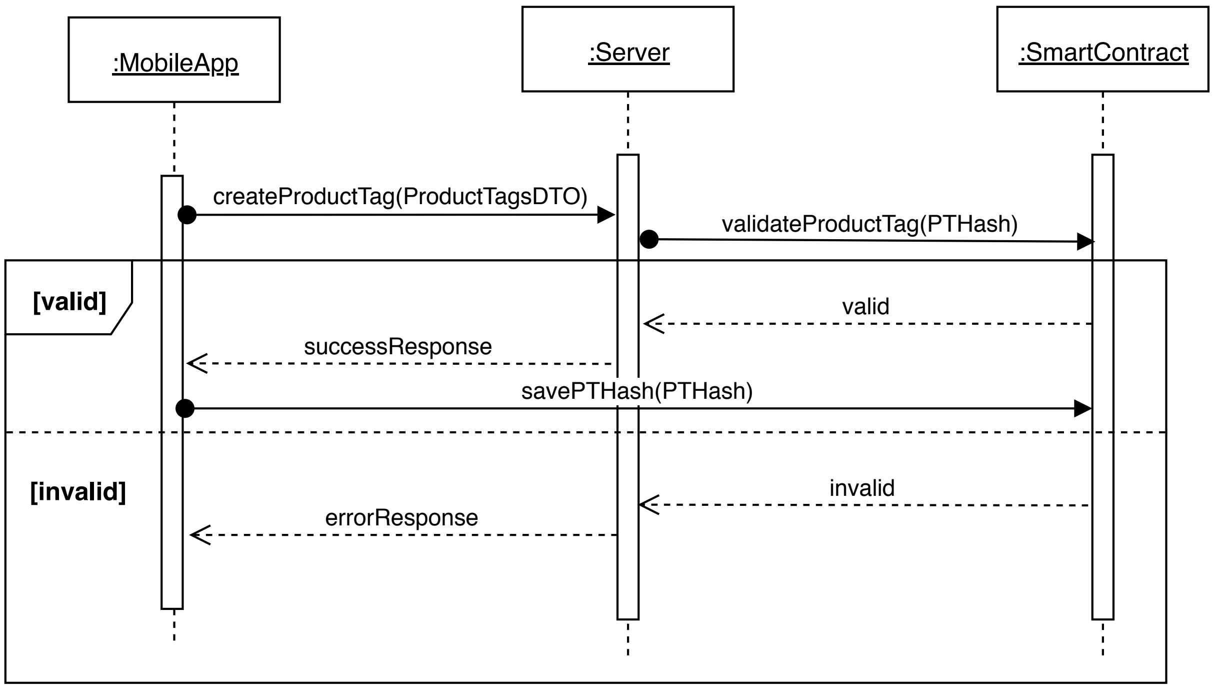


Figure 3.2: Product Tag Generation - Mobile application sends the new hash value to the smart contract

like the action list withing the Sign-Up form, only the outer scroll view will work, and the event on the inner scroll view will never be triggered. There are several workarounds to avoid this issue, but none of them is a clean and proper solution. All in all, the application is much more fluid and stable on iOS than on Android platform.

Chapter 4

Summary and Future Considerations

Food Chain Tracking Platform enables the more transparent and traceable running of the Supply Chain business. That leads to the mutual trust not just between between the parties withing the chain, but between them and final consumer which is especially important today when many scandals in that industry have occurred. This platform and the mobile application help to improve the business flow by using top-notch technologies such as Blockchain, without introducing significant costs. The data about the product is easy accessible and impossible to forge due to the Blockchain nature.

One of the main goals of this Independent Study was to bring the FC Tracking System to the iOS market while integrating into the existing platform. It allows both consumers and producers to use the platform in an easy way.

One of the intentions was to examine the capabilities of the React Native framework and it has been concluded that it meets the expectations. The target applications on both platforms are quite stable with almost native performances. Maintaining one code base for both platform is a clear benefit. As written in chapter 3, the React Native does not work as good on Android platform as on iOS. The possible solution would be to eject the application from the Expo managed flow 2.2.4 and write the Android native modules for the parts that do not work well. This breaks the React Native principles since the code in Java or Cotlin should be written, but it is there as an option. This would increase the code base and would introduce a new language, but it would solve the compatibility problems on the other hand.

It has to be mentioned that this application is a functional prototype. It is not production-ready mostly because of the suggestions made in chapter 3. Once the changes are implemented based on the suggestions, the application will be one step closer to production readiness.

Bibliography

[1] THE NETWORK EFFECT Beyond Supply Chains, FIXING THE 5 BIG PROBLEMS IN THE FOOD SUPPLY CHAIN `https://supplychainbeyond.com/5-big-problems-in-the-food-supply-chain/`, Last visited August 2019.

[2] The Wikipedia Website, Radio-frequency identification `https://en.wikipedia.org/wiki/Radio-frequency_identification`, Last visited August 2019

[3] GIS AS A DECISION SUPPORT FOR SUPPLY CHAIN MANAGEMENT, Sanjay Kumar and Suneeta Agrawal, `https://pdfs.semanticscholar.org/5841/cf5a366bc442415c9fdd2728696f1d0cdf25.pdf`, Last visited August 2019

[4] The Wikipedia Website, Global Positioning System `https://en.wikipedia.org/wiki/Global_Positioning_System`, Last visited August 2019

[5] Techopedia Website, System Design page `https://www.techopedia.com/definition/29998/system-design`, Last visited June 2019.

[6] Design and Development of anAndroid-based Supply ChainTracking Application, Atif Ghulam Nabi

[7] Learning React Native by Bonnie Eisenman, Oreilly Safary, `https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html`, Last visited July 2019

[8] Flutter Official Website, `https://flutter.dev`, Last visited July 2019

[9] Ionic Framework Official Website, `https://ionicframework.com/`, Last visited July 2019

[10] React Website, `https://reactjs.org/`, Last visited July 2019

[11] Redux Website, `https://redux.js.org/`, Last visited July 2019

[12] Pure functions in JavaScript, `https://www.nicoespeon.com/en/2015/01/pure-functions-javascript/`, Last visited July 2019

[13] React Redux website, Official React bindings for Redux `https://react-redux.js.org/`, Last visited July 2019

[14] Hackernoon website, Redux Step by Step: A Simple and Robust Workflow for Real Life Apps `https://hackernoon.com/redux-step-by-step-a-simple-and-robust-workflow-for-real-life-apps-1fdf7df46092`, Last visited July 2019

[15] Github website, Redux Thunk, Thunk middleware for Redux. `https://github.com/reduxjs/redux-thunk`, Last visited July 2019

[16] Material-UI Website, `https://material-ui.com/`, Last visited July 2019

[17] Understanding the React Native bridge concept, Marvin Frachet, Hackermoon Website `https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8`, Last visited July 2019

[18] Out-of-Tree Platforms, React Native official Docs `https://facebook.github.io/react-native/docs/out-of-tree-platforms`, Last visited July 2019

[19] Getting Started, The React Native CLI, React Native Official Docs `https://facebook.github.io/react-native/docs/getting-started.html#the-react-native-cli-1`, Last visited August 2019

[20] Expo Official Website `https://expo.io/`, Last visited August 2019

[21] Expo Official Docs `https://docs.expo.io/versions/v34.0.0/introduction/managed-vs-bare/`, Last visited August 2019

[22] IEEE International Conference on Blockchain and Cryptocurrency, Homepage `http://icbc2019.ieee-icbc.org/`, Last visited September 2019

[23] A Platform-independent, Generic-purpose, and Blockchain-based Supply Chain Tracking, IEEE `https://ieeexplore.ieee.org/document/8751415`, Last visited September 2019

[24] Department of Informatics - Communication Systems Group, University of Zurich `https://www.csg.uzh.ch/csg/en/`, Last visited September 2019

Abbreviations

REST	Representational State Transfer
API	Application Programming Interface
SQL	Structured Query Language
QR	Quick Response
SC	Supply Chain
IS	Independent Study Module
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
iOS	An operating system used for mobile devices manufactured by Apple Inc.
UI	User Interface
CLI	Command Line Interface
MAC	A family of PCs designed, manufactured and sold by Apple Inc.
PC	Personal Computer
JWT	JSON Web Token
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers

List of Figures

2.1 System Design Diagram [6] 5
2.2 Use Case Diagram 6
2.3 Container Diagram for Food Chain Tracking System. 7
2.4 React Native compiling flow [17] 8
2.5 Bridge sitting between JavaScript and Native Threads [17] 8
2.6 Redux Flow[14] 9
2.7 Home screen drawer 10
2.8 Home screen - German translation 10
2.9 Expo Managed Workflows [21] 11
2.10 Home screen 13
2.11 Producer Login Screen 13
2.12 Producer Sign-up Screen 1 13
2.13 Producer Sign-up Screen 2 13
2.14 Producer Sign-up Screen 3 14
2.15 Producer Home Screen 14
2.16 QR Scanner 15
2.17 QR Scanner - Valid Product Tag 15
2.18 List of scanned products 16
2.19 Product Tag actions and Product Tag Generation 16
2.20 Product Tag successfully generated 17
2.21 QR Code Print Page 17

2.22 Generate Product Tag Button 17

2.23 No previous Product Tags Warning 17

2.24 Producer's Product History Screen 18

2.25 Producer Update Screen 18

2.26 Consumer Screen 20

2.27 Invalid QR Code 20

2.28 Valid QR Code 20

2.29 Map View 20

2.30 Map View Details 20

3.1 Product Tag Generation - Server sends the new hash value to the smart contract 22

3.2 Product Tag Generation - Mobile application sends the new hash value to the smart contract 23

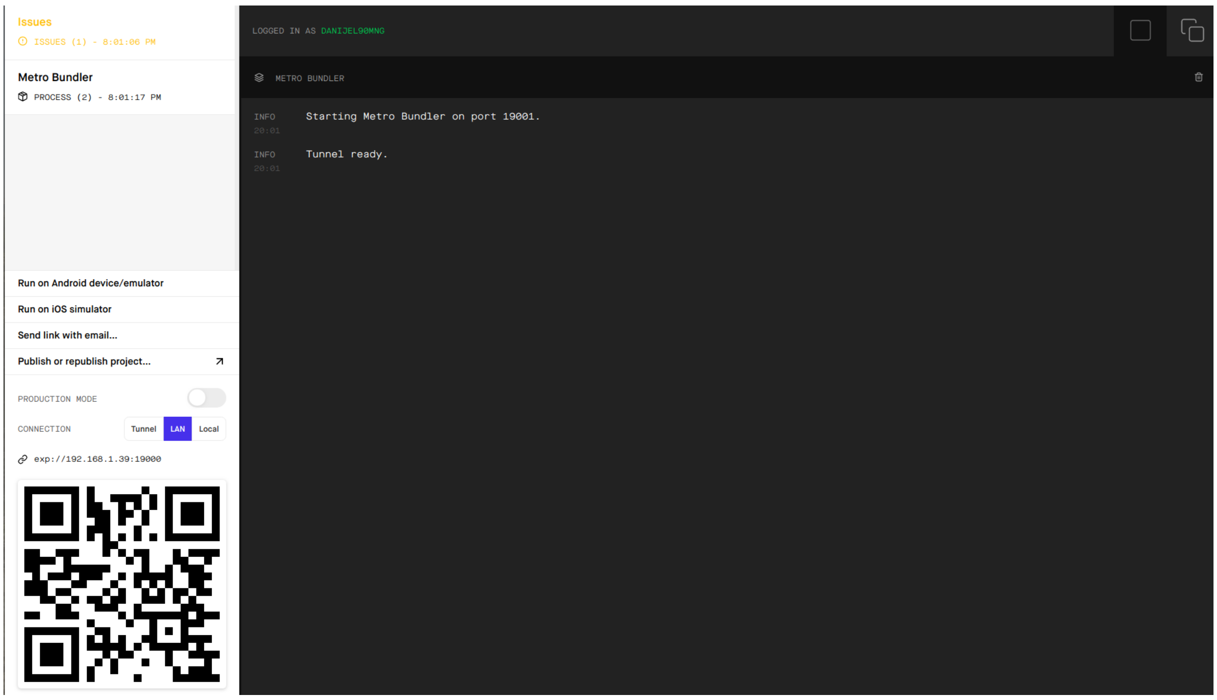
A.1 Expo Development Server User Interface 30

Appendix A

Installation Guidelines

Since we do not have an Apple developer account that is not free, the application can run on a device only by building the source code from the local machine.

The easiest way to run the application is to run the expo server by running the command **expo start** from the project root directory. This will bring up the Expo development server and provide the interface in the browser as the figure A.1 shows.



```
LOGGED IN AS DANIJEL90MNG

METRO BUNDLER

INFO    Starting Metro Bundler on port 19001.
20:01

INFO    Tunnel ready.
20:01
```

Issues
ISSUES (1) - 8:01:06 PM

Metro Bundler
PROCESS (2) - 8:01:17 PM

PRODUCTION MODE: off
CONNECTION: Tunnel [✓ LAN] Local
exp://192.168.1.39:19000

Run on Android device/emulator
Run on iOS simulator
Send link with email...
Publish or republish project...

Figure A.1: Expo Development Server User Interface

The next step is to install the Expo client application on your phone. The links for the Android and iOS platforms are:

- Android: `https://play.google.com/store/apps/details?id=host.exp.exponent&hl=en`

- iOS: `https://apps.apple.com/us/app/expo-client/id982107779`

The final step is to scan the given QR code from bottom-left corner of the Expo development server. This will trigger the code build for the target platform and send it to the phone. Once the built code is sent to the phone, the Expo application will run it as it was the native application.

When you run the application for the first time, it will ask you for the permissions to access the camera and location services.

In the future, once the Apple Development Account is obtained, the Expo flow from the section 2.2.4 should be utilized to publish the application to the iOS marketplace.

Appendix B

Contents of the CD

- A copy of the Food Chain Native Client Repository (fc-native-client)
- A PDF file of the Independent Study report
- The LaTeX source code of the Master Project report
- Slides used for the final presentation