



**University of
Zurich**^{UZH}

Developing a Blockchain- based Supply Chain Tracking Platform

Atif Ghulam Nabi 15-709-116

Danijel Dordevic 17-717-778

Tanbir Mann 16-744-807

Zurich, Switzerland

Supervisor: Sina Rafati Niya, Prof. Burkhard Stiller

Date of Submission: August 26, 2019

Abstract

In recent times food safety peaked upswings of academic and commercial concerns. In the area of the supply chain, a lot of rising technologies along with the rapid growth of the internet of things (IoT) have been applied in traceability systems. Yet, the majority of such technologies are monitored centrally which has vulnerabilities regarding trust, fraud, corruption, data integrity, and false information. Whereas today's technology Blockchain (BC), a decentralized information technology brings a whole new approach in the communication systems world. BC technology is disrupting society by enabling new kinds of disintermediated digital platforms. In this project, we have developed an Android application along with a web application for supply chain management to keep track of products and their life cycles. Our backend application depends on the Ethereum node to call smart contracts and validate the product tag's hash generated by the user.

Acknowledgments

We would like to thank to our project supervisors Sina Rafati and Prof. Dr. Burkhard Stiller for unconditional help durring the project which has been essential for its successful completion. We would also like to take the opportunity and thank our families for their support during our studies.

Contents

Abstract	i
Acknowledgments	ii
Contents	iii
1 Introduction	1
1.1 Motivation	2
1.2 Description of Work	3
1.3 Thesis Outline	3
2 Architectural Overview	4
2.1 System Context	4
2.2 System Architecture	5
2.3 Source Code Structure	8
3 Server Application	9
3.1 Basic Overview	9
3.2 Layers	10
3.2.1 Data Persistence Layer	12
3.2.2 Data Access Layer	14
3.2.3 Service Layer	16
3.2.4 Controller Layer	16

4	Smart Contract	25
5	Android Application	28
5.1	Design and Implementation	29
5.2	System Design and Overview	30
5.2.1	Use Case Diagram	31
5.2.2	Code Architecture	32
5.2.3	RESTful APIs	33
5.3	Implementation	35
5.4	Producer Mode	36
5.4.1	Authentication	36
5.4.2	Home Page	37
5.4.3	Multi QR Generation	38
5.4.4	Multi QR Scanning	39
5.4.5	QR Information Page	39
5.4.6	Interactive Map View	40
5.4.7	QR History Page	41
5.4.8	System Settings	41
5.4.9	Multi Language Support	42
5.5	Consumer Mode	44
5.6	Printing Module	44
5.7	P2P Payment System	47
5.7.1	Integration	48
6	Design and Implementation	51
6.1	UI Enhancement	51
6.1.1	Layout	51
6.2	App Generalization	57
6.2.1	Registration	57
6.2.2	Login	61
6.2.3	Setting	62

7	Security Analysis	67
7.1	Security Background	67
7.1.1	Security Concepts	67
7.2	Security Assessment	69
7.2.1	Scope	69
7.2.2	RSA Encryption	71
7.2.3	Retrofit	74
8	Scalability	76
8.1	Device Compatibility	76
8.1.1	Platform Versions	77
8.2	Screen Compatibility	79
8.2.1	Screen Size	79
8.3	Test Scenarios	81
8.3.1	Performance Test:	81
8.3.2	Load Test:	83
9	Web Application	84
10	Software Build and Deployment	87
10.1	Local Deployment	87
10.2	Cloud Deployment	89
11	Modification Examples	95
11.1	Server Application	95
11.2	Android	96
11.3	Web Application	96

12 Evaluation and System Analysis	97
12.1 Software Development	97
12.2 Unit Testing	98
12.3 Security	98
12.4 Robustness	98
12.5 Challenges	99
12.6 Onsite-Experiments	100
13 Summary and Future Considerations	101
13.1 Future Considerations	102
Bibliography	103
Bibliography	103
Abbreviations	108
List of Figures	108
List of Figures	109
List of Tables	111
List of Tables	112
A Installation Guidelines	113
B Contents of the CD	115

Chapter 1

Introduction

Demand for transparency is increasing, surprisingly we know little about most of the products we use in our day to day life [1]. Products travel frequently through a huge network of stakeholders before reaching the end consumer which includes retailers, distributors, transporters, storage facilities, and suppliers that participate in design, production, delivery, and sales, etc., still in almost all the cases these journeys remain an unobserved attribute of our possession.

Consumers as well the government's demands for more transparency from brands, manufacturers, and producers throughout the supply chain is increasing. Governments tend to provide more transparency in goods and their manufacturing details. "The 2016 Food Revolution Study, which surveyed 1,522 consumers to discover how they make food choices, shop and what they expect from brands in terms of product information, reveals that brands that to meet customer expectations for product information and deliver that information instantly develop a new dynamic of convenience, trust, and long-term value" [2]. This leads to increasing demand for producers to provide complete information about the product and its origin.

Internet of Things (IoT) is one of the most significant disruptive technologies of this century expanding at a fast pace. IoT provides a platform for devices e.g., smartphones, sensors, and wearables to connect devices via the Internet to share data. The devices in IoT can be controlled remotely to perform the desired functionality. The information sharing among the devices then takes place through the network which employs the standard protocols of communication. But this technology comes with various security and privacy challenges due to its massive size [3]. The deficiency of fundamental security safeguards in many of the first generation IoT devices have aggravated the privacy concerns related to IoT products. Many secure network protocols like IPsec and TLS are used to provide authentication and privacy, but these methods are computationally expensive which limits their versatility with resource-limited IoT devices [10]. In order to have control over the access of sensitive information, a distributed capability-based access control method is proposed in [11], but this method also compromises user privacy due to its excessive delays and overheads. Hence, there is a need to have an approach to share privacy-aware IoT data without tempering user privacy.

1.1 Motivation

Blockchain technology has attracted huge attention from the researchers of Computer Science in the past decade. Blockchain (BC) was introduced in 2008, as a platform for secure, anonymous transactions, using a decentralized network of computers or devices. Its first application was cryptocurrency Bitcoin, which assures anonymity, by allowing users to transfer tokens over a peer-to-peer (P2P) network without any need of centralized authority [6]. Blockchain maintains a distributed ledger in the form of transactions and these transactions are shared among all the nodes in the network. New transactions are embedded in the network, once they are verified and confirmed by all relative nodes in the system, thus abolishing the need for a central authority. BC has many features that include distributed structure, immutability and security and privacy [4]. Blockchain also provides a solution to most of the security and privacy challenges faced by IoT [5], specifically data authentication, Integrity, authorization, and privacy. BC provides a trusted platform to its users to share their data/information without bothering about the tampering of the data or unauthorized access.

From the past decades, the consumers' trust in the food industry is fading after many food safety risk scandals e.g., mad cow disease and genetically modified food in China [7] and "horsemeat scandal" in 2013 in Europe [8]. These scandals can leave a deep impact on the development of countries' economies and can incur a great threat to the stability of society. As an outcome, consumers are more concerned about the safety and quality of food, which has drawn quite attention from academic and industrial areas. In response to such issues, many IoT technologies were used such as RFID and wireless sensors based network in the supply chain for traceability and visibility. As the IoT technology is governed by centralized based systems that develop the issues of trust and integrity of data and valuable information can be tempered by bribing the administration. To deal with such issues it is required to come up with the solutions where the data can be transferred without any centralized control keeping the key security, privacy, confidentiality issues in mind. Whereas, Blockchain provides solutions to such issues but we cannot implement BC technology directly with IoT as it has key scalability and high resource consumption issues [9].

Therefore, combining BC and IoT could be an efficient solution for traceability in supply chain management. By making use of IoT devices such as sensors for information gathering and using BC for storing the gathered information securely without worrying about data integrity. There is much-related work in this field that has shown some immense results in tracing the product lifecycle at each phase of its life (starting from producer to its end consumer). In this project, we have developed the supply chain monitoring and tracking system to track the lifecycles of different products via an android application. The developed application is for both producers as well as consumers, producers will use the app to provide relevant and true information about the products by storing their information in a QR code. The QR generated by any producer is accessible to any other producers to perform an update on it by adding his set of actions on the product and this chain goes on from one stakeholder to another until the product is at its final stage. Whereas our consumers will scan this final QR of the product to see its lifecycle. The main goal of this project is to have a stable and robust system to keep track of a product

and it's detailed to give true information about a product to its end consumer.

1.2 Description of Work

The aim of this project to have a Platform to monitor SCM processes and to meet key security and scalability challenges of developing a peer-to-peer system that enables supply chain tracking and includes direct payments. The focus of the examination is on the data flows between components of the system, the exposed Application Programming Interfaces (API's), scalability and security. This technical document introduces the functionalities of the Food Chain App and guides on how the actors can easily create QR code for a product and customers can scan QR code for tracking the history of the product. It describes by listing the step-by-step procedure of how a user can register with food chain application as a producer, register the licenses and actions as producer which enables adding trust to the documented production transport- processing process for all interested stakeholders, how to link up previous QR code of product to the newly generated QR code and as a consumer how to see the whole life cycle of product by scanning a QR code on product. All this is meant to give a good understanding of the Food Chain application in order to make use of it right away. Subsequently, Foodchain Application is developed along with a Web Application to fulfill the desired goals.

1.3 Thesis Outline

In Chapter 2, an overall architecture of the developed system is defined with its system context, architecture, and code structure. Chapter 3 and 4 defines the server application and smart contracts respectively. In Chapter 5, frontend implementation is described along with a peer-to-peer system to build an android based application. Whereas in Chapter 6, work is done to enhance the user interface, to make the application user-friendly and to generalize the app for various products. In Chapters 7 and 8, all security and scalability practices are analyzed. Chapters 9 and 10 define the implementation of web application and software build and deployment. In Chapter 11 and 12 modification examples are provided along with system evaluation and analysis. Chapter 13 summarize the entire work and talks about our future considerations.

Chapter 2

Architectural Overview

In this section, the architectural overview of the system is given. It helps the stakeholders to understand the intention of the system. Also, it provides a high-level shared vision of the architecture and scope of the proposed IT system.

2.1 System Context

The system context diagram defines the boundary between the system, its environment and entities that interact with it [14].

The diagram shows all the external entities that may interact with the system, the system in the middle with no details of its internal structure. The system context diagram is shown in Figure 2.1. The following section explains shortly all the components from the system context diagram.

Food Chain Tracking System It provides the main functionalities by processing, validating product tags.

Consumer Uses an Android application for generating product tags.

Consumer Uses the Web application for scanning and getting the product tag's details.

Ethereum Network Runs smart contracts that stores and validates product tag's hashes.

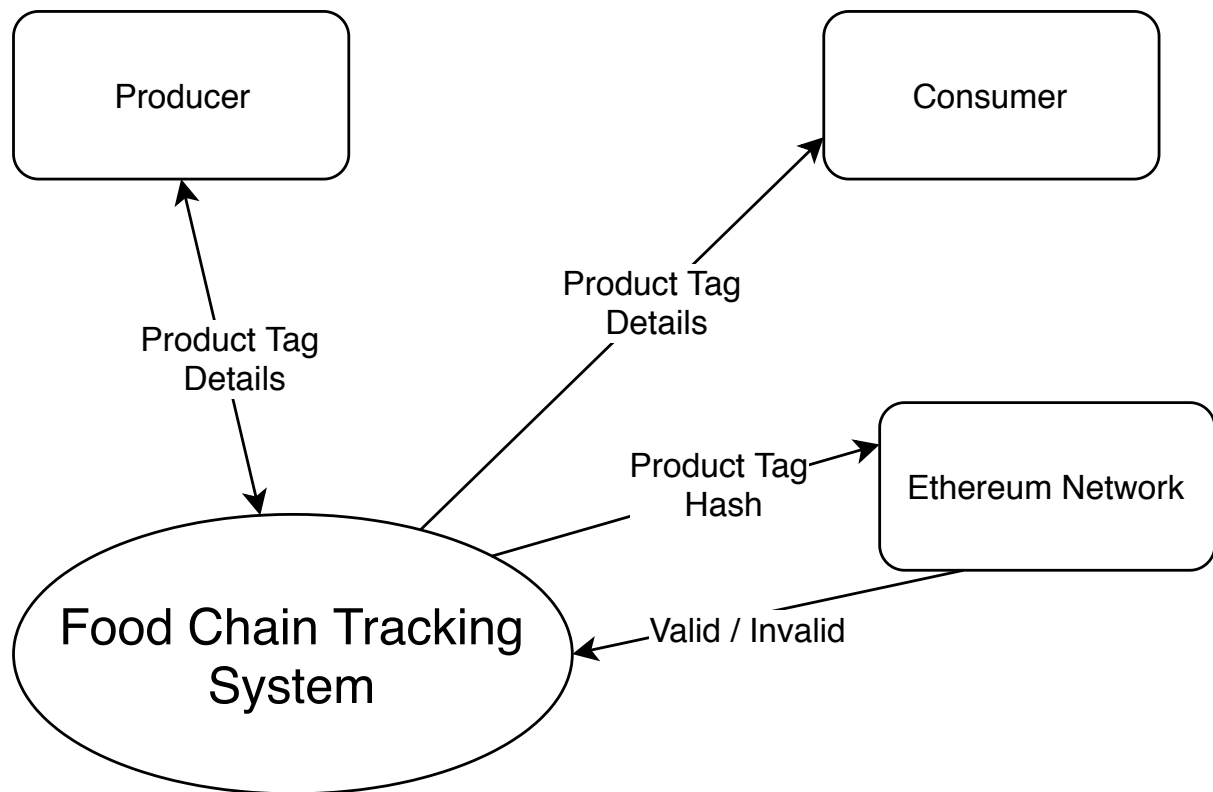


Figure 2.1: Context Diagram

2.2 System Architecture

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system sys-arc

An architectural description organizes a way that supports reasoning and about the structures and behaviors of the system. It is important to set up the proper architecture in the very beginning from the development process because changing the architecture can impact many components that depend on each other and therefore raise the costs and break the deadlines. Also, it is crucial to design a system that is easy to extend in order to support future requirements.

The system architecture is shown in Figure 2.2. It consists of system components that work together to implement the overall system. It gives a basis for analysis before the system has been built.

It can be seen that we have four main components that provide the necessary functionalities. Each of them has its section where the detail explanations are given.

Container Diagram

Figure 2.3 shows the container diagram. Essentially, every application represents one container that is runnable/deployable unit that processes, stores or serves data. In that

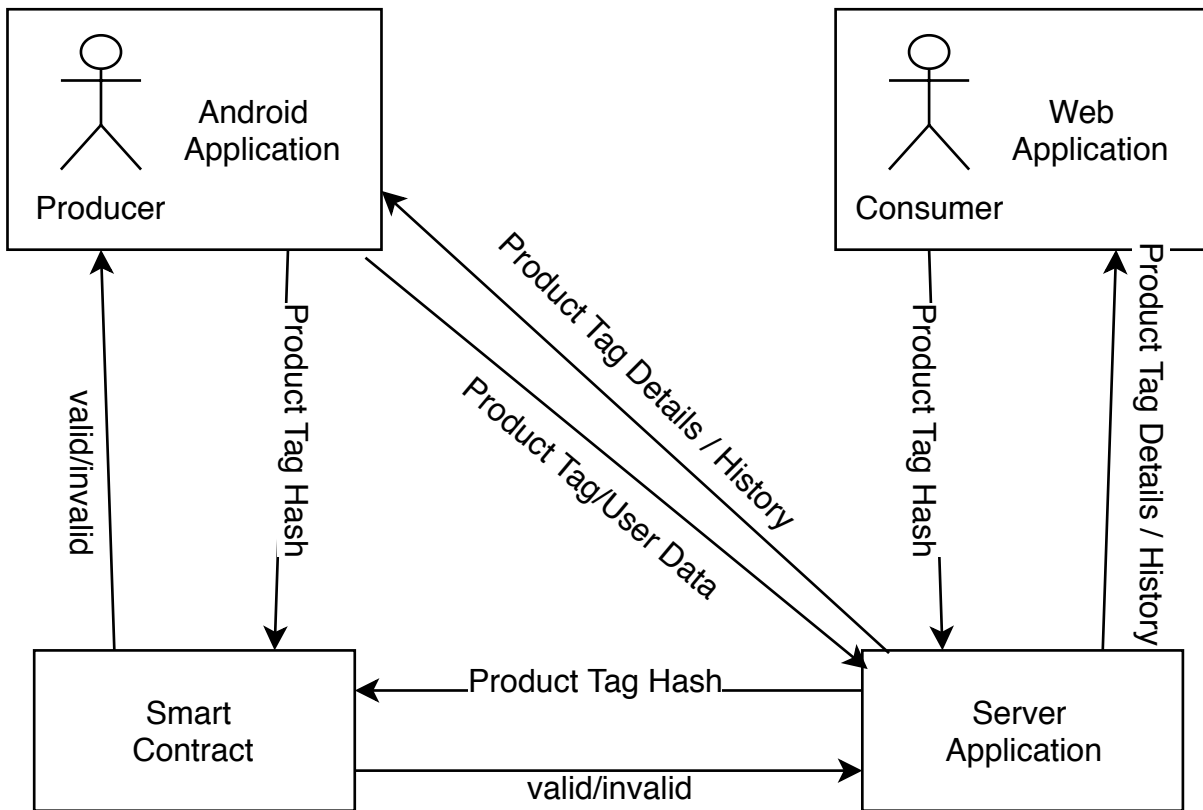


Figure 2.2: System architecture.

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 one another.

The next part shortly explains each of the components.

Database Food Chain Tracking System stores all its data in the SQL database. All kinds 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 the 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 Application 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 built with the React Native framework provides the same look and feels on both Android and iOS platforms. It can also be used by both producers and consumers. It is mentioned here because it is a part of the Food Chain tracking system but it is not the part of this Master project.

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.

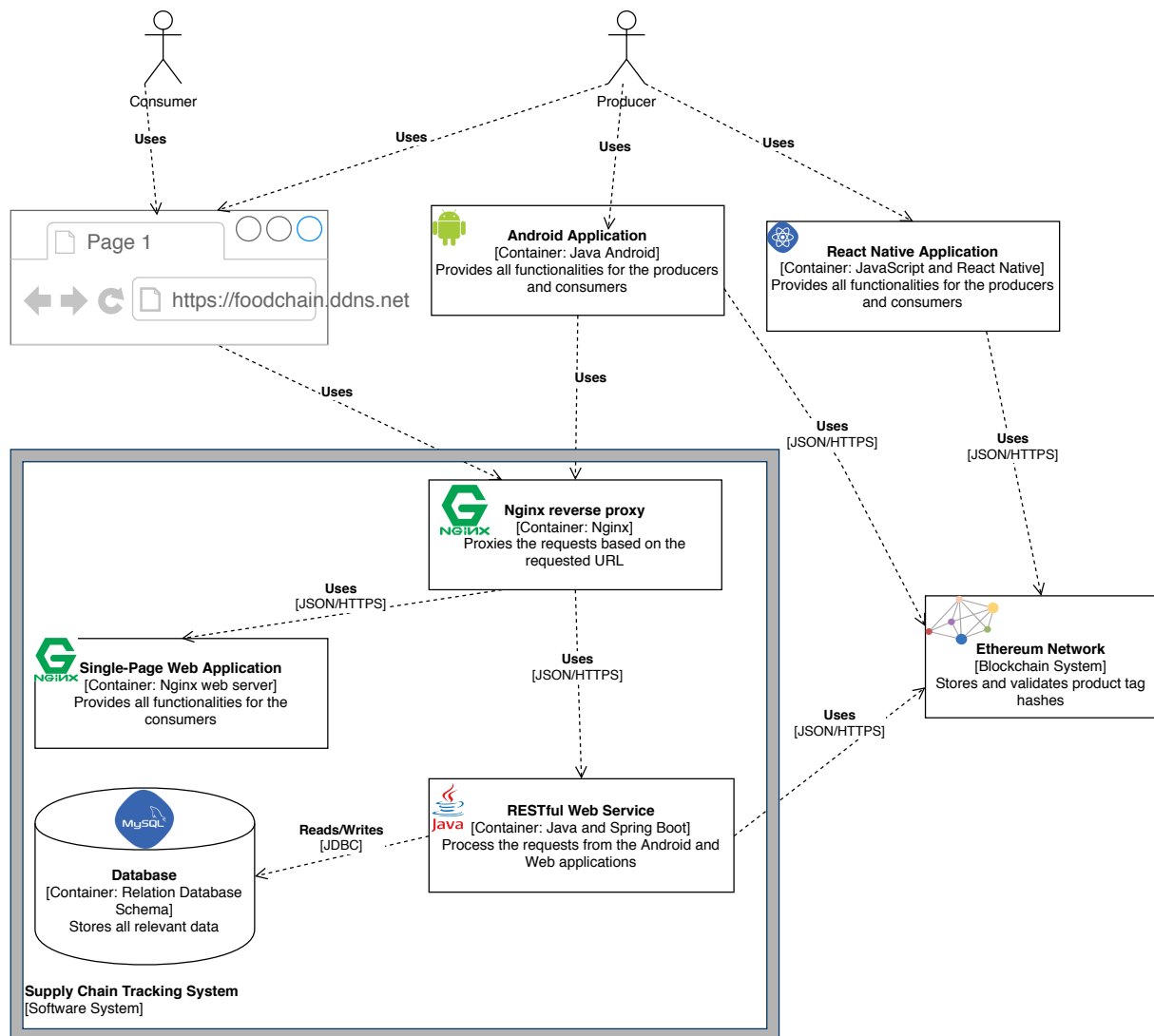


Figure 2.3: Container Diagram for Food Chain Tracking System.

2.3 Source Code Structure

Since the Food Chain Platform consists of several separate components, its source code structure is arranged accordingly. Every application (server, web, proxy) is inside its own folder. Figure 2.4 shows the folder structure of the system. While the first three folders are self-explanatory, it can be mentioned that the folder **env-variables** contains the files that contain the environment variables that are passed to the running containers. More details about the project's structure will be given in chapter 3.

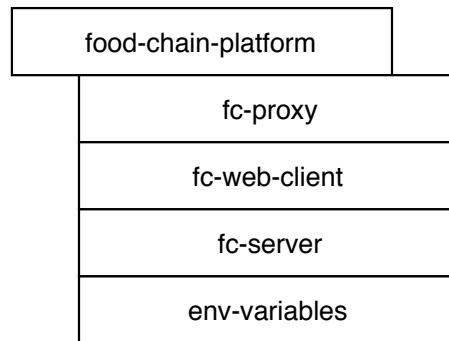


Figure 2.4: Folder Structure of the Food Chain Tracking System.

Chapter 3

Server Application

This chapter will describe the server application which provides RESTful APIs. The REST APIs provides the endpoints for the client's applications.

3.1 Basic Overview

The architecture of the server application is based on multiple layers where each layer has its responsibility. This follows the separation of concerns principle in the software architecture [25] that helps the code to be more readable and easier to maintain.

Since the project started from the ground up, we were allowed to use the newest technologies in this project and follow the best practices from the very beginning. The next part will list and shortly explain all the technologies/programming languages along with the frameworks and libraries used while implementing the server application.

Technologies used:

- Java - a general-purpose computer-programming language. It is an object-oriented class-based. [27].
- MySQL - an open-source relational database management system. We decided on MySQL because it is a de facto standard for relational databases and it has a very stable Docker image. [30].
- Nginx - a web server that can also be used as a reverse proxy, load balancer, mail proxy, and HTTP cache [28]. In this project we have two instances of Nginx, one acts as a reverse proxy and the second one serves the production build of our web application.
- Docker - a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. This becomes crucial when you have to run a system in a new environment. [49]

Frameworks:

- Spring Boot - a Spring's convention-over-configuration solution for creating production-grade stand-alone Spring-based applications. [32]. It avoids a lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.
- Spring Security - a Java/JavaEE framework that provides authentication, authorization and other security features for enterprise applications [31]. We use JWT [33] for representing claims securely between two parties, in our case between backend and android applications.
- Swagger - a framework that helps design, build, document and consume RESTful Web services [34].

Libraries:

- Web3j - a lightweight, reactive, type-safe library that enables the Java application to communicate with Ethereum blockchains [26].
- Jackson - a high-performance JSON processor for Java. It allows us to enable the communication between the applications written in different programming languages, by exchanging JSON messages [35].
- Guava - an open-source set of common libraries [36]. The main Guava's feature used in the project is its hash function. In our case, we use it to generate a hash output based on product tag's data, and print it as a label on the product represented as a QR code.
- Project Lombok - a Java library used to reduce boilerplate code for model/data objects (generating getters/setters, constructors, logs). It makes the code less verbose and more readable. [37].

As a build automation tool, Apache Maven is used [53].

3.2 Layers

Before starting explaining the layers, the package structure of the server's project is given in figure 2.4

The next part explains briefly the content and purpose of each of the packages in the alphabetical order. More details for some of the packages will be given in the corresponding sections of the layers that the packages belong to.

bootstrap It contains the initial data that is loaded on the application's startup. It preloads the initial certificates, actions, technical user.

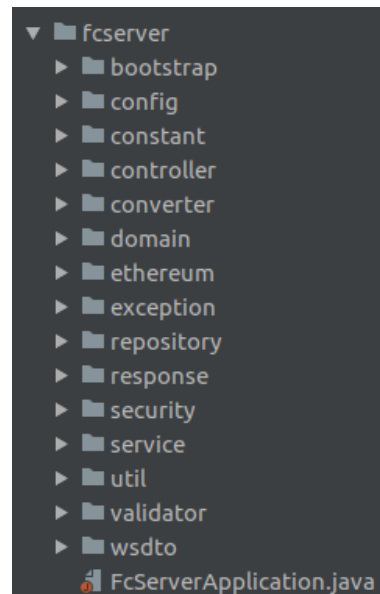


Figure 3.1: Server's folder structure

config It contains the configuration for some of the components. In this case the Swagger and H2 database endpoints.

constant It contains the common application constants used across other classes.

controller Contains the controller classes that expose the RESTful APIs. Controllers will be explained in more details in section 3.2.4.

converter It contains the classes that convert the WSDtos (Web Service Document Transfer Object) to the domain objects and vice versa.

domain It contains the domain (persistent) classes in which objects can be persisted into physical storage that will persist beyond the life of the JVM (Java Virtual Machine).

ethereum It contains the Java compiled version of the Ethereum smart contract that allows the contract's method to be called from Java classes.

exception Contains the custom exception classes.

repository Contains the repository classes for the corresponding domain classes. Those classes provide the connection between the domain classes and the database. More details will be given in section 3.2.2.

response It contains the custom classes that represent the responses from the RESTful APIs.

security It contains the Spring Security config classes.

service It contains the corresponding classes for each domain class and provides the business logic. It will in more details in section 3.2.3.

util It contains util classes used by other classes.

validator It contains the validator classes used to validate the object that is about to be created and return the corresponding error messages if there any.

wsdto It contains the classes that represent the domain classes but suited to the applications that use them.

And it is worth mentioning that the class *FcServerApplication.java* is a class that contains the main method which starts the application.

Figure 3.2 depicts the layer structure used in the server application. Each of them will be explained in the following sections starting from the bottom.

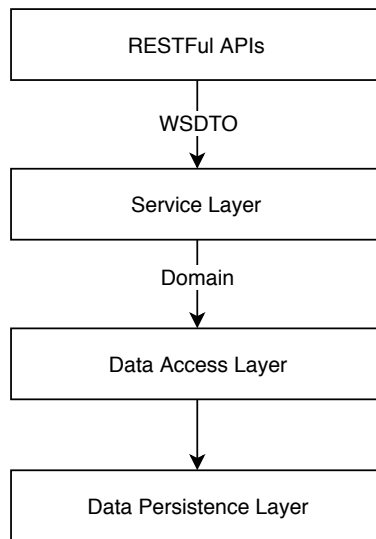


Figure 3.2: Server Layers

3.2.1 Data Persistence Layer

"Persistence, in computer science, is an adjective describing data that outlives the process that created it." [40]

The figure 3.3 shows the underlying database design that visualizes the major entities within the system scope and their inter-relationships.

Short explanation about each table and their fields is given.

- **Producer** - represents the producer entity.
 - **producer_id** - producer's primary key, auto generated value.
 - **ethereum_account** - represents an ethereum account address so the system can associate the transaction with the producer.
 - **licence_number** - the unique licence number assigned to the producer by the government agency.
 - **username** - producer's username that a producer logs in with.

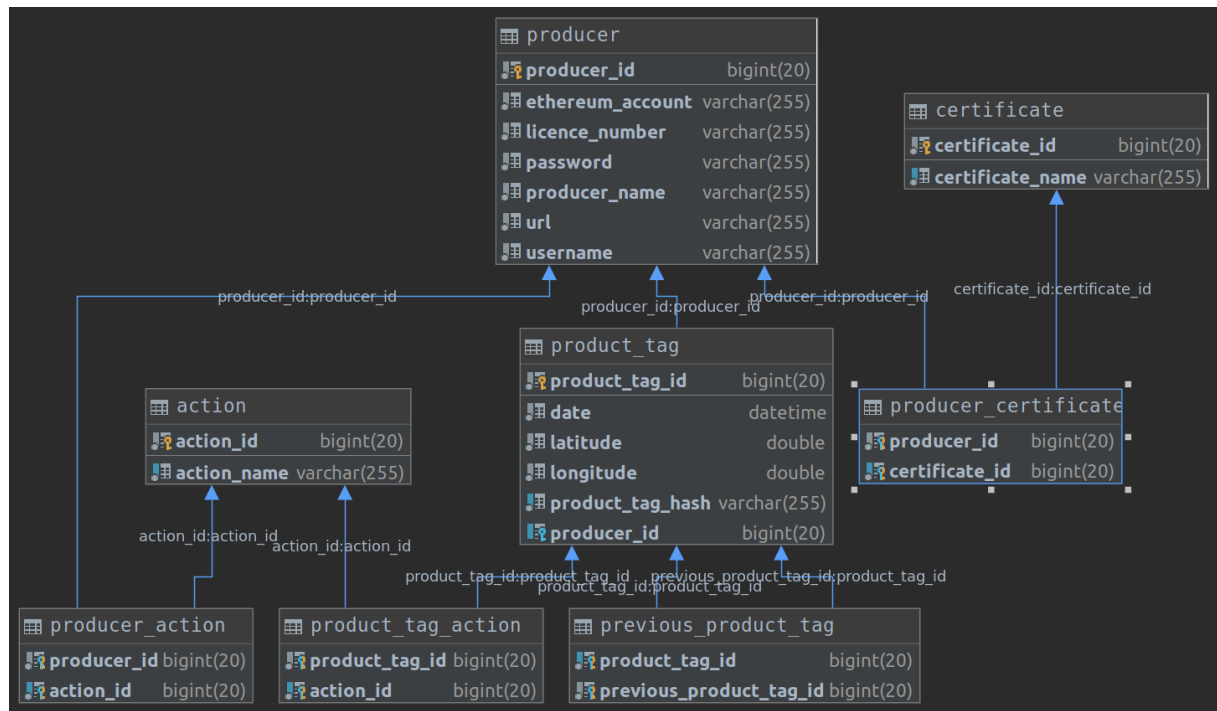


Figure 3.3: ER Diagram

- `password` - the hash value of the producer's password.
- `producer_name`
- `url` - producer's URL address.
- **certificate** - represents the producer's certificates that allow him to perform certain actions to the products.
 - `certificate_id` - certificate's primary key that is autogenerated.
 - `certificate_name`
- **producer_certificate** - aggregate table that keeps the relations between the producers and certificates since one producer can have many certificates.
 - `producer_id` - producer's primary key value.
 - `certificate_id` - certificate's primary key value.
- **action** - represents the actions perform on the products.
 - `action_id` - action's primary key.
 - `action_name`
- **producer_action** - the aggregate table that stores the producer's default actions that are preloaded when a producer sign in.
 - `producer_id` - producer's primary key value.
 - `action_id` - actions's primary key

- **product_tag** - keeps the data about the product tags.
 - **product_tag_id** - product tag's primary key, auto generated.
 - **date** - date when the product tag is created.
 - **latitude** - geo data - latitude value
 - **longitude** - geo data - longitude value
 - **producer_id** - the foreign key of a producer that created this product tag.
 - **product_tag_hash** - a hash value of the product tag. This value is also stored on the smart contract, and also its value is converted to a QR code.
- **previous_product_tag** - the aggregate table that keeps the relations between a product tag and its previous product tags that it originates from. One product tag can have multiple previous product tags.
 - **product_tag_id** - foreign key of the current product tag.
 - **previous_product_tag_id** - foreign key of the previous product tag.

3.2.2 Data Access Layer

In the persistence layer, the Hibernate framework is used in order to map Java classes to database tables and map Java data types to SQL data types. The most interesting example of Hibernate in action is the Product Tag model.

The following code snippet shows the product tag domain that is the most interesting one since it has the references with several other domain classes and with itself as well.

```
@Entity(name = "ProductTag")
@Table(name = "product_tag")
public class ProductTag implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "product_tag_id", nullable = false, updatable = false)
    private Long productTagId;

    @Column(nullable = false, updatable = false)
    private LocalDateTime date;

    @Column(nullable = false, updatable = false)
    private Double longitude;

    @Column(nullable = false, updatable = false)
    private Double latitude;

    @Column(nullable = false, updatable = false, unique = true)
```

```

private String productTagHash;

@ManyToMany(fetch = FetchType.LAZY, cascade = {
    CascadeType.DETACH})
@JoinTable(name = "product_tag_action",
    joinColumns = @JoinColumn(name = "product_tag_id"),
    inverseJoinColumns = @JoinColumn(name = "action_id"))
private Set<Action> productTagActions = new HashSet<>();

@ManyToOne
@JoinColumn(name = "producer_id")
private Producer productTagProducer;

@ManyToMany
@JoinTable(name = "previous_product_tag",
    joinColumns = @JoinColumn(name = "product_tag_id"),
    inverseJoinColumns = @JoinColumn(name = "previous_product_tag_id")
)
private Set<ProductTag> previousProductTags = new HashSet<>();

@ManyToMany
@JoinTable(name = "previous_product_tag",
    joinColumns = @JoinColumn(name = "previous_product_tag_id"),
    inverseJoinColumns = @JoinColumn(name = "product_tag_id")
)
private Set<ProductTag> nextProductTags = new HashSet<>();

```

Important things to note:

- @Entity - specifies that the class is an entity.
- @Id - specifies the primary key of an entity.
- @GeneratedValue - specifies the generation strategy for the primary keys.
- @Column - defines the the field annotated is an attribute of an entity.
- @ManyToMany - defines a many-valued association with many-to-many multiplicity. In this particular case, one product tag can have many actions while one action can have many product tags.
- @JoinTable - specifies a column for joining an entity association or element collection.

The great thing about Hibernate/JPA is that the developers in most of the cases do not have to write manually SQL queries. Everything that needs to be done is that for every model class the corresponding interface which extends the CRUD repository interface. Hibernate will based on the reflection API create the basic CRUD methods. Also, for

more specific methods, only the method in that interface has to be declared without implementing it. The example of the Product Tag repository is shown in the next code snippet:

```
public interface ProductTagRepository extends CrudRepository<ProductTag, Long> {
    Optional<ProductTag> findByProductTagHash(String hash);

    Set<ProductTag> findAllByProductTagProducer(Producer producer);

    @Query("SELECT coalesce(max(pt.productTagId), 0) FROM ProductTag pt")
    Long getMaxId();
}
```

3.2.3 Service Layer

The service layer resides on top of the data access layer. It acts as a transaction boundary and contains both application and infrastructure services. It provides the API for the Web layer which is on top of it. Each model class has its own service class that contains the methods that correspond to the ones from the controller layer 3.2.4. In order to pass data efficiently between controllers and services, the WSDTOs [38] (Web Service Data Transfer Objects) are created. The reason is because in some cases, the requests/responses do not always match our DB entities, so we have better suited objects we want to expose. We also gain in the performance because we transfer the least amount of data through the network. The other benefit is that we can hide the object's database primary keys from the users since it is a bad practice to expose them. Also, since multiple clients use the same backend APIs, we are able to tailor the request/response entities to the specific client separately.

In service layer we have various validators that are used while creating new objects. Every field is validated and if its value is not correct, the corresponding error message is returned to the client.

Here is the place where the Smart contract is called in order to validate if the previous product tag hash is valid or not. The figure 3.4 shows the overview of the previous product tag validation and the dependencies between various validators. The figure 3.5 shows in form of a sequence diagram a flow of creating and validating a new product tag.

3.2.4 Controller Layer

The server application exposes REST API endpoints that are grouped by several controllers where each of them contains several method that consume and produce JSON content.

Each of the controllers is explained in the following section.

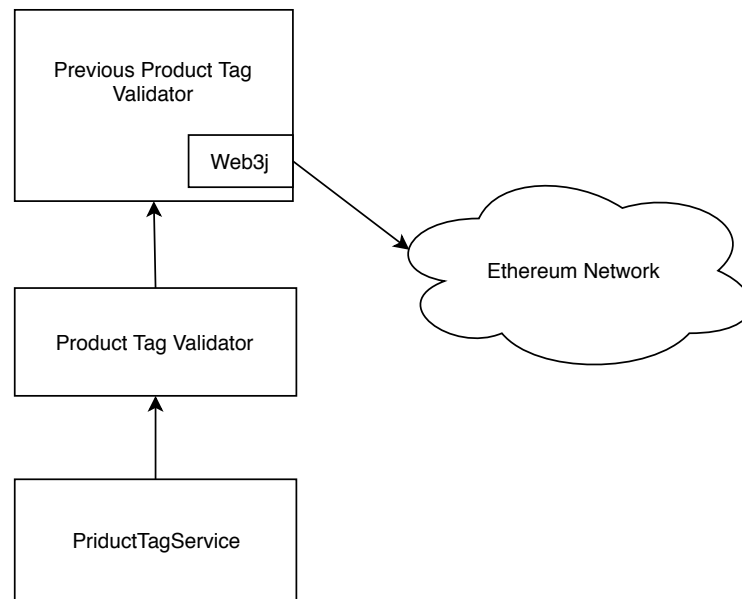


Figure 3.4: Previous Product Tag Validation

ActionController

Action Controller provides the RESTful endpoints for obtaining and creating actions that can be performed on products.

/actions

- Returns all actions
- Method: **GET**
- Protected: **true**

/actions/actionId

- Returns a specific action by its id
- Method: **GET**
- Path Params:
 - Required: **actionId**
- Protected: **true**

/actions

- Create a new action
- Method: POST
- Body Params:
 - Required: `actionName`
- Protected: **true**

/actions/actionId

- Update an action
- Method: PUT
- Path Params:
 - Required: `actionId`
- Body Params:
 - Required: `actionName`
- Protected: **true**

CertificateController

Certificate Controller provides the same types RESTful endpoints as the Action Controller.

/certificates

- Returns all certificates
- Method: GET
- Protected: **true**

/certificates/certificateId

- Returns a specific certificate by its id.
- Method: GET
- Path Params:
 - Required: `certificateId`
- Protected: **true**

/certificates

- Create a new certificate
- Method: POST
- Body Params:
 - Required: certificateName
- Protected: **true**

/certificate/certificateId

- Update a certificate
- Method: PUT
- Path Params:
 - Required: certificateId
- Body Params:
 - Required: certificateName
- Protected: **true**

ProducerController

Producer Controller exposes the APIs for registering, logging in/out, updating producers.

/producers/register

- Register a new producer
- Method: POST
- Body Params:
 - Required: username
 - Required: password
 - Required: producerName
 - Required: licenceNumber
 - Required: ethereumAccount
 - Required: producerWebsite
 - Required: certificateList
 - Required: actionList
- Protected: **false**

/producers/login

- Login producer to obtain a JWT token
- Method: POST
- Body Params:
 - Required: username
 - Required: password
- Protected: **false**

/producers/logout

- Logout from the system
- Method: POST
- Body Params:
 - Required: producerId
- Protected: **true**

/producers/producerId

- Returns a producer data by its id.
- Method: GET
- Path Params:
 - Required: producerId
- Protected: **true**

/producers/producerId

- Update producer's data
- Method: PUT
- Path Params:
 - Required: producerId
- Body Params:
 - password

- producerName
 - licenceNumber
 - ethereumAccount
 - producerWebsite
 - certificateList
 - actionList
- Protected: **true**

ProductTagController

Product Tag Controller exposes the APIs for creating and obtaining product tags.

/productTags/hash/hashValue

- Returns product tag's data by its hash value
- Method: **GET**
- Path Params:
 - Required: hash
- Protected: **true**

/productTags

- Create a new product tag
- Method: **POST**
- Body Params:
 - Required: longitude
 - Required: latitude
 - Required: actionList
 - Required: producerId
 - Required: previousProductTagHashList
- Protected: **true**

/productTags/producer/producerId

- Returns a list of all product tags for the given producer
- Method: GET
- Path Params:
 - Required: producerId
- Protected: **true**

The next code snippet shows the product tag controller class.

```
@RestController
@RequestMapping("/api/v1/productTags")
public class ProductTagController {

    private final ProductTagService productTagService;

    public ProductTagController(ProductTagService productTagService) {
        this.productTagService = productTagService;
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProductTag> getProductTagById(@PathVariable String id) {
        return ResponseEntity.ok().body(productTagService.getProductTagById(Long.valueOf(id)));
    }

    @GetMapping("/hash/{hash}")
    public ResponseEntity<ProductTagWsDTO> getProductTagByHash(@PathVariable String hash) {
        return ResponseEntity.ok().body(productTagService.getProductTagsByHash(hash));
    }

    @PostMapping
    public ResponseEntity<?> addProductTag(@RequestBody CreateProductTagWsDTO productTag) {
        return ResponseEntity.ok().body(productTagService.addProductTag(productTag));
    }

    @GetMapping("/producer/{producer_id}")
    public Set<ProductTagWsDTO> getProductTagsForProducer(@PathVariable String producer_id) {
        return productTagService.getProductTagsForProducer(Long.valueOf(producer_id));
    }
}
```

Things to note here:

- `@RestController` - a convenience annotation that tells that this is an endpoint so the class will be loaded in Spring context, also tells that this is a REST endpoints.
- `@RequestMapping` - annotation for mapping web requests onto methods in request-handling classes.
- `@GetMapping`, `@PostMapping` - annotations for mapping HTTP GET/POST requests onto specific handler methods.
- `@CrossOrigin` - annotation for permitting cross-origin requests - CORS [39] on specific handler classes and/or handler methods.

We leveraged heavily the Spring Boot framework that is build on top of Spring MVC framework. It removes some tedious development steps which allowed us to create a robust, production ready backend application in minimal time.

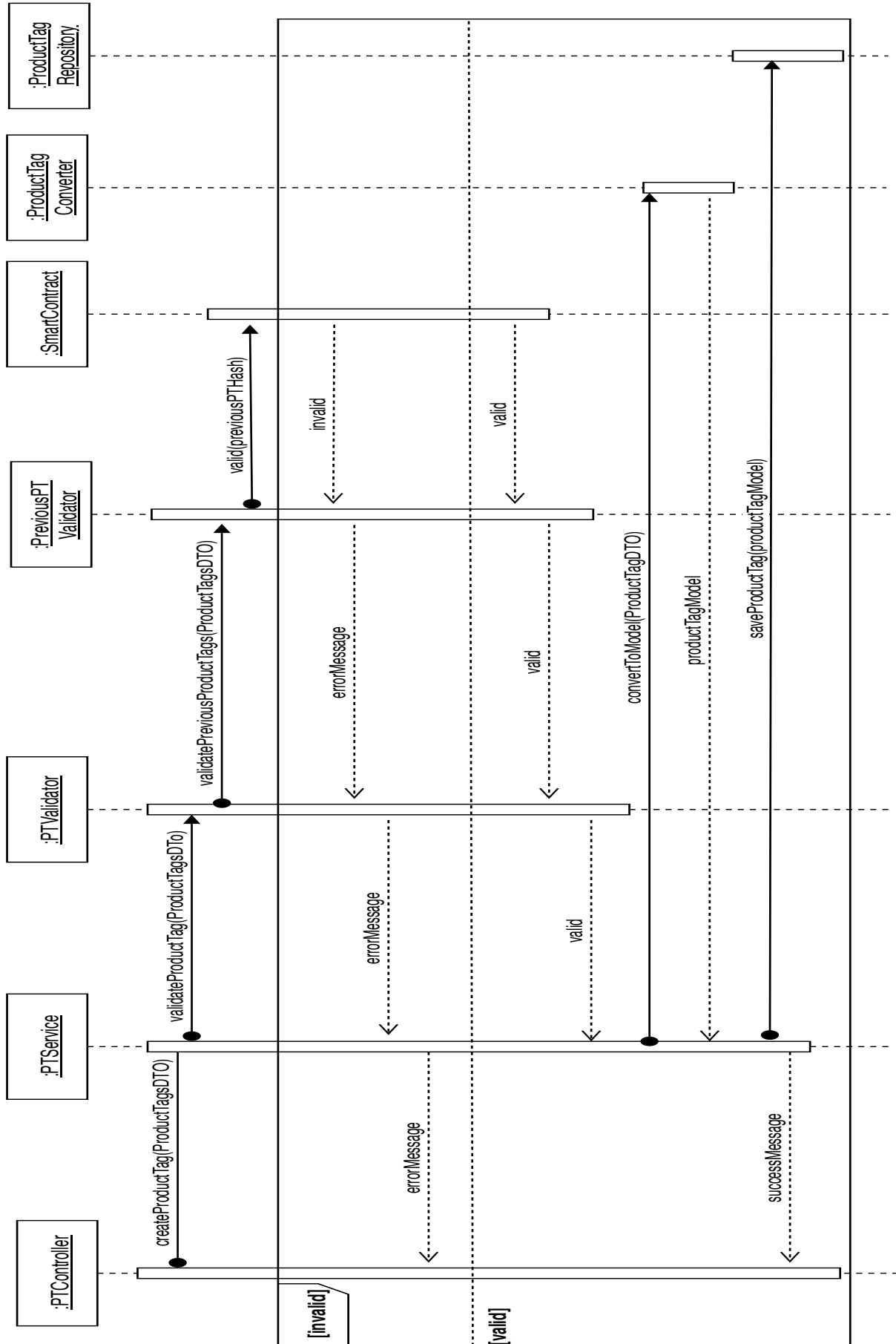


Figure 3.5: Create Product Tag - Sequence Diagram

Chapter 4

Smart Contract

Smart contracts allow us to execute credible, traceable and irreversible transactions without third parties. The main goal of using the smart contract is to provide security that is superior to traditional contract law and to reduce other transaction costs associated with contracting [41].

Smart Contracts usually run on a blockchain network. Our smart contract runs on the Ethereum blockchain and it is written in Solidity programming language [42].

It contains just two methods, one for adding product tag hash value and the second one for checking if the product tag hash value is valid or not. The reason for simplicity is that these two methods provide necessary functionality and that storing more data would incur higher costs for running the smart contract on the Ethereum network.

Smart contract code is given in the next code snippet:

```
pragma solidity ^0.4.17;

contract FoodChain {
    address public manager;
    mapping(string => bool) productTagHashes;

    constructor() public {
        manager = msg.sender;
    }

    function addProductTagHash(string hash) public {
        productTagHashes[hash] = true;
    }

    function isHashValid(string hash) public view returns (bool) {
        return productTagHashes[hash];
    }
}
```

The explanation about the fields and methods is given.

- Fields
 - **manager** - stores an address of the contract creators. Its value is immutable.
 - **productTagHashes** - it is a map data type and it stores true values for the stored hash values.
- Methods
 - **constructor** - a method that is called only once when the smart contract is created. It initializes the manager field with creator's address.
 - **addProductTagHash** - method that receives a product tag hash value and adds it to the **productTagHashes** map with **true** value for the given hash.
 - **isHashValid** - receives a hash value and returns **true** or **false** depending on if the given hash is valid or not.

Once we have the smart contract written, the next step is to deploy it to the Rinkeby test network [43]. The first step is to compile our smart contract. For that purpose, simple script (`compile.js`) is written that compiles our smart contract and its compiled version stores in JSON format. The script is given in the next code sniped:

```
const path = require('path');
const solc = require('solc');
const fs = require('fs-extra');

const buildPath = path.resolve(__dirname, 'build');
fs.removeSync(buildPath);
const contractPath = path.resolve(__dirname, 'contracts', 'FoodChain.sol');
const source = fs.readFileSync(contractPath, 'utf8');
const output = solc.compile(source, 1).contracts;
fs.ensureDirSync(buildPath);

const contract = output[":FoodChain"];
fs.outputJsonSync(
  path.resolve(buildPath, 'FoodChain.json'),
  contract
);
```

You can run the script by executing the command: `node ./compile.js`

Once we have a compiled version of the smart contract, we are able to deploy it. For that purpose, the script `build.js` is created. Its code is given in the next code scipet:

```
const HDWalletProvider = require('truffle-hdwallet-provider');
const Web3 = require('web3');
```

```

const compiledFoodChain = require('./build/FoodChain');
const rinkebyConnect = require('../config');

const provider = new HDWalletProvider(
  rinkebyConnect.walletMnemonic,
  rinkebyConnect.rinkebyAccessNode);

const web3 = new Web3(provider);

const deploy = async () => {
  const accounts = await web3.eth.getAccounts();

  console.log('Attempting to deploy from account: ', accounts[0]);

  const result = await new web3
    .eth
    .Contract(JSON.parse(compiledFoodChain.interface))
    .deploy({data: '0x' + compiledFoodChain.bytecode})
    .send({gas: 1500000, from: accounts[0]});

  console.log('Contract deployed to: ', result.options.address);
};

deploy().catch(error => {
  console.log(error)
});

```

The contract can be deployed by running the command: `node ./deploy.js` If there is no error, the smart contract address should be displayed in the console. To verify that the smart contract is successfully deployed, you can check the **etherscan** website where you can check the address. In our case the contract is deployed on the address: `0x668003aaa66e9f195e5626a3cbba543ffb94fe20` and that can be verified by opening the URL:

<https://rinkeby.etherscan.io/address/0x668003aaa66e9f195e5626a3cbba543ffb94fe20>

Chapter 5

Android Application

Android platform is considered to be one of the most used and open-source platforms for mobile application development across the world. Over the past couple of years, we can observe a huge increase in the use of smartphones all around the world that depend on the android platform. An important aspect of the Android application environment is that Android applications have been written in the Java programming language as the native platform. But now, they can also be written in different new programming languages such as Kotlin from Google. This master project focuses exclusively on Java, as the Kotlin is gaining momentum in the developer's community and it might be a good idea to take a closer look at Kotlin for future extensions. Java is still a dominant language in the industry as well as in academia because it is a safe place to work for a couple of reasons. First, it has years' worth of Android resources on the web or among the development community focused on Java. Second, Java is a language that still has life in it for other platforms - particularly server-side web technologies. We also used Java to develop the back-end system for supply chain tracking platform. As a classic object-oriented programming language proven through industrial applications. It is important to note, there are several interesting details on how an Android application comes to life, from the underlying source code, on Android-based smartphones [16].

"Java source code is compiled at the machine level and is formatted into the bytecode, that represents the complete logic of the written code of the application but not the specific instructions for a specific hardware device. Similar to traditional Java environments used across a wide range of business domains, early versions of Android operated by converting these so-called bytecodes to hardware-specific executable code. This conversion process is conventionally accomplished by using a standard Just-In-Time (JIT) compiler and it takes place every time the application is executed. As soon as the JIT compiler converts the code, it is executed in a virtual machine known as the Dalvik VM. But, the Dalvik VM has been replaced since the emergence of Android version 4.4 (KitKat). The JIT compiler converts bytecodes to executable code directly on the device at runtime each time the application runs; the new approach is called Ahead-of-Time (AOT) compilation. That is a more efficient approach in consuming the resource [16]."

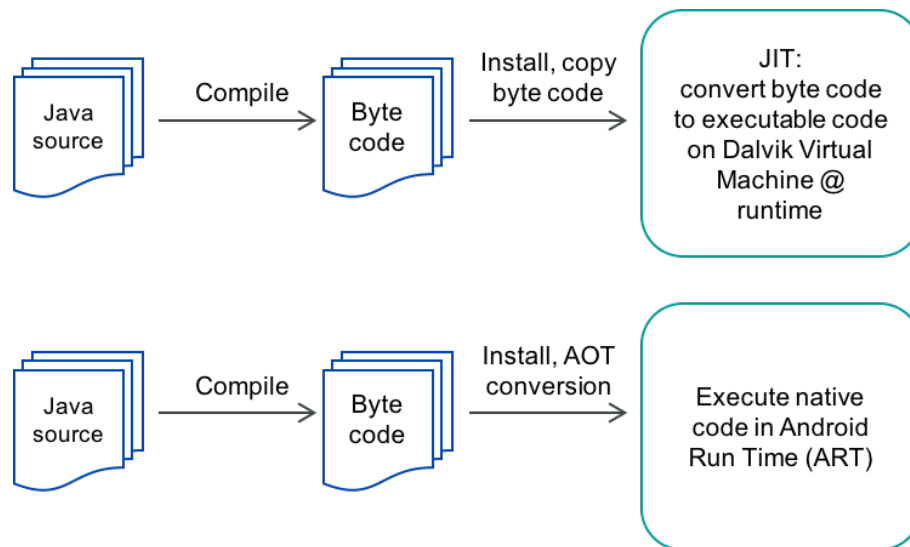


Figure 5.1: Dalvik VM versus Android runtime [16]

5.1 Design and Implementation

It is important to know the fundamentals of android application before designing and implementation of a scalable and extensible application. Apart from the moving pieces of how applications get converted from source code to running code, from a programming perspective, there are some fundamentals to understand when starting with Android development. App components are important parts of an Android application. Each component is an entry point through which the system or a user can enter your app. An Android application consists of one or more of the following main classifications [15].

- **Activities:** Any android application that has a visible user interface is implemented through an activity. An activity can be considered an entry point to interact with the user. It represents a single screen on phone allowing interacting with the user. When an application is selected from the Home screen or application launcher icon, an activity gets started. To understand it easily, we can take an example of an email app that might have one activity that displays a list of new emails, another activity to compose a new email, and another activity for reading emails. Although the activities work with each other to form a nice user experience in the email app, each one is independent of the others. The activity supports the following key interactions [15].
 - It keeps the track of what the user currently performs on the screen to make sure that the system keeps running the process that is hosting the activity.
 - It highly prioritizes keeping the processes around e.g. knowing that previously used processes contain things the user may return to (stopped activities).
 - It also helps the application to handle its process killed so the user can return to activities with their previous state restored.
 - It provides a way for apps to implement user flows between each other, and for the system to coordinate these flows.

- **Services:** A service is a general-purpose entry point for keeping an app running in the background. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. You can use a service for any application that needs to persist for a long time, such as a network monitor or update-checking application. For example, a service might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it to interact with it [15].
- **Content providers:** A content provider handles shared app data that you can save on any persistent storage location. The easiest way to think about content providers is to view them as a database server. A content provider's job is to manage access to persisted data. If the scope of application is very simple, it might not necessarily create a content provider, however, if a complex business application or one which makes data available to multiple activities and/or applications, a content provider is a way of accessing the data [15].
- **Broadcast receivers:** It allows to launch of an Android application to process a specific element of data or respond to an event, such as receiving a text message. A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements [15].

5.2 System Design and Overview

Systems design is the process of defining the architecture of the system, describing modules, what are the main interfaces, and flow data for a system to meet the requirements. It allows us to define the context of the system, and its components, showing the actors that interact with it. The architectural perspective of software highlights the importance of system structure, functions and views on the system, whereas the logical view on system design adds an abstract layer to define the data flows in the system. System design is a visual representation of a system, its parts and their interactions with each other [17].

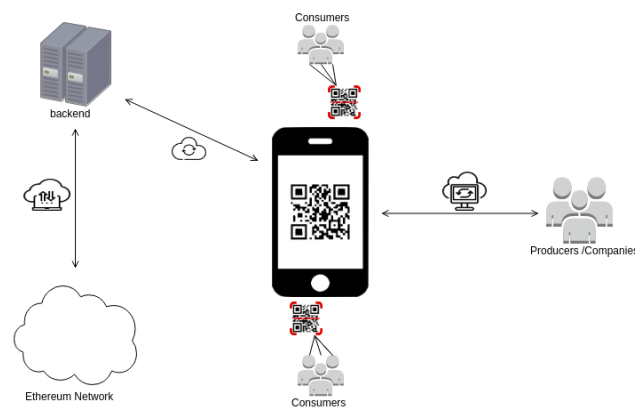


Figure 5.2: System Diagram [17]

It describes the interactions of all participants with the system e.g producers, consumers, farmers, etc. where producers are capable to generate QR code at different locations, whenever a producer makes some changes in the product. It gradually develops a digital chain that provides the consumer with a facility to trace the life-cycle of a product using their smartphones [17].

5.2.1 Use Case Diagram

A use case diagram is a visual representation of the interactions among the components of a system. The purpose of a use case diagram is to show the interaction with the system in an easy way and to show the salient features of the system. It is considered a high-level diagram and without showing internal details [17].

”Following use case diagram breakdowns in four elements.

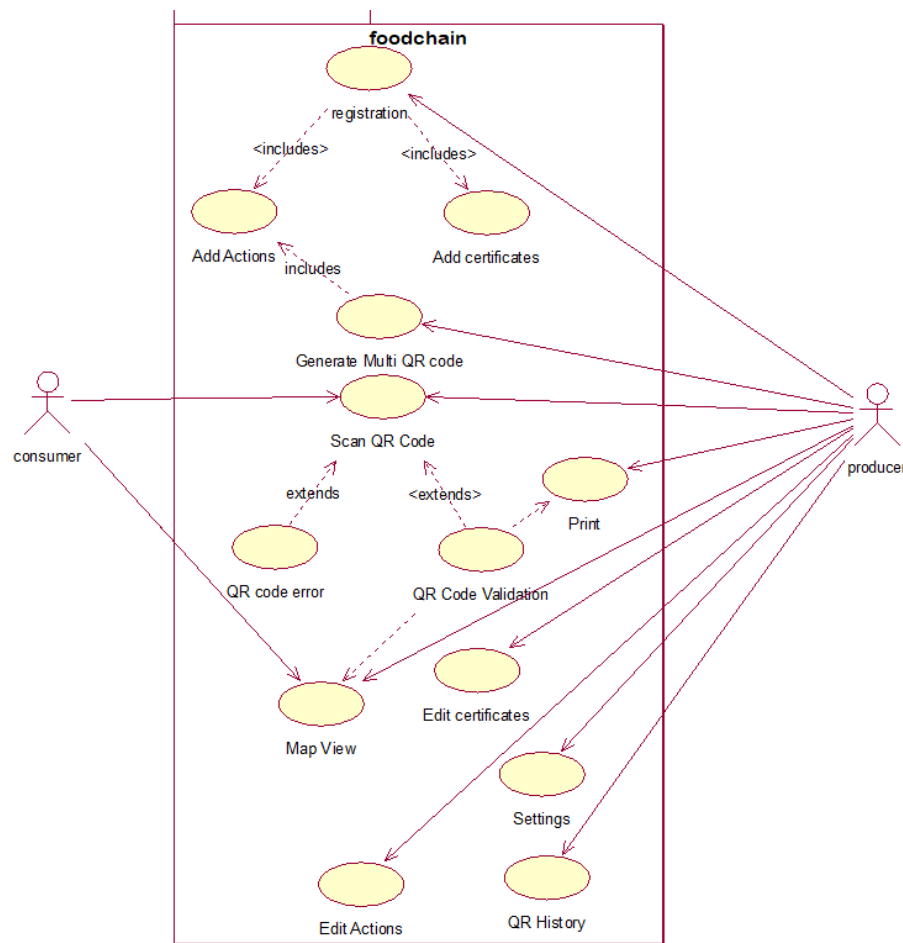


Figure 5.3: Use Case Diagram

1. System:

The system is what we have developed, which is a food chain app for monitoring and tracking the product life cycle. The rectangle shows a food chain system. This rectangle shows the scope and context of the system. Anything inside this rectangle is part of the food chain application.

2. Actors:

The actor is depicted by a stick figure. An actor is someone who is using a food chain application to achieve a goal. In the system, producers and end-users are interacting with the app. Producer interacts with the app to create and scan QR codes and the end-user uses this app to scan the QR code of the product to verify the provenance of the product. These actors are external objects and they are placed outside of our system.

3. Use cases:

Use case is depicted by oval shape and it represents an action that accomplishes some sort of task inside system. Food chain application allows producers to register and then login into system, create QR codes, read QR codes and view them on map, add actions, add certificates, edit actions and edit certificates. And it allows end-users to scan QR codes and see the product lifecycle.

4. Relationships:

- **Association:** Solid arrows show the association relationship which depicts the actions, that a particular actor can perform using food chain application. The consumer is associated to read QR code only because there is no need for end-user to register or login into the system or perform other actions.
- **Include:** This relationship shown by a dotted arrow with the head pointing to use cases that we want to include. It shows we require to add a certificate, add action use case to complete registration and to generate QR code.
- **Extend:** This relationship describes the use case that meets certain conditions and it is depicted by a dotted arrow and its head pointing to use case from which we want to extend. For instance, if the hash query is successful, a map view is shown to producer or end-users which shows the lifecycle of the product and if the hash query is unsuccessful then it shows an error message that product information is not available [17].”

5.2.2 Code Architecture

Code Architecture plays a vital role in the development of a scalable and extensible application framework. It defines the principle on which software is constructed which saves a programmer’s time in the future extensions of the project. It also makes easy to write the code and helps other developers to understand code easily following the defined structure and work on the project to make changes. Foodchain application following MVC Architecture for decoupled major components inside the application [17].

Foodchain application has different layers of code as shown in Figure 5.4.

The main layers of code consist of design using XML, the XML layouts are responsible for the front-end design of User Interface which is accessible to the end-user where an end-user can interact with the application. The design is considered one of the most important parts of any application. Controllers and Adapters are main layers in the application where different functionalities and controlling functionalities exist, e.g event

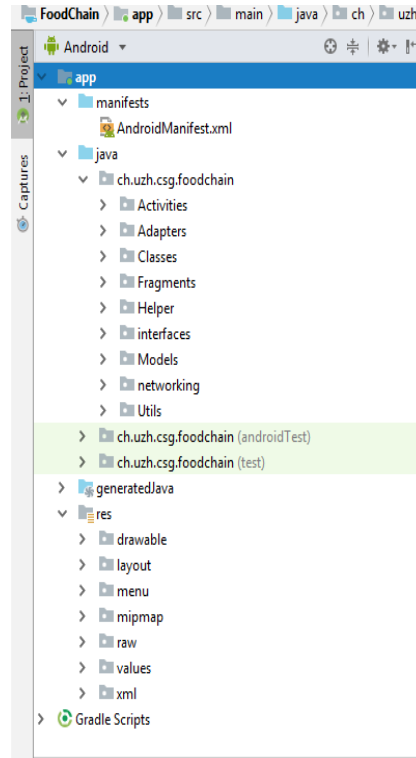


Figure 5.4: Code Architecture

listeners and how to handle request and response of an API. It also includes the business logic for interactions between User Interface, data modeling and data transmission logic over the network through API calls. The main purpose of Adapters and controllers is to communicate with the design components and provide the result back to models or display it on the User Interface with the loaded data from API calls. Adapters are essential blocks in android development. They work as a bridge between the controllers and User Interface. In android application, adapters are mainly used to display data on Listviews, Spinners, Gridviews, Recyclerviews, etc. Android system provides some built-in adapters like ArrayAdapter, ListAdapter, and BaseAdapter to facilitate developers for a simple application, but it's required to design custom adapters following the business and structure of APIs. To develop a clean code, it's imperative to construct all components separately and define the architecture of the system.

5.2.3 RESTful APIs

Restful APIs use HTTP requests to GET, POST, DELETE and UPDATE data on the server. Restful APIs make it easy for developers to access and alter data on the server using API calls. To access data from the server, the Foodchain application is using Retrofit Library which is an open-source type-safe HTTP client for Android and Java. FoodChain application has a remote database and to interact with the database it uses different endpoints. These API calls POST data to the server where it saved and GET data when the application needs to display it for the end-users. The following list of APIs is being used in the application.

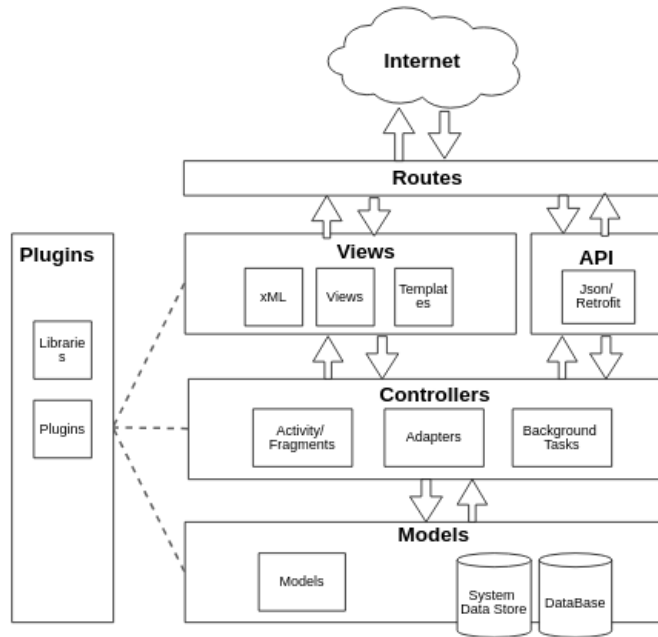


Figure 5.5: Architectural Components

Foodchain API end-point: <https://foodchain.ddns.net/api/v1/>

• Producer

- Register /register is used to register a new producer, for registering a new producer with the system a set of the following information is required, e.g. name, password, certificate, eth account, etc.
- Login /login allows the user to access the system if it has already been registered. A registered user can log in to the application providing the credentials i.e. username and password.
- Get Producer by Id /id an ID is used to get access for the registered user if the id already exists in the system API will get producer's information in the response from the server.
- Get all Producers /producers get all producers in the system, API will get information of all registered producers in response.

• Product Tag

- Create Product Tag ./ to create a product tag or tags a set of information is required such as product hash, previous product hash, actions, current location, and producer.
- Get all Product Tags ./ Will get all the product tags from the server. This information will be used to display on Map showing the list of products to the end-user and it also contains information about the product e.g. a hash of product, the geographic position where is it was generated, the previous hash value on the product, etc.

- Get Product by hash /hash returns in response to the information of the product if the provided hash matches with the back-end system.
- Get Product tag by Id /id gets product information by user id. The response will include the information about the product which matches the id. If the id does not match on the server it returns a response that id does not exist.
- Get Product Tag by Producer /producer/id gets all the products that are generated by a producer with matching id provided through API call.

- **Certificate**

- Create Certificate: ./ it creates a new certificate by providing a certificate name in the API request.
- Get All Certificates:./ gets all certificates registered in the system in the response of an API call.
- Get Certificate by Id:/id it gets the certificate which matches the provided id in API request.
- Update Certificate:/id update a certificate by id. This will update one certificate from the list of certificates that matches the provided id.

- **Action**

- Create Action:./ creates new action. To create a new action, a name needs to be provided through API request.
- Get All Actions:./ gets all the actions created and saved in the system through API call.
- Get Action by Id: /id gets an action by ID. This will get in response only to the action that corresponds to the id. If it does not exist response contains a message that no actions found.
- Update Action: /id to update an action it is required to provide an id for the action which has to be updated. An update will alter only one action with the matching id.

5.3 Implementation

”Food Chain (supply chain monitoring and tracking system) is a potential means of providing value to producers, who tend to provide healthy food for consumers, and consumers, who may select the final product based on food chain details. With such a tracking system, end users can track a product’s life cycle from the first producer via processing stakeholders to the point of sale, where the end-user has finally purchased the product. In this application, such a supply chain tracking system is designed and developed to fulfill the requirements of end-users [17].” The main parts of the Foodchain application consist of consumer and producer mode as it was initially developed in an independent study module [17]. The implementation in Master Project is the modifications and extensions in producer and consumer mode along with adding new and advanced features. These

newly added features such as printing module, multi-QR codes for producer and consumer mode and other modifications are described in the following sections of this report.

”This system is developed in the form of an Android application to enable a fast and easy tracking, which users and producers can use easily, install on their smartphones, and will apply to persist all relevant, necessary, may legally require data and information. Food Chain android-based application consists of following three core components ” [17]:

- Producer Mode
- Consumer Mode
- Back-end

5.4 Producer Mode

As discussed in the implementation part, the Foodchain application has two modes, producer and consumer modes. A producer needs to get registered into the system to use functionalities of Producer Mode. It is required to provide information like name, password, Ethereum account, select actions and certificates during the registration. User validation is implemented on this page with appropriate responses from the backend server. This producer mode offers a wide range of functionalities to producers such as multi-QR code generation and scanning of multi-QR codes, it allows to print the QR code using a label printer and share the QR code using any third-party app installed on the phone, other advanced features to make the map interactive and full tracking history of a product with required attributes, a history of previously generated QR-codes, etc. These features are explained one by one in the following sections.

5.4.1 Authentication

The FoodChain application provides authentication for security reasons which restricts malicious users from accessing other users’ information. The purpose of this functionality is to secure the application from unauthorized access. It is required to provide username and password to log on to the system successfully [17]. FoodChain back-end generates an authentication token when a user login to the system. This authentication token is saved inside the application at the front-end. This token is used when a user generates a QR code, create actions and certificates to securely communicate with the server on the back-end through RESTful APIs. If the token does not exist or if it’s expired, the application disallows the user to set any changes inside the application. It prompts users to re-login to access the supply chain tracking platform.

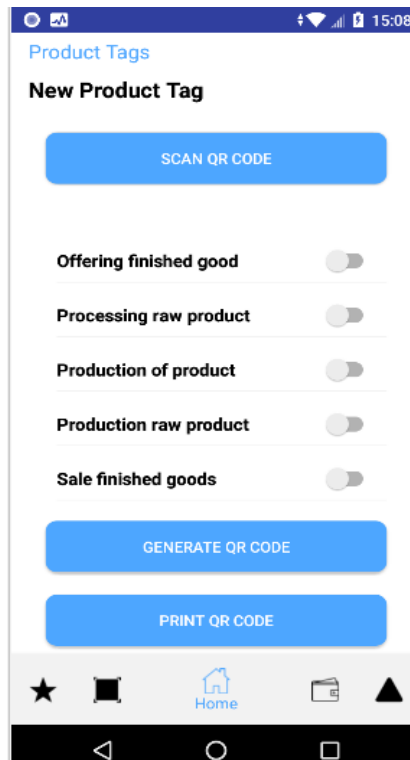


Figure 5.6: Home Page

5.4.2 Home Page

If a producer accesses the application by providing authorized credentials, the main page of the application will be displayed as shown in the figure.

This landing page provides access to all features in the application. The bottom navigation bar contains five options.

1. Product History
2. Generate QR Functionality
3. Scanning Feature
4. Wallet- P2P Module
5. System Settings

The home page also shows the available actions in the system that can be performed on the product while creating a new QR code. A producer can also create new actions, which will be explained in the System Settings on subsequent sections. A producer can also create actions at the time of registration with the system. A producer can select the actions from the available list before generating a QR code. The android application automatically gets the geographical position of the producer before generating a QR code, that represents the location where certain actions were performed on a product. This data is stored in the database and the hash of the data is registered on the Ethereum blockchain through back-end APIs.

5.4.3 Multi QR Generation

A producer can generate a new QR code on clicking on the Generate QR code button as shown in Figure 5.6, the producer has to select a set of actions before generating a new QR code. If a user does not select any action, a message will be displayed on the screen "Please select Actions". If the user clicks on the QR code generate button after selecting the desired actions, the application will check:

- If GPS on phone is turned on. In case if the GPS is off, the application will display an error message to turn the GPS on a device.
- It will show an error message if the authentication token is not valid to access the back-end server, in such a case user needs to re-login.
- Application will also check if it has scanned any previous QR code.

If a producer is generating QR code the first time, the application will generate a pop-up notification "Are you sure it's the first product?" as shown in Figure 5.7.

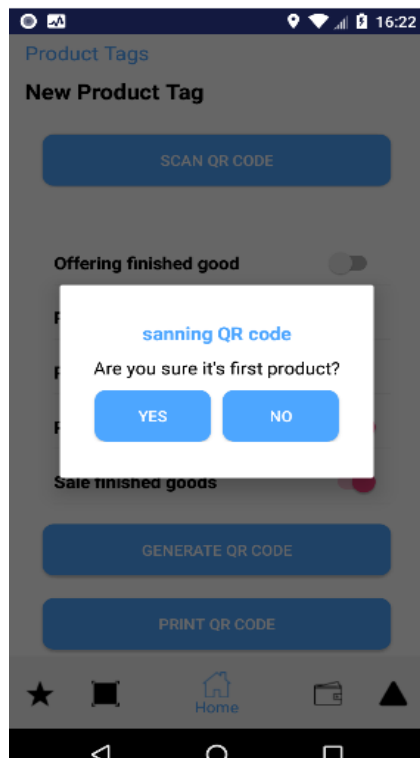


Figure 5.7: First QR Code

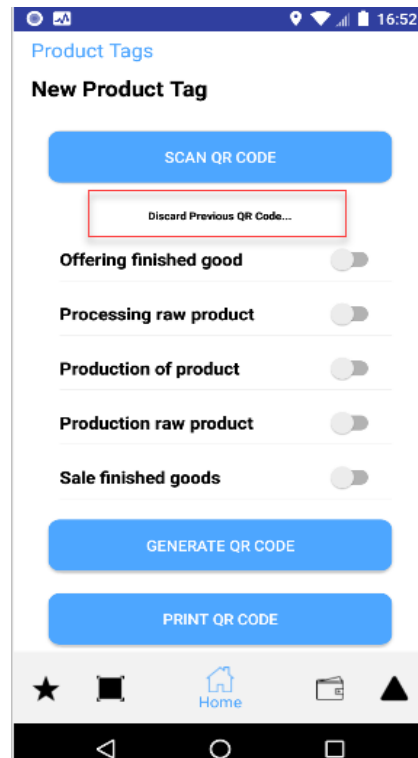


Figure 5.8: Discard Previous QR Code

If a user selects the "YES" application will allow the user to generate QR code if the user selects the "NO" application will ask the user to scan a previous product and a popup will be displayed "Please scan previous product" which take users to the scanning page. On scanning the previous product application will allow the user to generate a new QR code, the newly generated QR will be connected with previous QR code, and it will become a

part of a chain of QR code in the system. The multi-QR feature allows generating codes based on previously scanned QR code(s). If the application has already scanned a QR code, it will show a text on the screen to discard previous code(s) as shown in Figure 5.8. The previous QR-code(s) can represent a single QR-code generated at a location or this QR code can be connected previous chain of products. The purpose of adding this feature is to allow producers to combine QR-code(s) into a single QR-code i.e. a product that is prepared by adding various ingredients from different manufacturers or the same product with contributions from different vendors. It will allow keeping a track of a full history of product with all sources of the product at different places. The user can also generate multiple individual QR codes of a product, In this case, each QR code will be independent of each other.

5.4.4 Multi QR Scanning

We discussed multi-QR code generation in the previous section, where a user can generate multiple QR codes by connecting it with previously generated QR code(s) or chain of QR code(s). The functionality of scanning multi-QR code is developed according to the multi-QR generation. Each time a user scans a QR code application will check by sending a request to the server if the product is valid or not. The server verifies the hash code of product on the blockchain and sends the result back to the android application for further actions.

5.4.5 QR Information Page

If the product is invalid a message will be displayed to the user that the "product is invalid". If the product is valid, which means that the hash code is available on the blockchain and server sends the data for verified hash code to the android application through RESTful API. This data is displayed to the user with basic information as shown in Figure 5.9.

It contains Producer's name, License(s), URL, hash value of QR code. The QR information page also contains two buttons.

Display Map: By clicking on the display map button app will allow the producer to visit a map to see a full history of QR scanned code.

Print QR Code: On clicking the share button a list of options will be available on which user can share QR code or send QR code.

In multi QR scanning, the server will send a list of QR code(s) to an android application. This list contains the information of the current code QR code, which might also contain the information of previous QR code(s) that were created independently or with chain(s) of QR code(s).

This scanning part is also connected with QR code generation in case if the user wants to generate a QR code based on the scanned QR code. To create a chain of QR codes, it's required to generate each QR code based on scanned QR code.

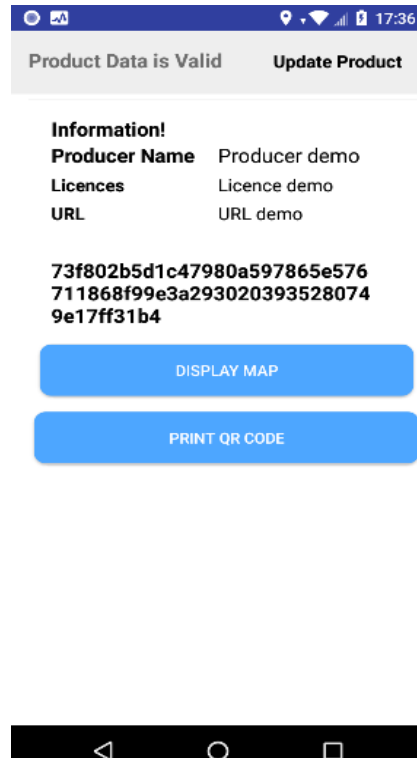


Figure 5.9: Product Data is Valid

5.4.6 Interactive Map View

When a producer clicks the Display Map button on QR Information Page, it shows an interactive map on the screen where user can see a detailed view about the history of products including the information of each product. To this end, markers are displayed at locations on the map where the QR code(s) were generated. The marker positions on the screen are numbered in ascending order, which makes the tracking of objects on the map easy to understand for consumers as shown in Figure 5.10.

This data comes from a server in the form of a nested list of objects or the form of an array of the object according to the created structure of QR code(s) in the back-end. The response from the server is handled recursively and is added to a data structure such as ArrayList. Once the data is loaded into List it is sorted in descending order w.r.t time and date on which product tag is generated.

MapView displays marker in place of products scanned at different places and at different times. On clicking each marker, a pop-up window will open on the map which contains detailed information about the scanned product including the hash of the product, previous hash, time and date and location details including latitude and longitude, actions and certificates.

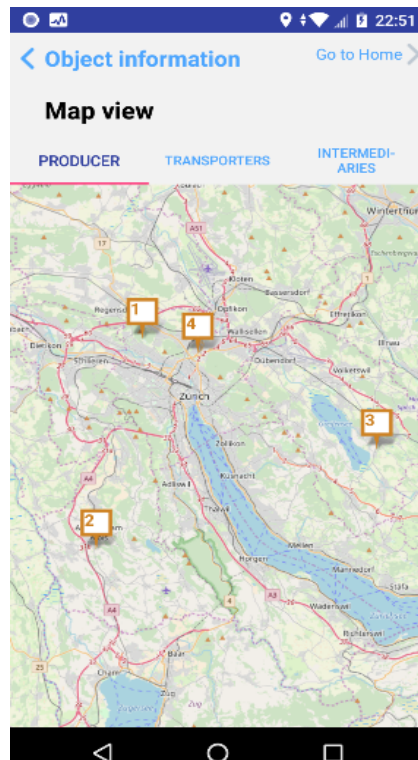


Figure 5.10: Map View

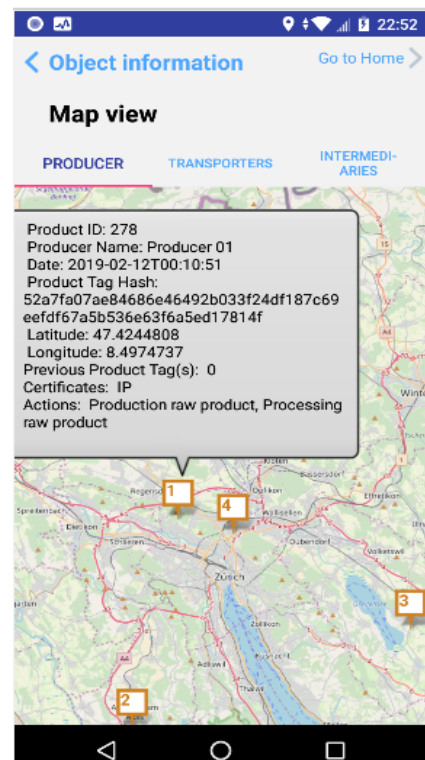


Figure 5.11: Product's Information on map

5.4.7 QR History Page

QR history page contains a list of date and time of QR codes generated in the past. This list is sorted in ascending order so that the latest generated QR code always shows on the top. Each time the producer generates a new QR code of the product, it is displayed on top of the list. The user can click on a list item as shown in Figure 5.12, on the clicking list item QR information page will open.

5.4.8 System Settings

The setting page allows the user to perform different settings and functionalities, these functionalities include[17]:

Create Action A producer can create new action using this functionality, newly included action(s) will be stored in the system.

Create a Certificate A producer can add new certificates using this option.

Logout A producer can log out of the system any time when it is not required to use the application, the current session will expire.

Printer Settings It provides different functionalities for the user to print a QR code through label printer using different modes of communication such as Bluetooth, WiFi, and USB Cable. In basic settings, a user can select or change printer model, port, paper

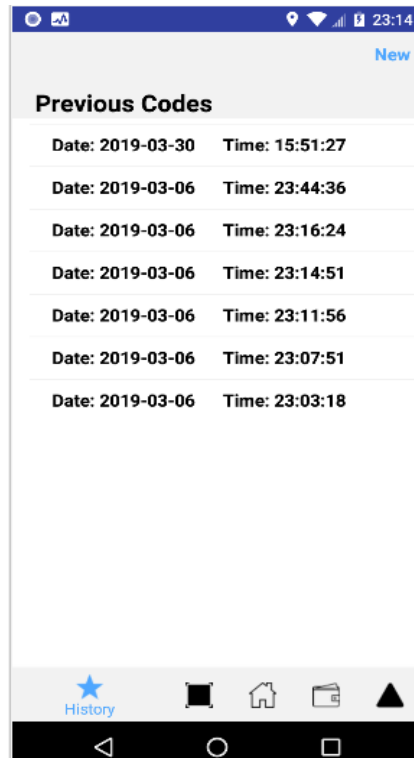


Figure 5.12: History of QR Codes

size, orientation, number of copies, etc. A user can select a file from the device to print, in case the user saved QR on the device or can also use this function for testing the connection with the hardware. Bluetooth settings will allow a user to on/off Bluetooth.

All Actions A user can view all currently added actions.

All Certificates User can view all available certificates in the system.

5.4.9 Multi Language Support

The foodchain application also provides multi-language support for users. It helps the native users to use the application easily. Android system provides the facility to adopt the language of the application using localization strings to the native language of the mobile. In other words, localization provides the functionality to change language strings into the system language. It helps users using the application in their native language. The food chain currently supports five languages i.e. English, French, Italien, Portuguese and German as shown in the following images.

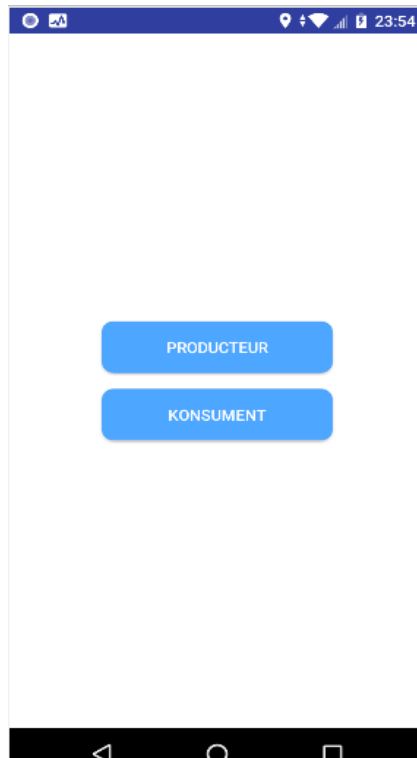


Figure 5.13: French Translation



Figure 5.14: German Translation

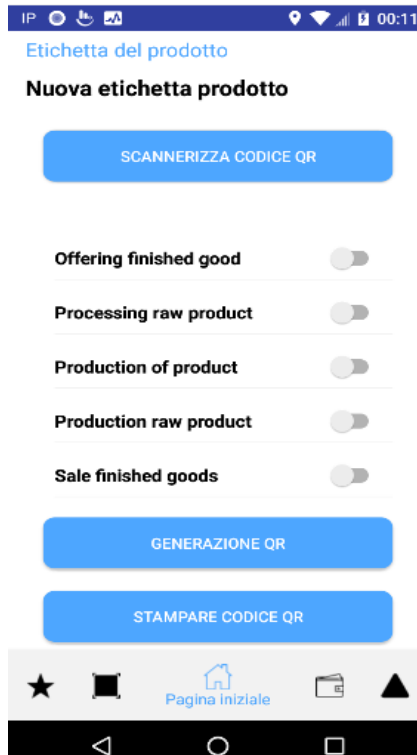


Figure 5.15: Italian Translation



Figure 5.16: Portuguese Translation

5.5 Consumer Mode

"If you click on the food chain application icon a startup screen appears with following two options:

- Producer
- Consumer

The consumer view of the app allows the end-users to scan QR codes for provenance and traceability of the product. Customers can track a product's life cycle from the first producer via processing stakeholders to the shelf, where the end-user finally purchases the product. By scanning QR codes they can see the locations on the map at which the products were produced or actions were performed by the producer [17]."

Consumer mode does not any need registration or credentials to access the application. This mode is provided with limited functionalities only to scan the product and see object tracking on the map. Consumer Mode allows a user to scan a QR code of a product, when a user scans a QR code it will take the user to QR Information Page as shown in Figure 5.9 where user can see the product information. The product information page has a button to display the map. The map shows markers at locations where the QR codes generated. The consumer view cannot access the Producer's functionality such as Generating QR Code, Printing, etc. The consumer view provides the same features on the map but it does not allow the consumer to update the product.

5.6 Printing Module

Printing Module is an important part of the Foodchain application. To this end, we are using Brother's QR-820NWB label printer [18]. It allows producers to print QR codes using three modes of communication i.e. Wifi, Bluetooth and USB cable. It is mandatory to configure the printer before using it in the application. The outline of the general flow of the printing process is shown in Figure 5.17.

A producer can adjust the printer settings from Setting navigation at the bottom as shown in Figure 5.18

Printer option shows general settings upon clicking the Printer button as shown in Figure 5.19. The main functionality of the printing module are [19] :

Select File: Select file option allows the user to select a saved file from device memory and print the file using the connected printer. It allows to test the printer works correctly.

Printer Status: The printer status option shows the currently connected printer is functional. If no printer is connected it will display a message showing "Printer not found".

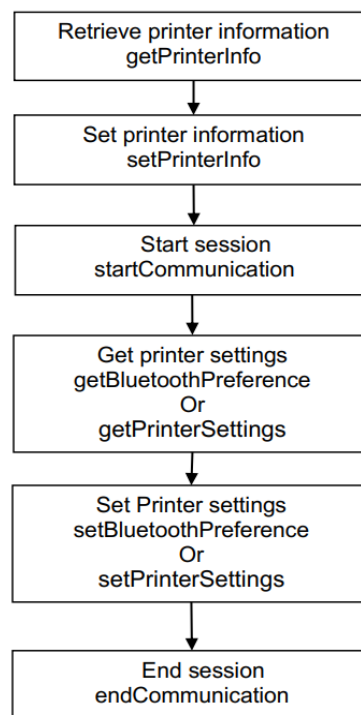


Figure 5.17: Processing Flow of Printer[19]

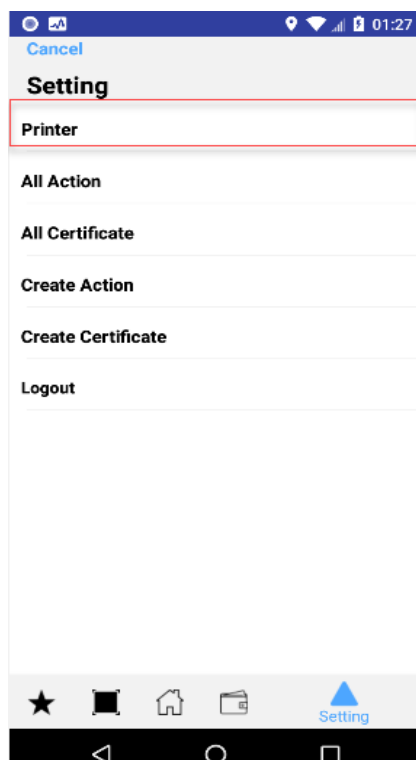


Figure 5.18: Printer

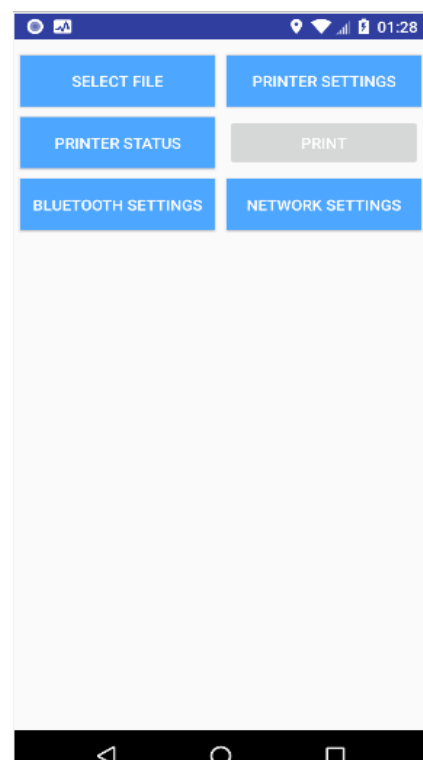


Figure 5.19: Printer Settings

Bluetooth settings: Bluetooth settings allow users to connect with the printer using Bluetooth functionality. This does not require any peripherals to get connected to a printer. Bluetooth settings will ask the user to turn on Bluetooth. If the user allows and

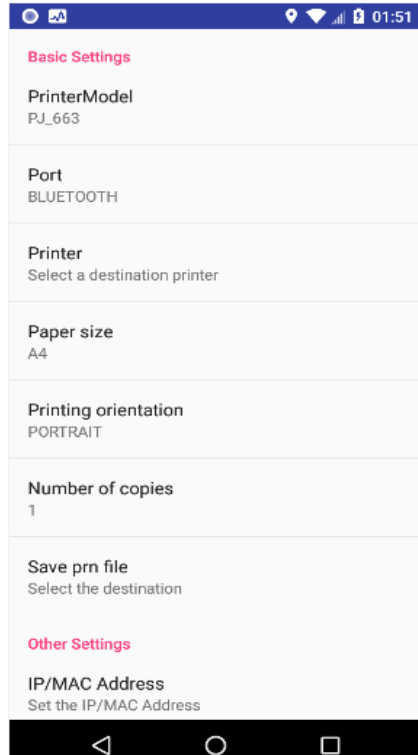


Figure 5.20: Basic Settings

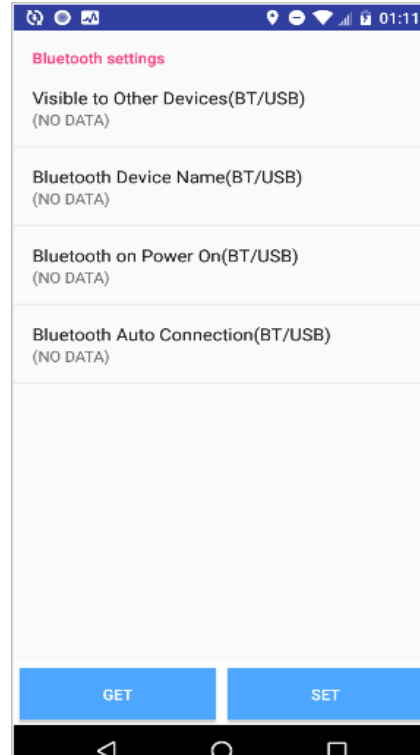


Figure 5.21: Bluetooth Settings

turns on Bluetooth option on the devices, the user can search for connected printers. The Bluetooth setting will show the user all currently visible devices and allow to connect with them. Users can also search for a connected printer by entering the device name. These settings also include auto-connection functionality.

Printer settings: Printer settings contain a list of functionalities. *Printer Model:* The printer model option allows the user to select the printer model from a list of available models. A user can select the printer model of choice from the list.

- *Port:* Port have two options, either connect the printer using Bluetooth port or USB port.
- *Paper size:* Paper size has different paper size options, a user can select the desired paper size(s) from the available options.
- *Printer:* The printer option allows the user to select a printer from a list of connected printers.
- *Printing orientation:* Portrait and Landscape orientations are available for printing.
- *Fit to page:* Fit to page option have the functionality to adjust the printed page. A user can select options like fit a page which will print a full-page print. It also allows the user to fit the print size according to paper size or the user can select a scale from 1 to 10.

- *Number of copies* Number of copies defines that how many copies user wants to print. By default, the count is 1.
- *Save print file*: Save file allows the user to save the file on the device while printing or can overwrite on an existing file.
- *Timeout*: Time out allows the user to set a time interval for printing a paper or the user can also specify a time for waiting for the printer to connect. If the time interval over, it will ask the user to try to reconnect or quit.
- *IP/MAC address*: IP and Mac addresses are two different kinds of methods to connect with a hardware device. Every hardware device which connects to a network has an IP and Mac address, using these addresses we can connect to a device. This functionality allows a user to enter the IP or Mac address of a printer to connect.
- *Alignment setting*: Alignment settings allow a user to set alignment for print. Print on paper will be set according to that alignment setting. In case alignment set to left print will be on the left side of the paper. A user can also specify margins.
- *RJ/TD Settings*: List of settings are specified inside custom settings, a user can rotate the paper.

5.7 P2P Payment System

The P2P payment system allows the user to send and receive ethers using the OySy wallet. OySy wallet was developed at the University of Zurich for the Bazo cryptocurrency, OySy wallet is a Progressive Web App (PWA)-based Mobile Wallet for Bazo blockchain and it was built using Vue.js Javascript framework [20]. We integrate the OySy wallet making the necessary modifications with the Supply Chain Tracking platform to enable crypto payments using Etheruem blockchain.

The Bazo Wallet is integrated with the FoodChain android application using Chrome custom tabs which allows opening web applications inside the android application. The OySy wallet is a progressive web application and it works as a native application using the chrome custom tabs as shown in Figure 5.22.

OySy wallet allows the user to send and receive Ethers. A user can add more than one Ethereum accounts in the wallet as shown in Figure 5.23. It is required to provide the public and private keys of the account. It allows us to add multiple accounts in the wallet, and one account can be select as the primary account for transactions.

For sending payment, the sender's account should have a sufficient balance before initiating a transaction. If the account balance less than the amount to be transferred, the transaction will not take place. It is required to provide receiver's Ethereum account address and amount in Ethers before initiating a transaction, once the user provides the necessary information and click the send transaction button the transaction will be processed using Rinkeby Test Network and the transaction can also be verified on online using <https://rinkeby.etherscan.io/>. OySy wallet provides the facility to visit the

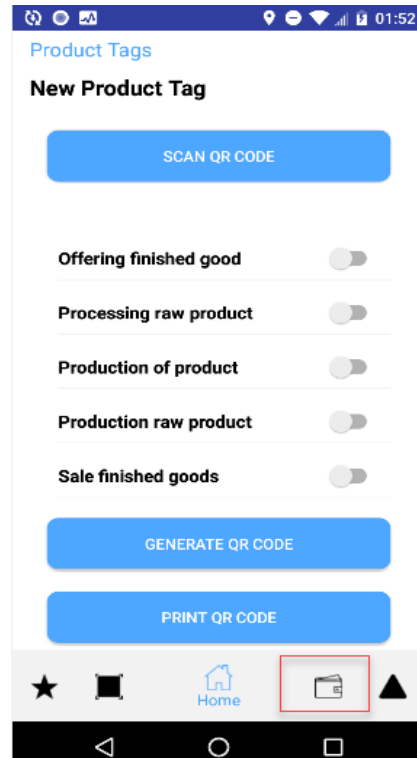


Figure 5.22: P2P Module

history of previous transactions. It also allows seeing the history of transactions of an account. Ethereum test network will work on the request sent by the user and once the transaction mined successfully, the amount will be added to the receiver's account. The mining process may take a few seconds to complete the transaction.

5.7.1 Integration

It was required to integrate the OySy wallet with the Ethereum blockchain. To this end, it was required to study the OySy wallet and understand the underlying frameworks of the wallet.

Transfer Ethers The changes were added in the submitPreparation function in the UserSend.vue component. The modifications are as follows: using a third party APIs i.e. web3.js and Infura's rinkeby http endpoint, web wallet was connected to the test blockchain enabling it to commit valid user transactions on the test network.

Web3.eth.getTransactionCount was used to get the current transaction count of the specified sender's address. The transaction count, as well as other values, were used to form a raw transaction that was signed using the sender's private key.

Inside the web3.eth.getTransactionCount function, another function web3.eth. sendSignedTransaction was used to send the signed transaction to the network and return the transaction hashes.

Request Ethers

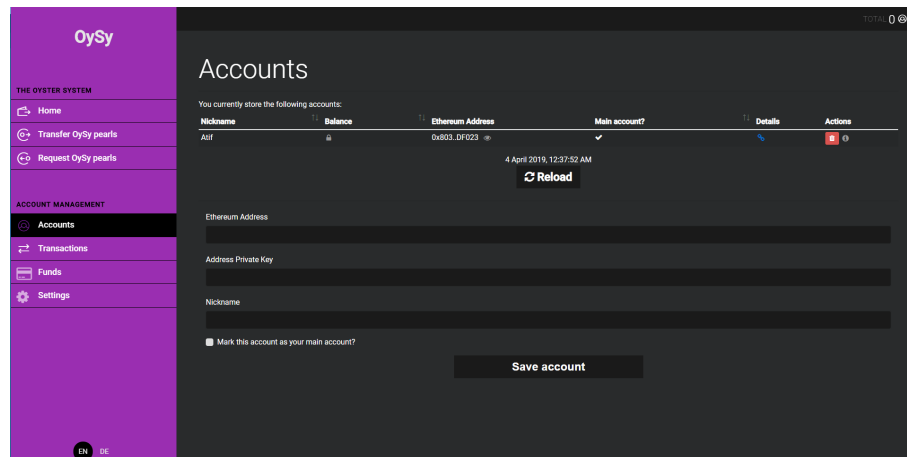


Figure 5.23: Accounts

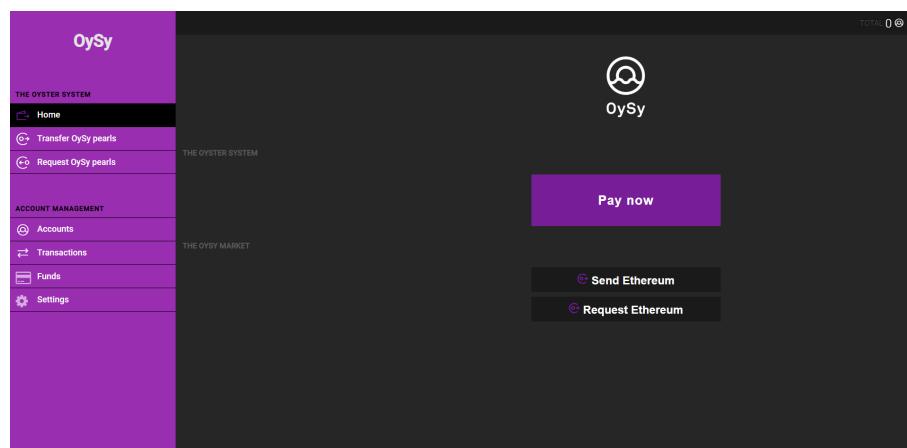


Figure 5.24: P2P Module [20]

The `UserRequest.vue` file was changed to request the Ethers. The dropdown was uncommented in the HTML to enable the feature to select destination accounts. Once the user enters number of Ethers they want to request, selects a destination account and clicks on the QR code button, the `encodedPaymentInformation` function in the `compute` object assigns the information for this request including recipient's address, amount and `posId` to URL which becomes the content of the QR code that is scanned by the sender.

Accounts

In the details section of accounts, the `href` attribute of the anchor was changed to the `rinkeby` network to display the transaction history for the specified account. Besides, an option was provided to add new account information as shown in Figure 5.23.

Transactions

In the transactions view, a `Tx Hash` row was added to show transaction hashes as returned by the transfer function. To this end, a local transaction array is maintained in the `Store.js` configuration file to hold the state of each successful transaction. The code that commits to this array can be reviewed in the `UserSend.vue` component in the last part

of the `web3.eth.sendSignedTransaction` promise. This code is calling `updateTransactions` function exposed by the `Store.js` configuration file.

Chapter 6

Design and Implementation

In this section, we have covered the user interface(UI) enhancement practices that we have implemented in our application and the generalization steps which we performed to make the App generalized for different categories of products. Since explaining each and every component in the App for UI enhancement and generalization process is a lengthy process as we have to explain every layout of our app and every category of our generalized version which is practically very much possible but is not the scope of this report so we focused on some examples to get an idea about how designs are implemented and structured to provide user-friendly Interface. The following sections describe some of the examples of how we implemented the designing.

6.1 UI Enhancement

App's user interface is everything that the user can see and interact with so it plays a crucial role in any App to be successful. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allows building the graphical user interface it also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

6.1.1 Layout

Layouts are responsible for the structure of the user interface. Views and ViewGroups are responsible for building each element in the layout, a view is something the user can see and interact with whereas the ViewGroup is a container that is not visible to the user but determines the structuring of layouts for Views and other ViewGroup objects. We have used Linear Layout and relative Layout to design the layouts in the xml.

1. Linear Layout

The main functionality of Linear layout is to allow us to arrange components horizontally in a single column or vertically in a single row. The direction of the Vertical

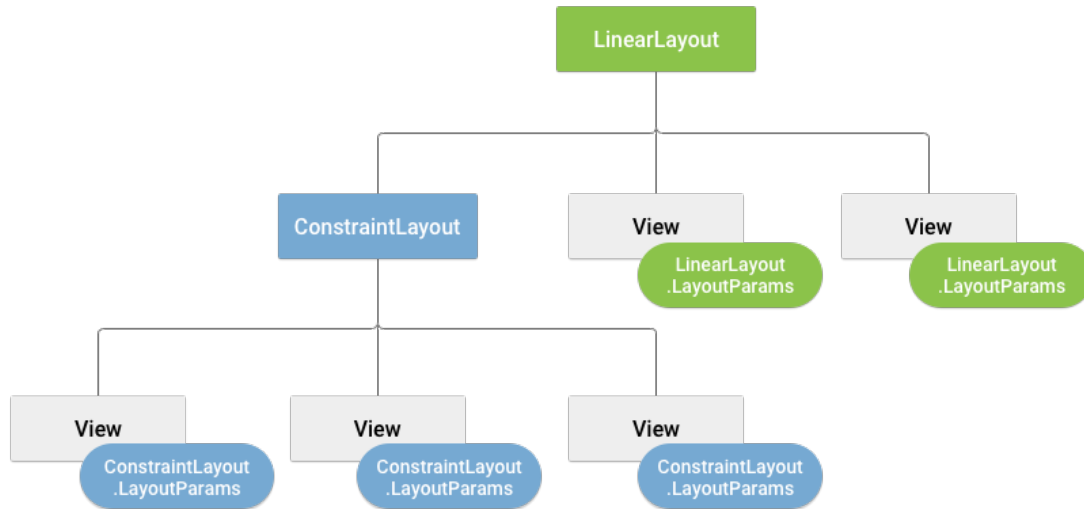


Figure 6.1: Visualization of a view hierarchy with layout parameters associated with each view [72].

or horizontal components can be specified through an attribute **android: orientation**. Linear layouts are very flexible and elementary to use, as can be seen in the figure:6.1, parent view group defines layout parameters for each child view and it also provides a scroll bar if there are any inconsistencies among lengths of the screens and windows. The vertically linear layout has only one item per row. The linear layout has many different attributes that can be used to customize the linear layout according to needs. “All children of a Linear-Layout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding)”[73]. Below are some attributes of the linear layout of Android.

- **Id:** Unique identifier of layout.
- **Orientation:** Attribute to set linear layout orientation as vertical or horizontal.
- **Layout_Weight:** This attribute assigns important value at each component.
- **Gravity:** This attribute shows an object position in the x-y plane such as center, right, top, bottom and left.
- **Weight_sum:** This attribute defines the maximum weighted sum.
- **Divider:** drawable to use as a vertical divider between buttons.

Following are the constructors of Linear Layout.

- **LinearLayout(Context context)**
- **LinearLayout(Context context, AttributeSet attrs)**
- **LinearLayout(Context context, AttributeSet attrs, int styleAttribute)**
- **LinearLayout(Context context, AttributeSet attrs, int styleAttribute, int styleRes)**

Figure 6.2 and Figure 6.3 shows the enhanced view of the app during Registration and Login activity. The Producer Name and UserId field etc., and Register button

each take up only the height they need. This configuration allows the message itself to take up the rest of the activity's height. The following code snippet shows how layout weights might work:

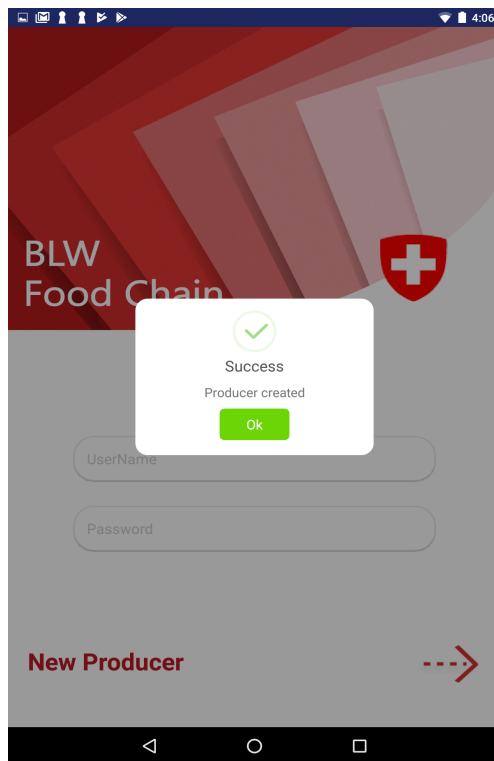


Figure 6.2: Login View

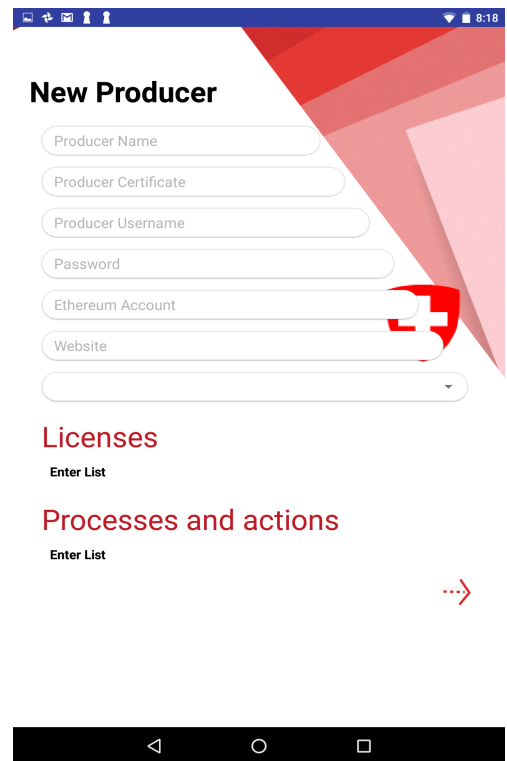


Figure 6.3: Registration View

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:weightSum="5">

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="1" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:text="2" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
```

```

        android:text="3" />

</LinearLayout>

```

It's `android:layout_weight`. Weight can only be used in `LinearLayout`. When the orientation of linear layout was `Vertical`, we used `android:layout_height= "0dp"` and when the orientation was `horizontal`, we used `android:layout_weight width = "0dp"`. The role of orientation is most important in linear layout because on the behalf of orientation all UI depends.

2. Relative Layout

The main functionality of the Relative layout is to display the position of components relative to each other. The position can be specified with respect to consecutive elements or the parent component. It is one of the most flexible layouts available in Android studio, it allows to position elements on the screen and also sets all components at the top left of the layout by default. "It can be defined as a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent `RelativeLayout` area (such as aligned to the bottom, left or center)"[74].

Below are the attributes of linear layout of Android:

- **Id:** Defines 'id' of layout
- **Gravity:** It specifies position of object in x-y plan.
- **IgnoreGravity:** It is added to ignore gravity on a specific component.

Relative Layout has four different constructors as described below:

- `RelativeLayout( Context context)`
- `RelativeLayout( Context context, AttributeSet attrs)`
- `RelativeLayout( Context context, AttributeSet attrs, int defStyleAttr)`
- `RelativeLayout( Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)`

Below are several important methods of relative layout:

- **setGravity():** It sets gravity of the child views as centered, left or right.
- **setHorizontalGravity():** Used to positioned elements horizontally.
- **setVerticalGravity():** Used to positioned elements vertically.
- **requestLayout():** Used to request layout. **setIgnoreGravity():** Used to ignore gravity of any particular element.
- **getGravity():** Used to get the position of elements.
- **getAccessibilityClassName():** Returns class name of object.

Relative layout has following xml attributes:

- (a) **android: layout_above**, It position the bottom edge of the given component above the given component ID.
- (b) **android: layout_alignBaseline**, It position the baseline of the given component above the baseline of given component ID.
- (c) **android: layout_alignBottom**, It aligns at the bottom of given component ID.
- (d) **android: layout_alignEnd**, It aligns at the end of given component ID.
- (e) **android: layout_alignLeft**, It position the component left to the given component ID.
- (f) **android: layout_alignRight**, It position the component right to the given component ID.

Following is the example of code snippet which shows how relative layout may look like in our implementation during registration and its views can be seen in **figure 6.3**.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Fragments.ProducerFragments.ProducerLoginFragment">
```

```
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="fitXY"
        android:src="@drawable/loginbackground" />
```

```
    <EditText
        android:layout_above="@+id/password"
        android:id="@+id/userName"
        android:layout_width="@dimen/_220sdp"
        android:elevation="1dp"
        android:layout_height="wrap_content"
        android:background="@drawable/ed_bg"
        android:hint="@string/user_name"
        android:layout_centerHorizontal="true"
        android:inputType="textEmailAddress"
        android:padding="@dimen/_8sdp"
        android:layout_marginBottom="@dimen/_16sdp"
        android:textColor="@color/black"
        android:textColorHint="@color/gray_btn_bg_color" />
```

```
<EditText
    android:layout_marginBottom="@dimen/_48sdp"
    android:elevation="1dp"
    android:id="@+id/password"
    android:layout_centerHorizontal="true"
    android:layout_width="@dimen/_220sdp"
    android:layout_height="wrap_content"
    android:layout_above="@+id/rel_bottom"
    android:background="@drawable/ed_bg"
    android:hint="@string/password"
    android:inputType="textPassword"
    android:padding="@dimen/_8sdp"
    android:textColor="@color/black"
    android:textColorHint="@color/gray_btn_bg_color" />

<RelativeLayout
    android:layout_marginBottom="@dimen/_16sdp"
    android:id="@+id/rel_bottom"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:padding="@dimen/_12sdp">

    <TextView
        android:id="@+id/tvRegisterProducer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:text="@string/new_producer"
        android:textColor="@color/mat_red"
        android:textSize="@dimen/_15sdp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/btnProducerSignIn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:background="@drawable/nextbtn" />

</RelativeLayout>
```

```
</RelativeLayout>
```

The main functionality of relative layout is it enables us to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent. In this we give padding, padding is the space we need to maintain inside two-component. If we want to give space outside the components then we can use Margin.

6.2 App Generalization

Generalization is an important module of this project. In this module, we have generalized our app for more products like Tea and Coffee, Dairy Products, Fabric and Clothes, Meat Fish, Handmade and Homemade Products such as lather made products and Rugs, and Jewelry. We have integrated all these fields in Category. Each Category e.g., Tea has a specific set of actions and licenses belong to itself. These actions and certificates will be displayed when a producer registers itself in the App. In the following sections, all the changes are explained in detail.

6.2.1 Registration

The major changes are done during the registration phase of the app. After generalization, When a producer registers itself as a valid user of the app he/she has to select a specific category to which the product belongs. We have added a separate Field 'Category' which will display all the available categories offered by the app to the producer. When a producer selects the category then in the following fields named 'Licenses' and 'Processes and Actions' all the relevant data belonged to that specific category will be displayed.

In the Licenses Field, all the certificates related to a specific category is displayed and from these Licenses, the producer has to select the certificate that he/she holds. The same flow follows for the Processes and Actions field, this field will show all the actions related to a specific category and the Producer has to select the relevant actions in the field which he/she is performing on the product. **Fig 6.4 and Fig 6.5** shows the changes made on the registration page. Once the user has filled all his required details, he can register himself as a valid user of the app and can log-in into the app.

Below we have mentioned the API and codebase used for the implementation.

1. When we create a new producer we get all the list of categories from API

```
http://18.219.39.223:8080/api/v1/categories.
```

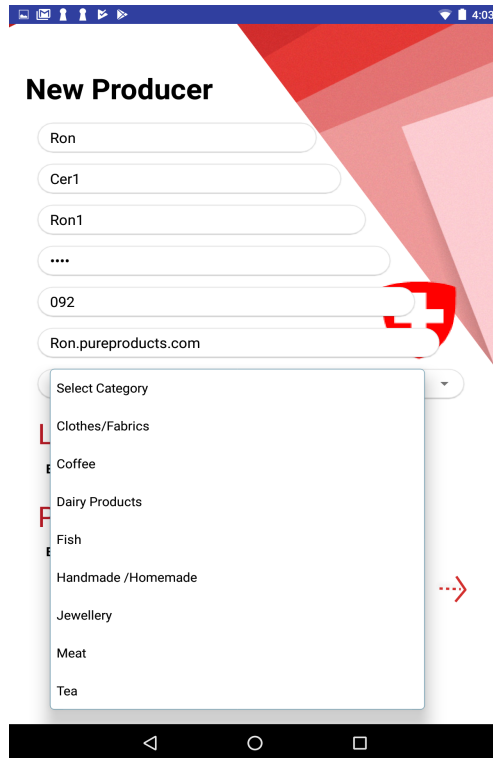


Figure 6.4: List of Categories

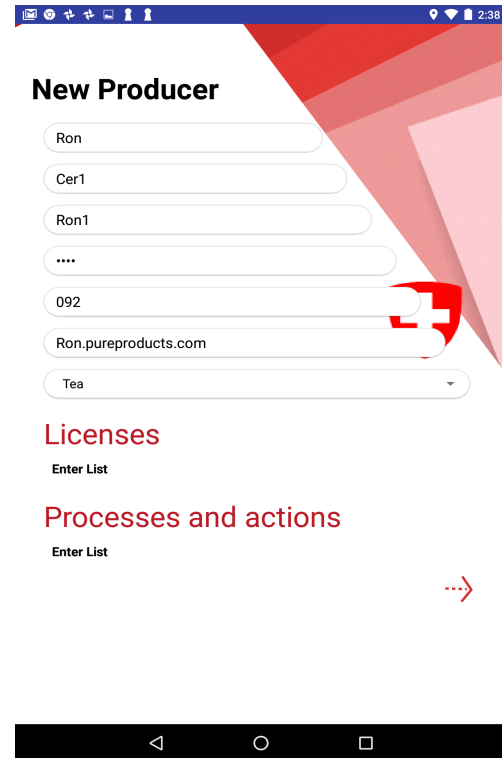


Figure 6.5: Selected Category is Tea

In response to this API we get “categoryId” and “categoryName”. We set the value of these fields in our model class and add that model class in ArrayList which further set in the adapter of spinner where all categories list are showing.

Following is the code of this process:

```
for (int i = 0; i < jsonArr.length(); i++) {
    JSONObject jsonObject = jsonArr.getJSONObject(i);
    CategoriesModel model = new CategoriesModel();
    String certificateId = jsonObject.getString("categoryId");
    String certificateName = jsonObject.getString("categoryName");
    model.setCategoryId(certificateId);
    model.setCategoryName(certificateName);
    categoriesList.add(model);
}
```

```
CategoriesAdapter arr = new CategoriesAdapter(getActivity(),
categoriesList);
spinner_category.setAdapter(arr);
```

User has to select a category to choose license and actions as shown in **figure 6.4 and 6.5**. The category that is selected by the user we sent that id further for selecting license and actions.

2. On behalf of the category list, the License and Actions are coming. In the following points, we have explained the implementation of license and actions.

- We get the license list from API

`http://18.219.39.223:8080/api/v1/certificates/category`

With this API we sent the category id to get a license of that category as shown in **Fig 6.6 and 6.7**.

For example, from previous if we selected a category Tea and its category id is 3 then API looks like:

`http://18.219.39.223:8080/api/v1/certificates/category/3`

In response to this api we get “certificateId” and “certificateName”. We set the value of these fields in our model class (AllCertificateModel) and add that model class in arraylist.

Following is the code of this process:

```
for (int i = 0; i < jsonArr.length(); i++) {
    JSONObject jsonObject = jsonArr.getJSONObject(i);
    AllCertificateModel model = new AllCertificateModel();
    String certificateId = jsonObject.getString("certificateId");
    String certificateName = jsonObject.getString("certificateName");
    model.setCertificateId(certificateId);
    model.setCertificateName(certificateName);
    editLicensesModelArrayList.add(model);
}
editLicenseAdapter.notifyDataSetChanged();
```

- We get the action list from API

`http://18.219.39.223:8080/api/v1/actions/category`

With this API we sent the category id to get Action of that category as shown in **Fig 6.8 and 6.9**.

For example, from previous if we selected a category Coffee and its category id is 7 then API looks like:

`http://18.219.39.223:8080/api/v1/actions/category/7`

In response to this api we get “actionId” and “actionName”. We set the value of these fields in our model class (ProcessActionDataModel) and add that model class in arraylist.

Following is the code of this process:

```
for (int i = 0; i < jsonArr.length(); i++) {
    JSONObject jsonObject = jsonArr.getJSONObject(i);
    ProcessActionDataModel model = new ProcessActionDataModel();
    String actionId = jsonObject.getString("actionId");
```

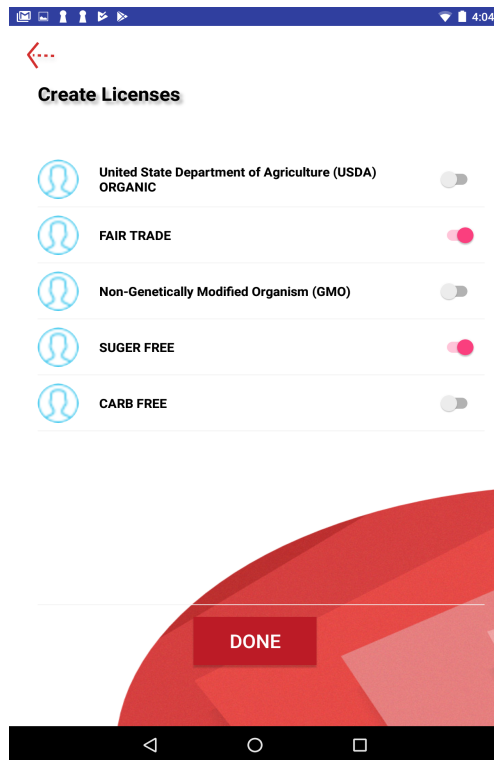


Figure 6.6: Certificates List related to Tea

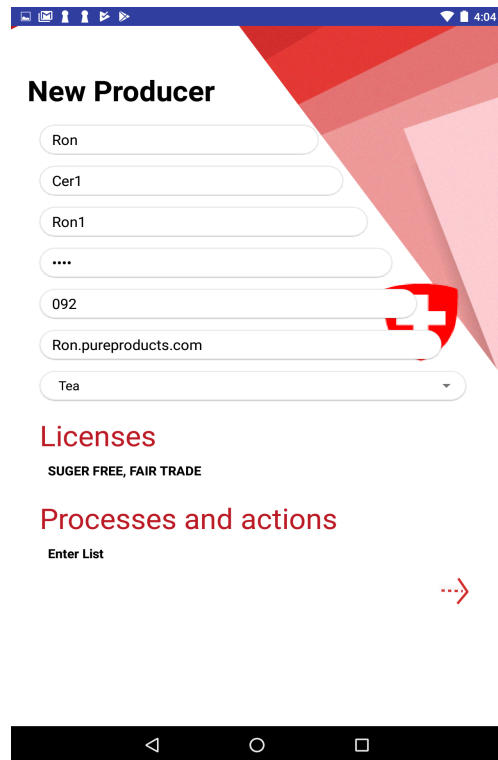


Figure 6.7: Selected Certificates

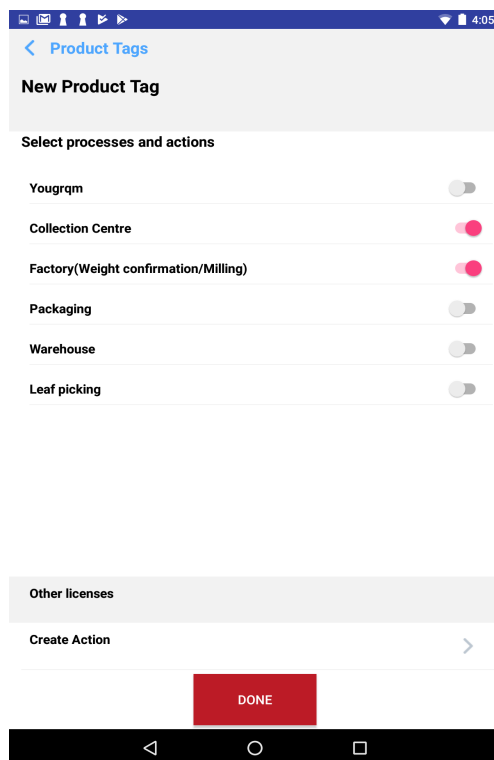


Figure 6.8: Action and Processes List

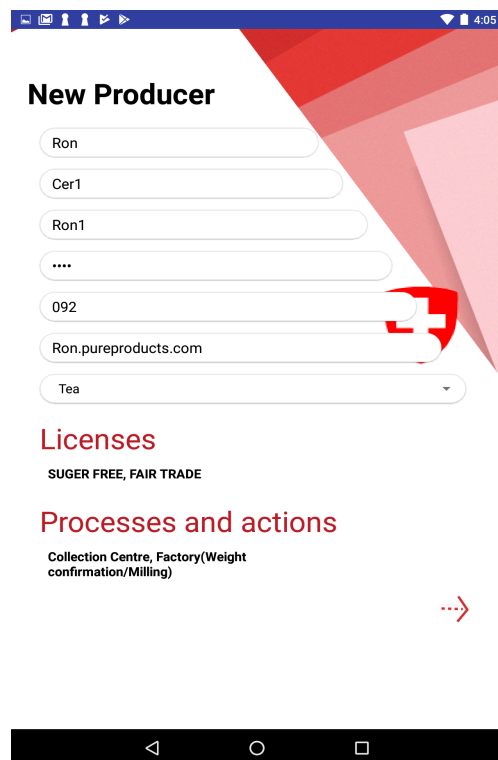


Figure 6.9: Selected Action

```
String actionName = jsonObject.getString("actionName");
```

```

        model.setActionId(actionId);
        model.setActionName(actionName);

        processActionArrayList.add(model);
    }
    processAndActionAdapter.notifyDataSetChanged();

```

6.2.2 Login

After successful registration, the user can log in to the app. In the login page, when the user tries to log in, first of all, we check the validation that “username” and “password” may not be empty. If it is not empty then we hit the following service for login:

`http://foodchain.ddns.net:8443/api/v1/producers/login`

In this API we sent username and password to find that user exists or not and the password is correct or not. If it's fine then on success of this API we get “api_token”, “producer_id”, “categoryId”. We saved these values in shared preference. In shared preference, data will remain till we uninstall the app or clear the app data. We stored the value of category_id for further use that users can see the action and license according to that category.

Following is the code of this process:

```

if (producerLogin.getHttpStatus() == 200) {

    if (alertDialog != null)
        alertDialog.dismiss();

    String message = producerLogin.getMessage();
    String token = producerLogin.getToken();
    String categoryId=producerLogin.getCategoryId();
    Log.d("categoryId:",categoryId);
    String producerId = producerLogin.getResourceId().toString();

    GeneralUtils.putStringValueInEditor(getActivity(),
        "api_token", token).commit();
    GeneralUtils.putStringValueInEditor(getActivity(),
        "producer_id", producerId).commit();
    GeneralUtils.putStringValueInEditor(getActivity(),
        "categoryId", categoryId).commit();

```

```

if (message.equals("producer is successfully logged in")) {
    GeneralUtils.connectFragmentWithBackStack(getActivity(),
        new GenerateQRCodeFragment());
    etUsername.setText("");
    etPassword.setText("");
    GeneralUtils.putBooleanValueInEditor(getActivity(),
        "loggedIn", true).commit();
} else {
    Toast.makeText(getActivity(), "you got some error!",
        Toast.LENGTH_SHORT).show();
}
}

```

The `categoryId` mentioned in the above code snippet is the one which user selects at the time of registration and On the behalf of this id all actions are coming to generate QR code.

6.2.3 Setting

When a user lands on the home page of the app where he/she can generate and share the QR, the app provides further options to see the History of product tag, Scanning along with **Setting**. In the Setting field, the app provides options like 'printer' where users can have access to the printing of the QR and to view All Actions and Certificates along with an option to Generate a new Action/Certificate. Whenever a user generates a new action or certificate we have provided the functionality to link that action/certificate to a specific category after the creation of a new certificate and action user can view that action or certificate in All Actions and All Certificate. This is done to ease the understanding and relevance of the newly generated action/certificate and to maintain the business use of the app. Below we have shown the API used and implementation of the mentioned functionalities which can be seen in **Figure 6.10, 6.11, 6.12, 6.13, 6.14 and 6.15**.

1. To view All Actions following api is used:

`http://18.219.39.223:8080/api/v1/actions`

Below is the used API response:

```

for (int i = 0; i < jsonArr.length(); i++) {
    JSONObject jsonObject = jsonArr.getJSONObject(i);

    ProcessActionDataModel model = new ProcessActionDataModel();
    String actionId = jsonObject.getString("actionId");
    String actionName = jsonObject.getString("actionName");
}

```

```

model.setActionId(actionId);
model.setActionName(actionName);

JSONObject categoryObj = jsonObject.getJSONObject("category");
Category category = new Category();
category.categoryId = categoryObj.getString("categoryId");
category.categoryName = categoryObj.getString("categoryName");
model.setCategory(category);

processActionArrayList.add(model);
}
processAndActionAdapter.notifyDataSetChanged();

```

After the success of this response, we get the list of Actions with their actionId and actionname along with their categoryid and categoryname. This helps to differentiate each action with their corresponding category.

2. To view All License following api is used:

`http://18.219.39.223:8080/api/v1/certificates`

Below is the used API response:

```

for(int i=0;i<jsonArr.length();i++){
    JSONObject jsonObj = jsonArr.getJSONObject(i);

    AllCertificateModel model = new AllCertificateModel();
    String certificateId = jsonObj.getString("certificateId");
    String certificateName = jsonObj.getString("certificateName");

    model.setCertificateId(certificateId);
    model.setCertificateName(certificateName);

    JSONObject categoryObj = jsonObj.getJSONObject("category");
    Category category = new Category();
    category.categoryId = categoryObj.getString("categoryId");
    category.categoryName = categoryObj.getString("categoryName");
    model.setCategory(category);

    editLicensesModelArrayList.add(model);
}
editLicenseAdapter.notifyDataSetChanged();

```

After the success of this response, we get a list of Actions with their certificateId and certificatename along with their categoryid and categoryname. This helps to differentiate each license with their corresponding category.

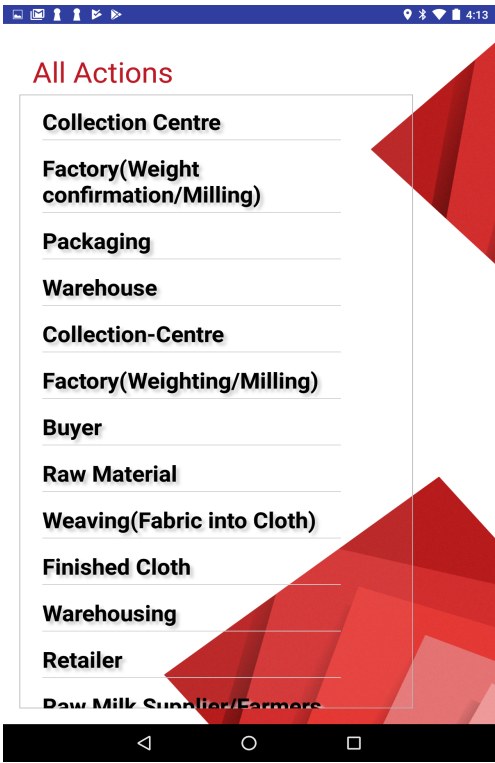


Figure 6.10: All Action List

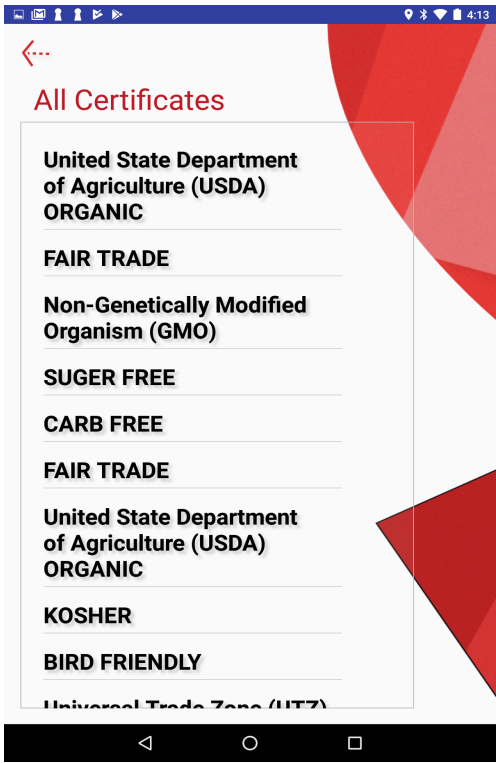


Figure 6.11: All Certificate List

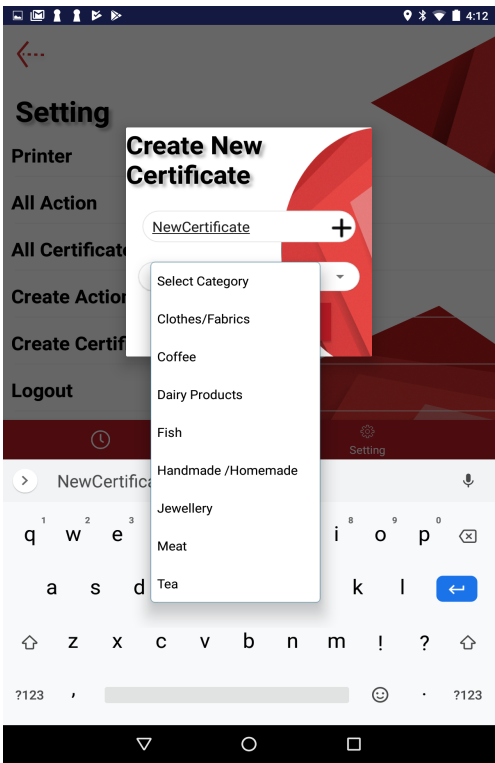


Figure 6.12: New certificate is created

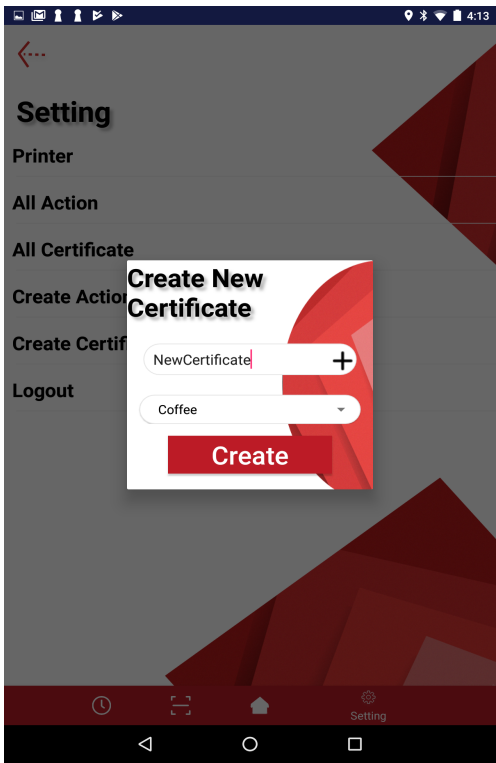


Figure 6.13: New certificate belongs to category Coffee

3. To Add a Certificate following api is used:

`http://18.219.39.223:8080/api/v1/certificates`

With the help of **POST** method we add **certificatename** mentioned by user along with **categoryid** and **categoryname** as described in the below mentioned code snippet.

```
JSONObject js = new JSONObject();
JSONObject categoryObj = new JSONObject();
try {
    js.put("certificateName", strCeriticate);
    categoryObj.put("categoryId", selectedCategoryModel[0].
        getCategoryId());
    categoryObj.put("categoryName", selectedCategoryModel[0].
        getCategoryName());
    js.put("category", categoryObj);
} catch (JSONException e) {
    e.printStackTrace();
}
```

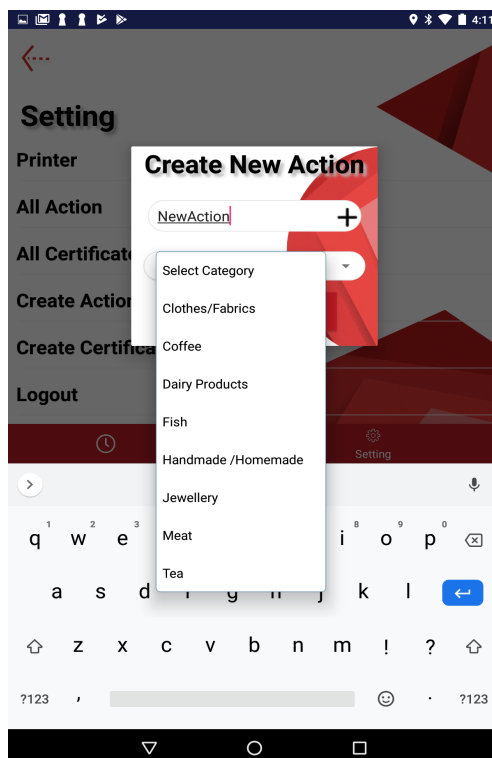


Figure 6.14: New action is created

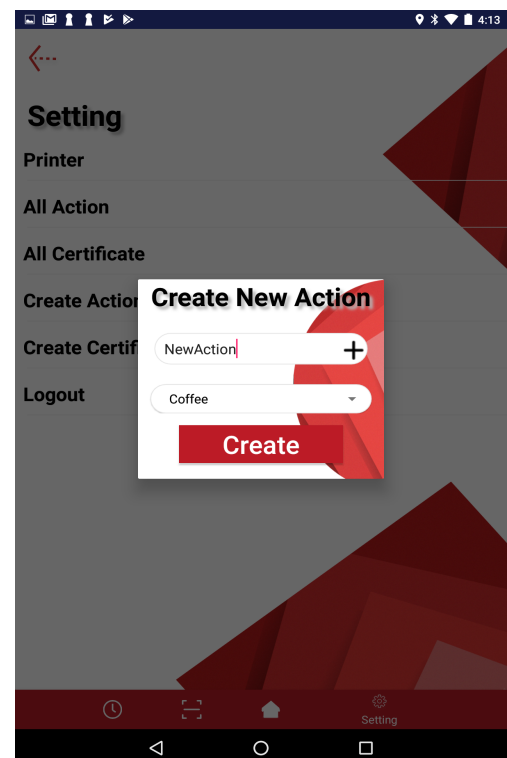


Figure 6.15: New action belongs to category Coffee

The above code can also be used to see how we handled the Exception. An exception is an event, which occurs during the execution of a program, disrupting the normal flow of the program's instructions. There are mainly two types of exceptions:

checked and unchecked. Whenever an error occurs within a method, the method creates an object and hands it off to the runtime system. Errors are considered as unchecked exceptions. We handled it through **try/catch** and **throws**.

4. To Add an Action following API is used:

`http://18.219.39.223:8080/api/v1/actions`

Again with the help of **POST** method we add **actionname** mentioned by user along with **categoryid** and **catergoryname** as described in the below mentioned code snippet.

```
JSONObject js = new JSONObject();
JSONObject categoryObj = new JSONObject();
try {
    js.put("certificateName", strCeriticate);
    categoryObj.put("categoryId", selectedCategoryModel[0].
        getCategoryId());
    categoryObj.put("categoryName", selectedCategoryModel[0].
        getCategoryName());
    js.put("category", categoryObj);
} catch (JSONException e) {
    e.printStackTrace();
}
```

Above mentioned are the major implementations that we have performed in the generalization module of the app.

Chapter 7

Security Analysis

In this chapter, the security of the existing android architecture and all the front and backend processes of the App will be analyzed. In the first section, the basic security model for this project and some general background will be provided. This includes the definition of security used in general and introduction to key security concepts we focused on in this project.

7.1 Security Background

Security is an extensive concept in the context of IT systems, several definitions cover this subject.“ 1)A system state that certificates from the creation and preservation of determinants to protect the system. 2) A system state in which its resource pool is guarded against unauthorized access and unauthorized or accidental change, destruction, or loss. 3) Measures are taken to protect a system.” [65]. According to SANS, Security is the process of the measuring system and information vulnerabilities to safeguard data (business and personal), information (voice conversations, images, videos, multimedia presentations including those yet not discovered). “It is the process of utilizing various forms of technology developed to create, store, use and exchange such information against any unauthorized access, misuse, malfunction, modification, destruction, or improper disclosure, thereby preserving the value, confidentiality, integrity, availability, intended use and its ability to perform theirs permitted critical functions”[66]. Out of different security classes (Network, Internet, Endpoint, Cloud and Application), our key focus is on the application security. Applications are secured at the time of their code creation, such an approach helps to make application creation as much secure as possible and to identify the vulnerabilities. To argue about security and vulnerabilities it is helpful to know some basic concepts, which are introduced in Section 6.1.1.

7.1.1 Security Concepts

For this report , the central notions received to a secure system will be used to reason about the security. The National Academies Press(NAP) Safe Computing in the Informa-

tion Age 1991, defines the concepts of Information security in three main categories i.e., confidentiality, integrity, and availability, which are also generally known as CIA triad. The importance of each of the major categories required for Information security depends intensively on the status quo. For example, the unavailability of the system should be mentioned in the requirements for recovery time. The mentioned definitions are from the NAP specifications [67]:

Confidentiality: controlling who gets to read the information; it is a requirement whose purpose is to keep sensitive information from being disclosed to unauthorized recipients. Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information. The secrets might be important for reasons of national security (nuclear weapons data), law enforcement (the identities of undercover drug agents), competitive advantage (manufacturing costs or bidding plans), or personal privacy (credit histories).

Integrity: assuring that information and programs are changed only in a specified and authorized manner; it is a requirement meant to ensure that information and programs are changed only in a specified and authorized manner. Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information. It may be important to keep data consistent (as in double-entry bookkeeping) or to allow data to be changed only in an approved manner (as in withdrawals from a bank account). It may also be necessary to specify the degree of accuracy of data.

Availability: assuring that authorized users have continued access to information and resources; it is a requirement intended to ensure that systems work promptly and service is not denied to authorized users. From an operational standpoint, this requirement refers to adequate response time and/or guaranteed bandwidth. From a security standpoint, it represents the ability to protect against and recover from a damaging event. The availability of properly functioning computer systems (e.g., for routing long-distance calls or handling airline reservations) is essential to the operation of many large enterprises and sometimes lives (e.g., air traffic control or automated medical systems). Contingency planning is concerned with assessing risks and developing plans for averting or recovering from adverse events that might render a system unavailable.

Furthermore, Authentication and Authorization are also key concepts in the security of the system described in NIST specification [65]. **Authentication** is defined as the verification process of an attribute value claimed by a system entity or system resource. System security depends highly on the authentication of the user's identity, whereas the authentication process can take any attribute type recognized by the system. The whole process involves two steps: 1) Identification, represents the claimed attribute value (e.g., user identifier) to the authentication system. 2) Verification generates authentication information that acts as evidence to prove the binding and that for which it is claimed.

Whereas **Authorization** is an approval granted to a system entity to access system resources. Its semantic depends on the application and implementation. The access control can be specified by any particular mode such as read, write, or execute for one or more

system resources. In the context of the mentioned system security, the subsistence of malicious user is presumed to compromise the system security to have access to information or data which is restricted. This malicious user is known as an attacker and there are many ways through which the security of the system can be breached by the attacker. Attacks can be classified into two categories passive and active. A passive attack learns from the system itself but does not affect system resources. The attempt is to make use of the information being transmitted from the system. **The release of message content** and **Traffic analysis** are some examples of passive attacks. Telephonic conversations, emails message comes under release of content attack where we want to prevent the sensitive information from leaking. Whereas in traffic analysis an attacker can observe the pattern of messages being exchanged between two parties and try to capture the information out of it, the attacker could easily identify the location and identity of the communicating parties from which it is easier to observe the length and frequency of the exchanged messages. On the other hand, **Masquerade** and **Denial of Service (DoS)** are some examples of active attacks. In Masquerade, an entity fakes its identity with some other entity and try to access system resources on fake identity. In the DoS attack, the goal is to disturb or halt the normal function of a service. Distributed DoS (DDoS) attacks are a subcategory of DoS attacks, where the source of the attack is not a single point but many different devices [68].

For mitigation of both active and passive attacks, the system can be secured with several shields of security measures. Although it is nearly impossible to protect the system from all possible attack types. Specifically, attacks against availability are often more frequent and difficult to prevent, the reason being most entities fail to afford extensive DDoS protection. For this project, efforts are made to cover the above-mentioned attacks, whereas for availability we can refer to QR generation vulnerabilities explained in one of the following sections.

7.2 Security Assessment

In this section the security of the whole system will be devised, the architecture used for development, how secure communication is implemented between front and backend processes. We will analyze the architecture used for the development of the application providing In-App security, we will get through the role of APIs to perform a smooth and secure walk-through between our systems front-end and back-end. We used RSA as our encryption technique to encrypt the QR generation on each level of the Supply Chain, starting from our first stakeholder to the last. The user credentials are also encrypted to keep the user data secure and to make sure that it is only the legitimate user who can access the app and perform the relevant actions on the product creation.

7.2.1 Scope

The architecture of the system is already introduced in Figures 2.1 and 2.2. Whereas for the development of the application MVC Architecture is used.

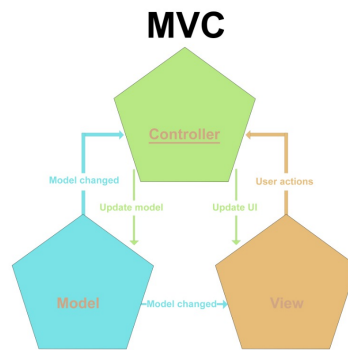


Figure 7.1: MVC Architecture

MVC Architecture:

Model View Controller (MVC) is an architecture pattern for software building and is widely used to implement user interfaces. It has been a popular choice among web apps for its architectural design. MVC architecture design works by splitting out the application logic into three parts which helps in advancing the app’s modularity, makes it easy for collaboration and reuse, enhancing the flexibility of the app and making it more resistant to iterations [69].

The traditional programming approach used the UI coding, business logic, input logic, and applications data domain was written into a single file, which creates certain downfalls in the maintenance of the app, The testability of the app was also sacrificed along with the scalability. MVC approach has improved the creation of the applications by separating the different aspects of the application (input logic, business logic, and UI logic) while providing loose coupling between these elements. The location of each of this logic is specified by the pattern, UI logic belongs in the view, Input logic belongs in the controller and Business logic belongs in the model. This separation helps in building the application efficiently by managing its complexity as it enables the developer to focus on one aspect of implementation at a time. For example, we can focus on the view without depending on the model or controller part [70].

1. The Model

The model defines the data structure of the application i.e., it defines what data the app should contain. “Model objects are the parts of the application that implements the logic for the application’s data domain and retrieve and store model state in the database. For example, a Customer object might retrieve information from a database, operate on it, and then write updated information back to a Customer table in a SQL Server database. In small applications, the model is often a conceptual separation instead of a physical one. For example, if the application only reads a dataset and sends it to the view, the application does not have a physical model layer and associated classes. In that case, the data-set takes on the role of a model object” [70]. The Model always notifies its neighbors whenever the state of this data changes. It notifies the view about recent data changes so that the display can be

changed according to the current state of the data. Sometimes it also notifies the controller if different logic is required to control the updated view [69].

2. The View

The view defines how the app's data should be displayed and deals with all the components that display the application's user interface (UI). This user interface is created typically from the model data. An example would be an edit view of the customer table that displays UI Controls based on the current state of the customer object [70].

3. The Controller

The controller contains logic that updates the model and/or views in response to input from the users of the app. "Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles query-string values and passes these values to the model, which in turn might use these values to query the database" [70].

Following are the advantages of using MVC architecture [70]:

1. Parallel development: Collaborative developers can work in parallel on the model, controller, and views.
2. High Cohesion: MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped.
3. Low coupling: The pattern of the MVC is such that there is very low coupling among different elements i.e., model, controller, and view.
4. Ease of modification: As every element in the model has its separate responsibilities, future development and modification are easier which increases the scalability of the product.

7.2.2 RSA Encryption

We have used the RSA encryption algorithm to encrypt and decrypt the data stored in the QR. This Algorithm is an asymmetric cryptography algorithm which means it works on two different keys i.e. Public Key and Private Key. Following are the keys we used for our implementation:

Public key:

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCoCB9gXo7DCw
uL2JyWZaCdDSPj83RvSIU1cz6f1/ufwcb00l86orXJ0598oNoR7zb/xwj
uUWtXDcLhz0DzA5vLZIbWCI7fHarcTpvbHQW4v/OEWFVfweodn1HVTY9teTEZ
N2ZJI/Hv2SqDnmeJif3nN4EYzSM0aSNM+0cAM75wuwIDAQAB;
```

Private key:

```

MIICdQIBADANBgkqhkiG9w0BAQEFAASCA18wgg
JbAgEAAoGBAKgIH2BejsMLC4vYnJZloJONI+PzdG9IhTVzPp+X+5/Bxs46X
zqitcnTn3yg2hHvNv/HCO5Ra1cNwuHM4PMDm8tkhtYIjt8dqtXOm9sdBbi/
84RYVUXB6h2fUdVNj215MRk3Zkkj8e/ZKo0eZ4mJ/ec3gRjNizRpIOz7Rw
AzvnC7AgMBAAECgYAYgDazApamcfHZDlimxWLKA6R8FH+hyllCEe/LrYm
PHd4zK+e+0QLYzvauYXXqr4rBePAgb9bK9sK30fvq81e/8ALvqhZPWZ+c
OG58PEMC1SXYMh+j4eDZ2LFon+GQugla7EMg6Vi+SEJs9vKd9C64kxtbK3a
WmamA+DJna8HtIQJBANUU2SPgTsvceiBkRE3a/YyAMeTbrvI43L+XIPLEZP
i2XitlUIjwPW0g/FKK7B7dkS+K
LAXPWWInn0PLpxV1bVcCQQDJ4F58Z5YKd41KwXu7X019KeILG47XR+y2tTR
EOzYN+qbAMDI+YMGBINsqAtiaCxx5Qk2S4VsS+2TNyZVj9tU9AkATXw/WiR/
gvKyFoVfsVApU6Zqf      mbEv6Cd5a33dVClxyvKIs28848pRz3vg7SxGKhJ2
5fLWnGE4UhdW5JXCZ+FAkBP9aS3FbhiMPKaP4Sos6ZvwvUbfTDYqE5kF+W5
2NIS02Ns4ComduBuKvtj6nismxTEjzR2pvgRzqOgrWWUkKHpAkBcgCt8SH
ZGFE2GiUfUNAw/4xz29lZTLuv8t0GDXYsFJ74FzfAavXpQZzrLOWLl9jhE
Fqe64JM/dMHHDNKoYLA;

```

QR generation

The above mentioned static public and private keys are used for a digital signature during the QR generation, the public key verifier is used to verify the data if the public key is null/ empty, the encryption will not work and therefore we have to regenerate the public key.

Encryption process:

```

File saveEncryPath, saveDecryPath;
FileEncryptionManager mFileEncryptionManager;

private void initEncryptionData(){
    mFileEncryptionManager = FileEncryptionManager.getInstance();
    File sdcard = Environment.getExternalStorageDirectory();
    saveEncryPath = new File(sdcard.getPath()+"/diapers_encry.txt");
    saveDecryPath = new File(sdcard.getPath()+"/diapers_decry.txt");

    try {
        mFileEncryptionManager.setRSAKey(publicKey,privateKey,true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

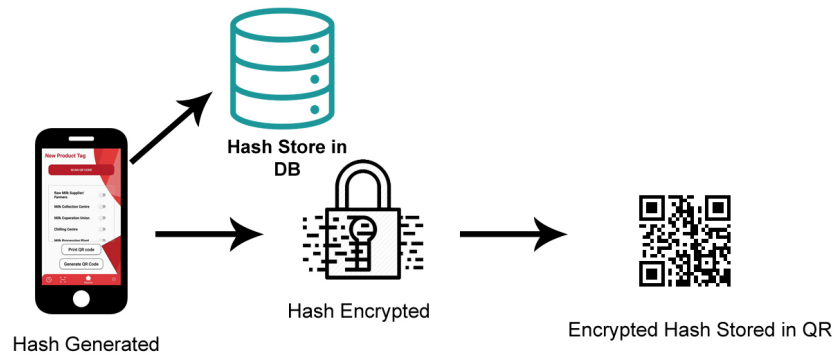


Figure 7.2: Encryption Process



Figure 7.3: Decryption Process

Producer performs certain actions on the product and generates the QR when he clicks the button, a hash value is generated. The generated hash is stored in the database and the copy is stored in the QR with an encrypted hash. We encrypt the data using `encryptPublicKey` method, in the parameter we provide data in byte array along with our public key and we saved the data in the file **diapers_encry.txt**. The reason for doing this is to prevent the access of data stored in the QR via any random scanner.

Decryption process:

```
private void initEncryptionData(){
    mFileEncryptionManager = FileEncryptionManager.getInstance();
    File sdcard = Environment.getExternalStorageDirectory();
    saveEncryPath = new File(sdcard.getPath()+"/diapers_encry.txt");
    saveDecryPath = new File(sdcard.getPath()+"/diapers_decry.txt");

    try {
        mFileEncryptionManager.generateKey();
        mFileEncryptionManager.setRSAKey(GenerateQRCodeFragment
            .publicKey,GenerateQRCodeFragment.privateKey,true);
    }
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Decryption is done during the scanning of QR through our application. At scanning Encrypted value will be fetched from the QR, the DecryptPublicKey method will decrypt the data bypassing the data(in byte array) and public key as our parameters. This will help us to get the Original Hash value and we fire this hash as a parameter via our APIs which gets the data for us from the database.

7.2.3 Retrofit

Retrofit is a REST Client for Java and Android. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST-based web-service. With the help of Retrofit, we can configure which converter is used for the data serialization. Typically for JSON, we use GSON, but we can add custom converters to process XML or other protocols. Retrofit uses the OkHttp library for HTTP requests [12].

To work with retrofit in our project we used the following three classes:

1. We use a Model class which is used as a JSON model
2. Interfaces that define the possible HTTP operations
3. Retrofit.Builder class - Instance which uses the interface and the Builder API to allow defining the URL endpoint for the HTTP operations.

Every method of an interface represents one possible API call. It must have an HTTP annotation (GET, POST, etc.) to specify the request type and the relative URL.

Retrofit authentication

Retrofit supports API calls that need authentication. Authentication can be done by using a username and a password (Http Basic authentication) or an API token [12].

There are two ways through which we can handle the authentication. The first is to manipulate the header for the request with the help of annotations. In this, if any user of the app wants to request his user details which require authentication, this could be done by adding a new parameter to our API definition. With the help of the @Header("Authorization") annotation, we tell Retrofit to add the Authorization field to the request header with the value we provide for credentials.



Figure 7.4: Requests-Response Handling [13]

Another possibility is to use an OkHttpClient interceptor for that. The first method is only useful when a user requests his details. Whereas when we have multiple calls that require authentication we use an interceptor. It is used to modify each request before its implementation and alters the request header. The advantage is, that it is not required to add `@Header("Authorization")` to each API method definition. To add an interceptor, we have to use the `okhttp3.OkHttpClient.Builder.addInterceptor(Interceptor)` method on the OkHttpClient Builder.

Authorization Token

The authorization token is uniquely generated randomly for every login. Each user of the application gets its authorization token during login and this token is responsible to refer the user as a valid user of the application. It is also used to allow the user to have access to specific service in the application with a reference to his generated token. It is stored in the shared preferences of the device, which is a local database used to store the authorization token. The token is overridden by the new one, each time the user login the app. The main reason for mentioning this here is that we handle this token generation and implementation in our app through Retrofit. Retrofit also helps us to send the request and get the responses from our backend server efficiently and securely.

Chapter 8

Scalability

In this chapter, various practices are performed to make the application scalable. Scalability in an android application typically maintains a good user experience over thousands of APP users. In section 3.1 we are going to check the compatibility of our system concerning different devices and versions. In section 3.2, analysis for different screen sizes is performed to have similar screen views on different screen sizes. Whereas in section 3.3, two different types of test Load Test and Performance Test are practiced to check the scalability of the application in terms of the number of users involved.

8.1 Device Compatibility

The design and architecture of android are such that it could be used for many different devices, starting from smartphones to tablets to smart televisions. As a developer, it helps you to approach a huge potential audience for your app with the range of devices that one could offer, it is also a key factor for any successful app to provide a scalable range. It must be tolerant to feature variability and provide a flexible user interface that adapts to different screen configurations. In order to expedite our efforts towards this goal, Android itself provides a **Dynamic App Framework (DAF)**, DAF is a framework which gives the liberty to a developer to provide configuration-specific app resources in static files (e.g., different XML layouts for different screen sizes) this helps to externalize the app resources and makes it easier to access them using resource IDs. It is always helpful to provide alternative resources for specific device configurations, we can do this by grouping them into specific resource directory, this helps Android to use appropriate resources during runtime based on the current configuration(such as providing different UI layout depending on the screen size or different strings depending on the language setting). For some discretion to the app design and further additional app resources, a single application package (APK) can be proclaimed that provides an optimized user experience on the different number of devices [71].

Application Programming interfaces (APIs) are the best means through which our application can leverage different features an Android platform can offer. Some of the features

are hardware-based (e.g., compass sensor), software-based (e.g., app widgets) and some of them are dependent on the platform version. It is meaningful to say that every device can't support every feature, so we need to control the availability of our app to devices based on our app's required features. We can scabies availability of our app with respect to device features through Android feature IDs, these feature IDs are used for any hardware or software feature that may not be available on all devices. Some examples of features IDs are `FEATURE_SENSOR_COMPASS` (used for compass sensors) and `FEATURE_APP_WIDGET` (used for app widgets). We can also prevent users from installing our app when the user device does not allow for given features, it can be done it declaring the feature with `<uses-feature>` element in our app's manifest file [71]. For example, if our app does not make sense on a device that lacks a compass sensor, we can declare the compass sensor as required with the following manifest tag:

```
<manifest ... >

<uses-feature android:name= "android.hardware.sensor.compass"

android:required="true" />

...

</manifest>
```

8.1.1 Platform Versions

It is not necessary for any android device to work on a specific android version, different android devices can run on many different versions of the Android platform, e.g., a device can run on android Version 4.0 or 4.3. Whereas each successive version of this platform very often adds new APIs which were not available in the previous versions. To demonstrate the available set of APIs, each platform version particularizes an API level. For instance, Android 1.0 is API level 1 and Android 4.4 is API level 19. These API levels authorize us to declare the minimum version within which our App is compatible, for instance `<user-sdk>` (a manifest tag) and **minSdkVersion** which is an attribute of the manifest tag helps to set the compatibilities to different versions. An example of this feature, the Calendar Provider APIs were added in Android 4.0 (API level 14). If the app does not function within these APIs, we must declare API level 14 as our APP's minimum supported version. The **minSdkVersion** attribute declares the minimum version with which your app is compatible and the **targetSdkVersion** attribute declares the highest version within which our app has been optimized. Furthermore, we have to be alert that some attributes in the `<user-sdk>` are overridden by corresponding properties in the build.gradle file. Since we are using Android Studio, it is necessary to declare the **minSdkVersion** and **targetSdkVersion** values there instead: For example.,

```
android {

defaultConfig {

applicationId 'com.example.myapp'
```

```
// Defines the minimum API level required to run the app.
```

minSdkVersion 19

```
// Specifies the API level used to test the app.
```

targetSdkVersion 28

```
... } }
```

Every successive version of Android comes with a feature that provides the compatibility for Apps that were built using the APIs from previous versions, so it is necessary for any app to be compatible with future versions of Android while using the documented Android APIs. There are many cases where some Apps uses APIs which are added in more recent Android versions, but does not need them for its elementary functionalities, in such cases we must check the API level at runtime and must be able to gracefully degrade the corresponding features when the API level is too low which can be done by setting the minSdkVersion to the lowest value possible for your app's primary functionality, then compare the current system's version, `SDK_INT` , to one the codename constants in `Build.VERSION_CODES` that corresponds to API level we want to check. For example.,

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
```

```
// Running on something older than API level 11, so disable
```

```
// the drag/drop features that use <code><a href="/reference/android/content/ClipboardManager.html">ClipboardManager </a></code> APIs
```

```
disableDragAndDrop();
```

```
}
```

In our application we performed API version checks on various Files as follows:

GenerateQRCodeFragment (Fragment)

```
if (Build.VERSION.SDK_INT >= 24) {
    try {
        Method m = StrictMode.class.getMethod
            ( 'disableDeathOnFileUriExposure ' );
        m.invoke( null );
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

BasePrinterSettingActivity(Class)

```

if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB_MR1) {
    Message msg = msgHandle.obtainMessage(Common.MSG_WRONG_OS);
    msgHandle.sendMessage(msg);
    return null;
}

```

```

if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB_MR1) {
    Message msg = msgHandle.obtainMessage(Common.MSG_WRONG_OS);
    msgHandle.sendMessage(msg);
    return false;
}

```

MainActivity

```

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    if (checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE)
        != PackageManager.PERMISSION_GRANTED) {
        return false;
    }
}

```

8.2 Screen Compatibility

Android runs various devices that exhibit different screen sizes and pixel densities. The system performs basic scaling and resizing to adapt the user interface to different screens, but this is not sufficient enough there is more work that should be performed to ensure that our UI adapts for each type of screen without behaving in any kind of abnormal or unexpected manner. Android features help our app to adapt accordingly.

8.2.1 Screen Size

The first most visible space which is provided for any app's UI is through the screen size. The screen size is not the actual size of the device screen instead it takes into account the screen orientation, system decorations, and window configuration changes. We can design different layouts that enhance the user experience for the available space on different devices such as phones and tablets. Android also helps to utilize the feature of having alternative layout files that the system applies at runtime based on the current device's screen size. Following we have shown how other smallest width values correspond to typical screen sizes:

- 320dp: a typical phone screen (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, etc).
- 480dp: a large phone screen 5' (480x800 mdpi).
- 600dp: a 7' tablet (600x1024 mdpi).
- 720dp: a 10' tablet (720x1280 mdpi, 800x1280 mdpi, etc).

In our application we used different XML files to make sure our UI Compatibility on different Screen sizes.

```
res/layout/main_activity.xml           # For handsets
res/layout-sw600dp/main_activity.xml  # For 7' tablets
res/layout-sw720dp/main_activity.xml  # For 10' tablets in landscape
```

Figure 8.1, 8.2 and 8.3 shows some samples of different screen sizes:

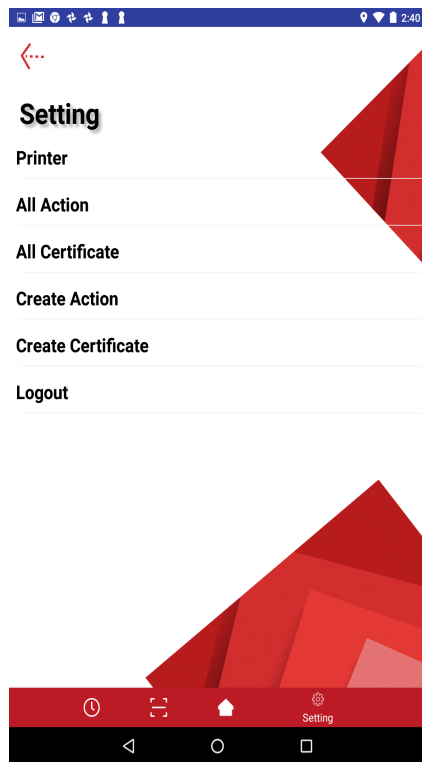


Figure 8.1: Sample of Google Nexus 5x

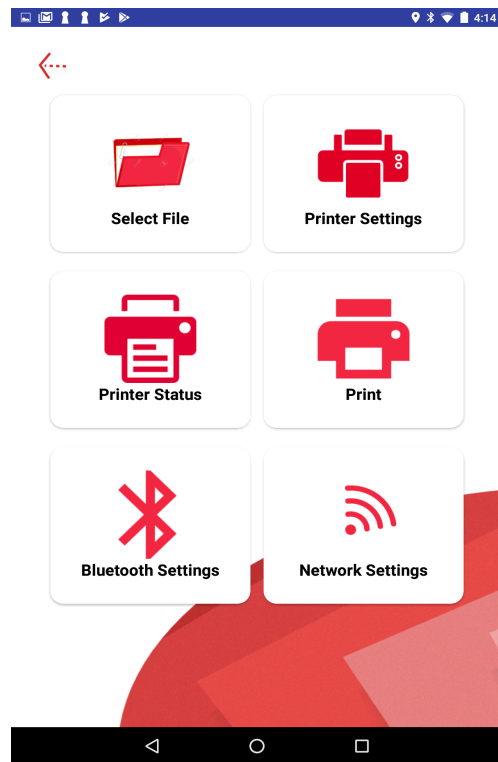


Figure 8.2: Sample of google Nexus 7 (Tablet)

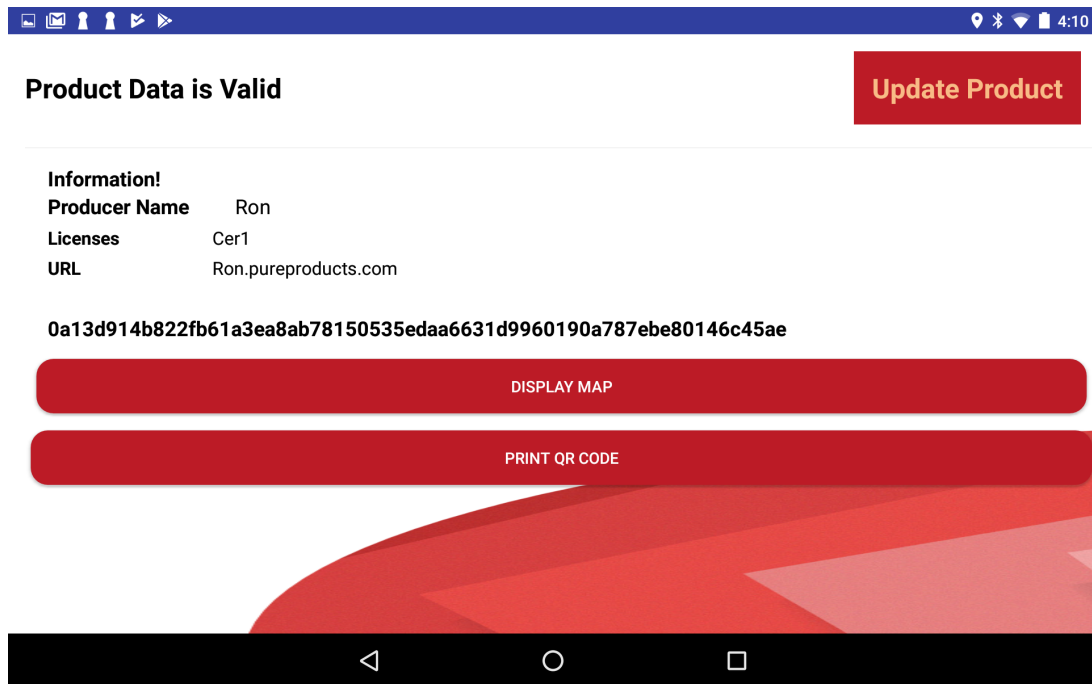


Figure 8.3: Sample of landscape mode

8.3 Test Scenarios

In this section, we tried to measure the performance of our application with the help of performance and load tests.

8.3.1 Performance Test:

Performance testing refers to testing done to analyze and improve our app's performance. This test focuses on optimizing resource consumption by analyzing data collected during testing. Profiler is used to measure the load, CPU and memory utilization, following are the results from the profiler.

Memory Utilization:

1. Code used memory: 7.5 mb
2. Graphics used memory: 61.8 mb
3. Java used memory: 38 mb
4. Others: 2.1 mb

CPU Utilization: App uses 4.9% CPU utilization when we open scanner the utilization is increased to 13%. The average CPU utilization of the whole app is 6%.

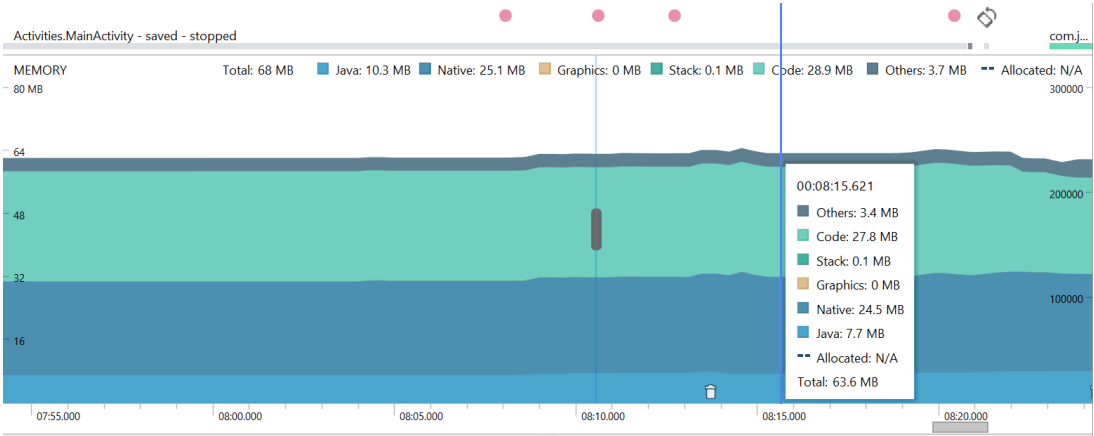


Figure 8.4: Memory Utilization

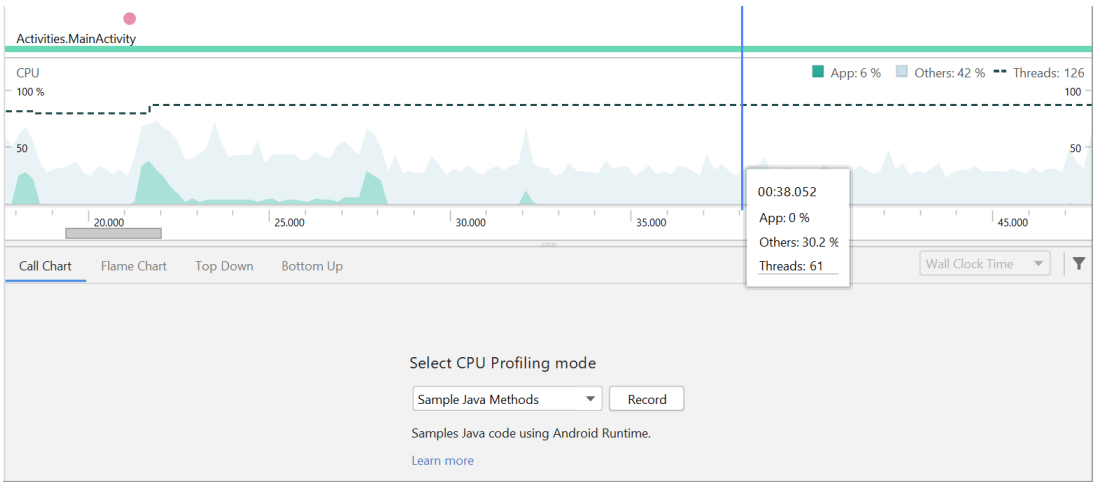


Figure 8.5: CPU Utilization

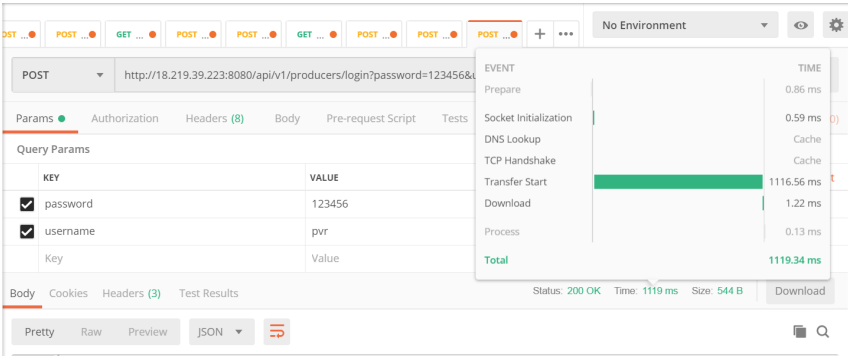


Figure 8.6: Response time for login api

8.3.2 Load Test:

A load test allows us to measure response times, throughput rates, and resources utilized, and to identify our app's breaking point. In load testing, we set criteria to ensure releases of a product meet certain conditions (like the number of users they can support) while delivering a specific response time. We were only able to measure the response time as shown in figure 8.6. On the other hand, to measure throughput rates and resources used is a complex task and is beyond the scope of this project.

Chapter 9

Web Application

To make it as easy as possible for the final consumers to validate products, the web application is created.

This allows consumers to use the web applications both from their computers and smartphones, just by opening its URL in a browser without installing any additional application.

It provides the same functionality as android app in its Consumer mode 5.5. It is a Single Page Application (SPA) [44] build with ReactJs library [45].

One of the main prerequisites is that the web application has to be responsive. To simplify the development of the responsive design, Material-UI framework [46] is utilized along with React. The web application has a very simplistic design with a simple UI interface that is very easy to use. Its landing page is given in figure 9.1.

By clicking the 'Scan Produkt' button, the QR scanner page from figure 9.2 will open when a consumer can scan a product and check its validity. If scanned QR code, which represents a hash, is invalid, the page from the figure 9.3 will be shown. Otherwise, map view with the pins as in figure 9.4, when one pin represents a product being scanned and other pins that lead to that pin/product are the products based on which the product is made. By clicking on any of the pins, the popup with all details of the product are shown as in figure 9.5. The same figure depicts the full-screen mode as well.

One of the main requirements was multi-language support. That is achieved by using Redux library [47] that allows us to manage the global application state easily. Figure 9.6 shows the same product details as figure 9.5 but in German language. Currently, five languages are supported (English, German, Italian, French and Portuguese). This is very easy to extend because Redux allows us to centralize global state and dispatch actions easily from/to any component. Users can change a language by clicking on the top-left button. The popup will appear where all the languages are listed as in figure 9.7.

It can be seen that the simplistic design is achieved by using the latest technologies. That allows anyone with a smartphone and Internet connection to check the validity of the product simply just by scanning its QR code. The application has good performances since it uses the production React build and it is served via the Nginx web server.

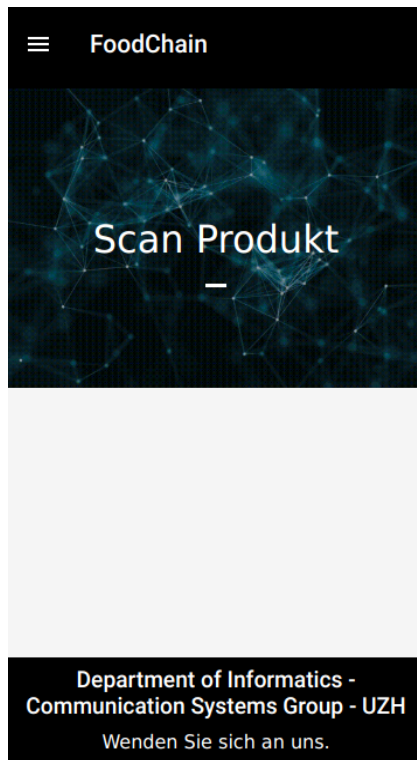


Figure 9.1: Homepage

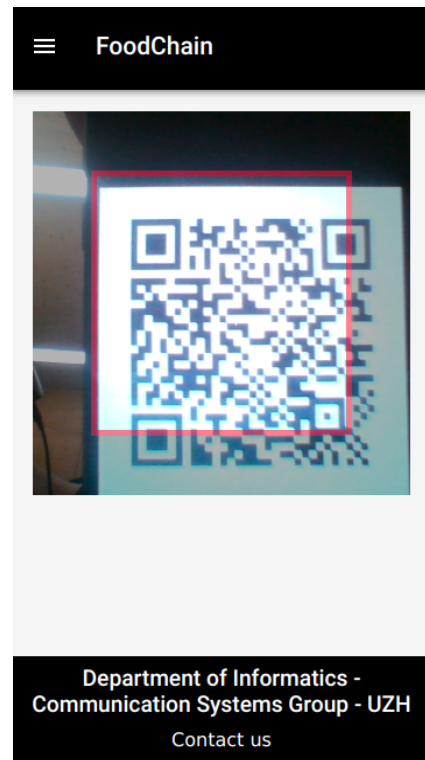


Figure 9.2: QR Scanner Page

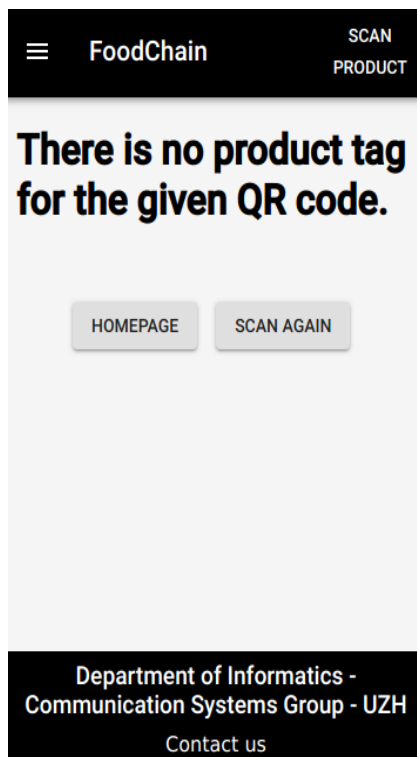


Figure 9.3: Invalid QR Code

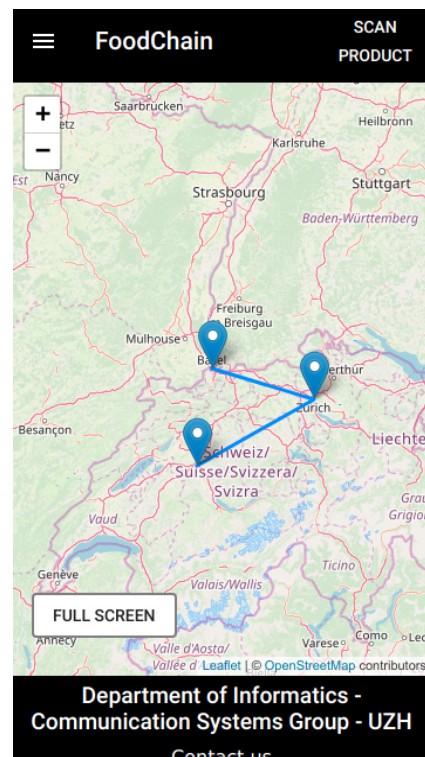


Figure 9.4: Map View on successful scan

The full screen mode can be seen in figure 10.6 in the deployment section 10.

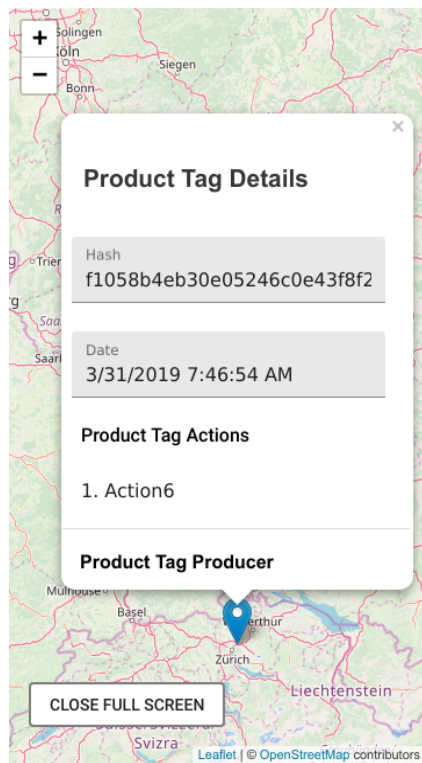


Figure 9.5: Product Details

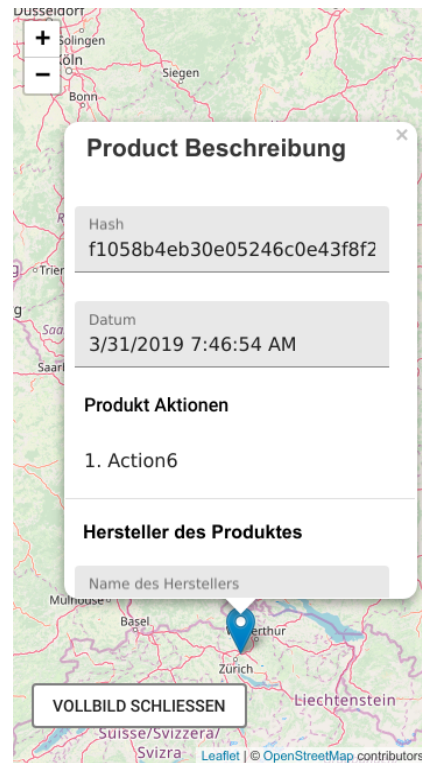


Figure 9.6: German translation

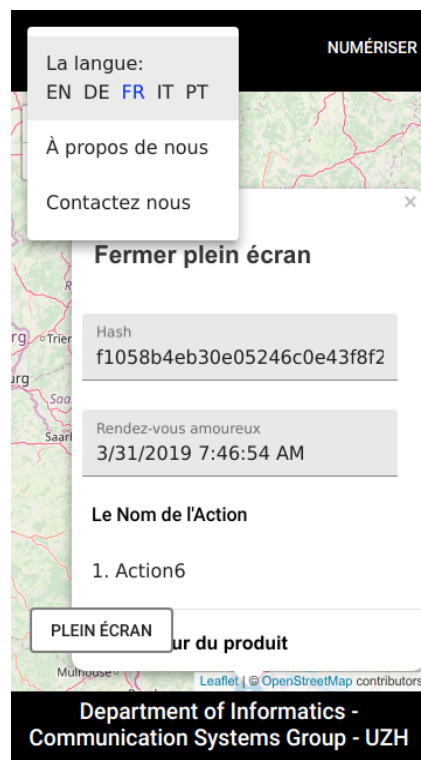


Figure 9.7: Change Language Button and French Translation

Chapter 10

Software Build and Deployment

To build and deploy a system is an important aspect of the software development process. To provide that, the latest practices are utilized. In this chapter, it will be described how to run the Supply Chain Tracking Platform both locally for the development and purpose and on the AWS cloud platform.

10.1 Local Deployment

To spin up the environment on a local machine, the first step you have to do is to clone the FC-Project [48] repository on your local machine.

```
git clone https://itdany@bitbucket.org/foodchainmp/food-chain-platform.git
```

After that, import the project in your favorite IDE.

The main prerequisite is that you have **Docker** [49] and **Docker Compose** [50] installed.

Docker is used to simplifying the developer experience and speed up the deployment process. It abstracts the system from the machine so it eases the deployment of the system in some other environment. It also makes it easier for new developers to join the projects since it requires minimal setup to start the development process.

Each of the components (**fc-server**, **fc-web-client**, **fc-nginx**) has two types of docker files that define the images that will be created for them depending on the environment they will be run on. Docker files define the template based on which we create docker images that we can run as a docker container. Simple visualization of this flow is shown in figure 10.1.

In order to support the image creation on the systems that do not have **NPM** [54], **Maven** [53] or **Java** [27], multi-stage docker build is used [56]. The other benefit is that the target images are much smaller since they just contain **NPM** and **Java** for the **dr-web-client** and **dr-server** respectively while the intermediary images are thrown away.

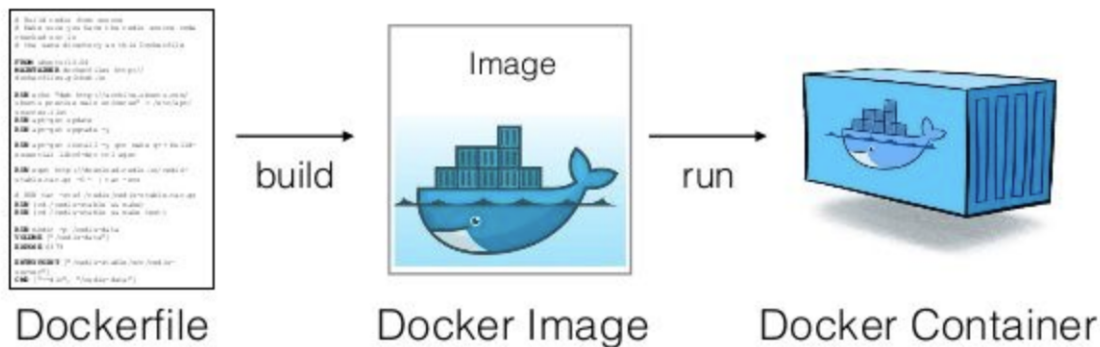


Figure 10.1: Flow from a docker file to a running container [51]

Since this section describes how to run the system locally for the development purpose, the corresponding docker-compose file `docker-compose.dev.yml` file from the project root will be considered. It has the paths to the corresponding docker files of the components. In all three components, the corresponding docker files have the name `Dockerfile.dev`.

The `dev` profile spins up the H2 in-memory database [55]. Also, to ease the local development, it doesn't send/validate product tag hashes on the smart contract but just stores/validate them in the database.

The only thing you should do to run the system locally is to execute the following command from the project root path:

```
docker-compose -f docker-compose.dev.yml up -build
```

This will build the images and run them as docker-compose services. The reason this approach is used even for the local environment which serves the development purpose is that its behavior is as close as possible as you run it in production.

Docker-compose will is a tool used for managing running containers, where each running container represents one service of the system. Simple illustration of the docker-compose is given in figure 10.2.

If the code is changed in one of the components, you do not have to re-run the whole docker-compose environment but you can rebuild only one service while others run seamlessly. After you change the code, the following command has to be executed:

```
docker-compose -f docker-compose.dev.yml up -d -no-deps -build <service-name>
```

This will rebuild and run just the target service/component.

For the web component (`fc-web-client`) it is even simpler because `fc-nginx` keeps the socket connection open so the `npm` development server can communicate with the browser.

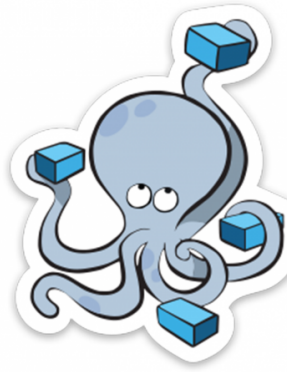


Figure 10.2: Docker-compose illustration [52]

That allows us to see the changes you make in the web project just by pressing **ctrl + s** so no image rebuild required.

By executing the comand: **docker-ps**, you can list all running containers as shown in figure 10.3.

```
daniel@daniel-XPS-L322X:~/Projects/MP/food-chain-platform$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
cb0b6fc335ff	foodchainplatform_fc-nginx	"nginx -g 'daemon of...'"	About an hour ago	Up About an hour	0.0.0.0:8443->80/tcp	fc-nginx-dev
00ad297d06c7	foodchainplatform_fc-web-client	"npm run start"	About an hour ago	Up About an hour		fc-web-client-dev
6f467985b8e7	foodchainplatform_fc-server	"java -Dspring.profil..."	About an hour ago	Up About an hour	8080/tcp	fc-server-dev

Figure 10.3: Listing running containers

10.2 Cloud Deployment

Once the system is deployed in the local environment, it is the time to run it on a server so it is accessible to the world.

We decided to go with Amazon Web Services (AWS) [57]. The reason is that it is a very stable cloud platform and it has a very good free tier offer. That allows us to use many available services for free while having the production-ready system without upfront commitments.

The first step is to create a new virtual machine, the so-called Amazon EC2 instance [58].

Since users do not want to access the web site via a machine's IP, the domain name is bought via Route53 service [61].

We decided for the domain: **foodchain-csg.ch**

After that, the virtual machine's IP is associated with the domain name. It is important to do this in the beginning because in the next step we will have to prove the ownership of the domain to the Certificate Authority.

Since the security is paramount, HTTPS is enabled. We used a free, open-source Certificate Authority called Let's Encrypt [62].

To obtain an SSL certificate, we have to SSH on the virtual machine and install the Certbot program [63]. The detail description of how to install Certboot and obtain the SSL certificate for our system can be found in the link:

<https://certbot.eff.org/lets-encrypt/ubuntuubionic-nginx>

The next step is to SSH to the machine and installs the following programs:

- git
- docker
- docker-compose
- tmux [60]

After that is done, the next step is to clone the project from the remote repository [48].

SSL certificate can be obtained automatically. For that purpose, there is a separate docker service (Certbot) that obtains the certificates and also renews it automatically. Detail description of the certbot service can be found here: <https://github.com/wmnnd/nginx-certbot>

To enable Nginx to be able to serve HTTPS traffic we have to provide it the obtained SSL certificates. The following files are provided by the Certbot service and passed inside the Nginx container.

- fullchain.pem
- options-ssl-nginx.conf
- privkey.pem
- ssl-dhparams.pem

Now an external database server should be created. We decided on PostgreSQL available under AWS free tier [59]. The micro instance is created along with the database called: `food_chain_db`. No further settings for the database required since the JPA/Hibernate will create the tables based on the domain classes.

The final thing that has to be considered before running the system is the file with the environment variables for the server application that will be passed inside the running docker containers. The file's location is: `project-location/Env/fc-server-aws.env`. The following variables have to be provided:

- `SPRING_PROFILES_ACTIVE` - active spring profile, in this case, `aws`
- `SPRING_DATASOURCE_URL` - database URL

- `SPRING_DATASOURCE_USERNAME` - username of the database user
- `SPRING_DATASOURCE_PASSWORD` - database password
- `FOODCHAIN_WEB3J_HTTPSERVICE_URL` - URL to the remote Ethereum node that can execute transactions on our behalf.
- `FOODCHAIN_CONTRACT_ADDRESS` - address of the deployed contract on the Ethereum network.
- `FOODCHAIN_ACCOUNT_PRIVATEKEY` - private key of our Ethereum account with some amount of ether on it.

We decided to use an external Ethereum client provided by Infura platform [64]. The reasons are that it is more cost-effective because we do not have to pay for the storage and CPU power, and because it brings more stability to the system.

Before the final command is run, it would be worth inspecting the docker-compose file with the specified services.

```
version: '3'
services:
  fc-server-aws:
    container_name: fc-server-aws
    restart: always
    env_file:
      - Env/fc-server-aws.env
    build:
      dockerfile: Dockerfile
      context: ./fc-server

  fc-web-client-aws:
    container_name: fc-web-client-aws
    build:
      dockerfile: Dockerfile
      context: ./fc-web-client
    volumes:
      - /app/node_modules
      - ./fc-web-client:/app
    restart: always

  fc-nginx-aws:
    container_name: nginx
    restart: always
    build:
      dockerfile: Dockerfile
      context: ./fc-nginx/aws
    ports:
```

```
- 80:80
- 443:443
depends_on:
- fc-server-aws
- fc-web-client-aws
```

It can be seen that all the services have the restart policy set to **always**. This is important because if the container crashes or the whole virtual machine restarts for some reason, the containers will restart automatically with minimal downtime.

At this stage, everything is set up so we can run **docker-compose** command:

```
docker-compose -f docker-compose.aws.yml up -build
```

Figure 10.4 depicts the components and how they communicate when the system starts. The Internet-facing component that handles the requests from the android and web applications is Nginx reverse proxy. Its only job is to sit in front of the other components and routes the requests according to the URL path, and to store the SSL certificates which enable HTTPS traffic.

If the URL path starts with **/api**, the Nginx will route the request to the **fc-server** component, otherwise it will forward the requests to the **fc-web-client**. The benefits of this approach are that all the requests towards the backend coming from the web application have the same origin. This means that we avoid the problems related to the CORS (Cross-Origin Resource Sharing) policy [39].

It can be seen that the server and web applications are running on the same EC2 instance. As mentioned, in this scenario, an external database instance is used. That brings more stability to the system as well as automatic backup provided by the AWS RDS service.

The status of the services can be checked by running the command: **docker ps** as in figure 10.5.

Once the docker-compose brings up the services, the web application should be accessible under the URL: <https://foodchain-csg.ch/> with the landing page as in figure 10.6.

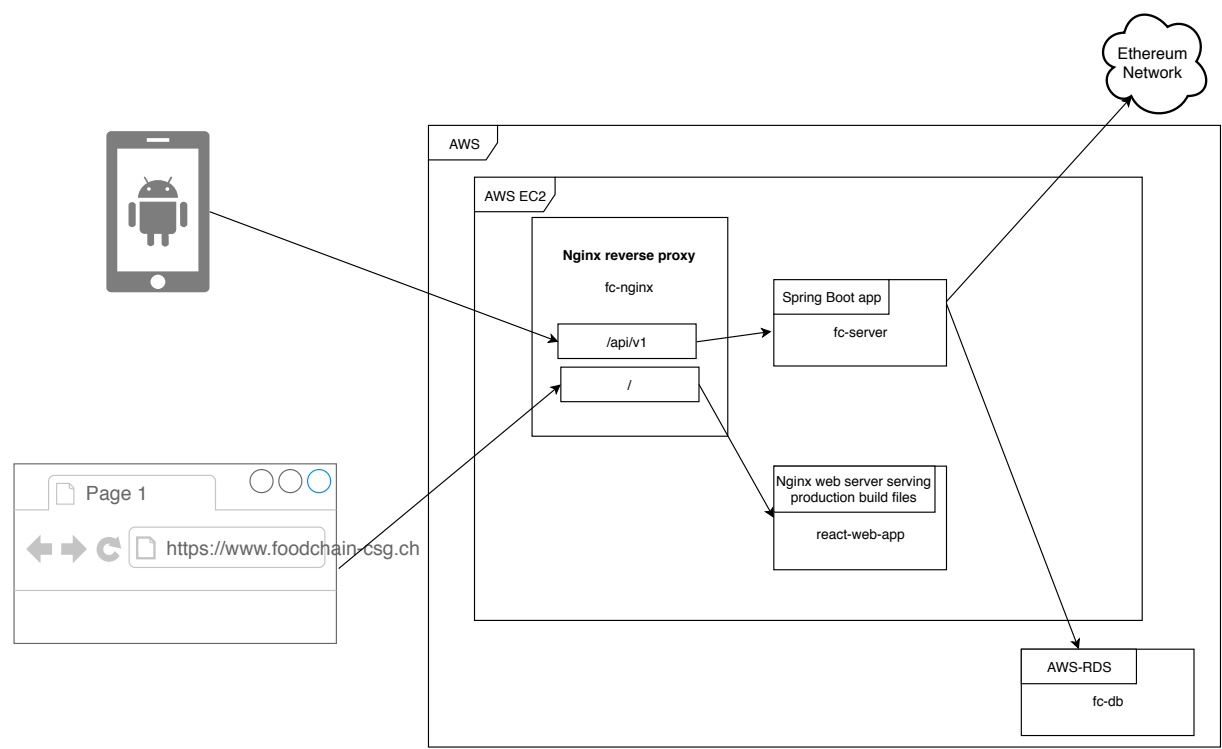


Figure 10.4: AWS Component Diagram

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
64b84999dca7	food-chain-platform_fc-server-aws	"java -jar fc-server..."	6 days ago	Up 6 days	8880/tcp	fc-server-aws
fc7db711de25	food-chain-platform_fc-nginx-aws	"nginx -g 'daemon of..."	3 weeks ago	Up 3 weeks	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp	nginx
1ee9657cfd3e	food-chain-platform_fc-web-client-aws	"nginx -g 'daemon of..."	4 weeks ago	Up 3 weeks	80/tcp, 8050/tcp	fc-web-client-aws

Figure 10.5: Listing running docker containers

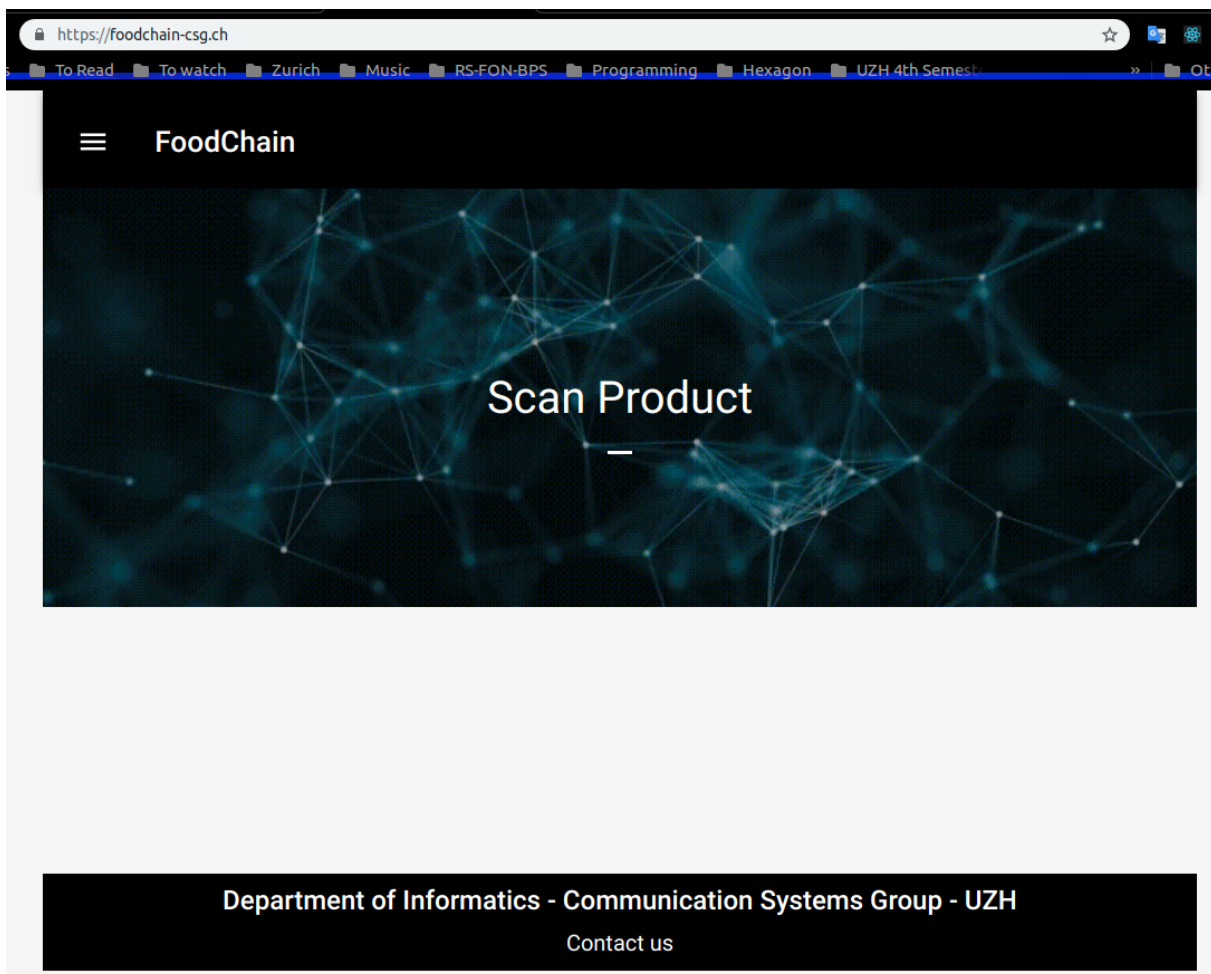


Figure 10.6: Landing page

Chapter 11

Modification Examples

A simple modification example is described in this section for future extensions in the system. We assume that there is a new requirement to add an action date in the system. The following sections will describe the necessary steps to implement this change.

11.1 Server Application

Before adding new functionalities to the system, all components that can be affected by that change should be considered. In our use case where we add the data field to the action objects, all components are affected by this change.

Since the whole flow in the system starts from the database, we will first implement the aforementioned change in the data model layer. That can be achieved by updating the domain class `Action.java`. The field with the annotation as in the next code snippet should be added:

```
@Column(nullable = false)
private LocalDateTime date;
```

It defines the field named `date` of the java date type `LocalDateTime`. The annotation `Column` defines that the field can not be `null`. The field `nullable = false` tells that the field `date` can not have `null` value.

No getter and setter method are required since we use project Lombok library that will do that for us.

This change will update the data model by the Hibernate framework when the system starts.

The next step is to update the class `ActionService.java` by setting the date when a new action is created. The exact method is `createAction`. After the new action is validated, the date should be set like this: `action.setDate(LocalDateTime.now())` just before the

method `actionRepository.save(action)` is called. This will set the action date to the current date when the action is created.

Since we do not have a WSDTO class for the action, we send to the client applications the objects that represent directly the model class. The reason is that the action class is very simplistic and we would not benefit by having the WSDTO action class. Since that is the case, no further changes are required on the backend part because the Hibernate and JPA frameworks will automatically compound the action objects when they are retrieved from the database.

11.2 Android

System modifications and changes may include new features and functionalities. Modifying the existing code and updating the user interfaces requires the understanding of application e.g. architectural components, frameworks, structure and business flow of the system. An overview of the android application code structure is defined in the Code Architecture. A change in one part of code directly or indirectly affects other parts of the system or also to deliver a change in a reliable way we need to follow the architecture of code, and follow all necessary steps. We need to add a variable `actionDate` inside `ProducerActions.java` model class where we need to define setter and getter methods for `actionDate`. Once the model is defined, we can get the `actionDate` in `ProducerMapFragment.java` class by calling `actionDate` getter method. Then, we need to pass `actionDate` information to `MapUtils.java` class, where we set that information to `MapDataModel.java` (add a new attribute and setter getter class for `actionDate` inside `MapDataModel.java`), finally we can get the information into the `ProducerMapFragment.java` class and set `actionDate` information into `addMarkersToMap()` method and set the action date to markers in the map.

11.3 Web Application

The final step to successfully modify the action class is to update the web application. The only place where the web application should be updated is the `ActionList` file located under the path:

```
fc-web-client/src/components/fragments/ActionList.js
```

Inside the `Listitem` tag, along with the action name, action date should be added. That could be achieved by adding the following tag:

```
<ListItemText primary={new Date(action.date).toDateString()}/>
```

After this change, the action date should be successfully rendered on the web page, action date should be displayed along with the action name.

Chapter 12

Evaluation and System Analysis

This chapter discusses the summary and evaluation of the Blockchain-based Supply Chain Tracking Platform. A complete overview of the newly developed system has been discussed in the previous sections of the report. It also provides an overview of the source and quality of code as described in the above sections. It also provides an analysis of the Blockchain-based Foodchain Tracking Platform. We have tested the platform for its functionalities in different onsite settings. The back-end server has been deployed on a secure VM provided by the University of Zurich as well as on the AWS. A practical evaluation especially the performance and security will be completed in the final version of the report. The supply chain tracking platform provides a fully functional proof-of-concept running on the production environment, and the front-end android application that communicates with the server. We didn't face any disruptions in the final release of the platform.

12.1 Software Development

We used native android SDK for the development of the android application as the front-end part of the platform. Android Studio, the official integrated development environment (IDE) for Google's Android operating system for Android development. It provides several features such as [15]:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch the performance, usability, version compatibility, and other issues
- It provides ProGuard integration and app-signing capabilities
- Template-based easy wizards to create common Android designs and components for the development
- A rich layout editor that helps the users to drag-and-drop UI components

- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

It provides support to build, compile and run the application on virtual emulators as well as real devices. The debug feature allows an easy and smooth approach to quickly fix the problems during the development of [15].

12.2 Unit Testing

In order to produce a high-quality android application, we tested the individual units and components of the system. The purpose of these unit tests was to verify each unit of the application is working according to the requirements and objectives set in the task description. Here, the unit can be considered each testable part of the application which may have more than one inputs and a single output. A method is a simple example which may belong to the parent or derived class.

We also used the Espresso unit testing framework [15] provided by Android Studio in order to test the UI components of the application. The unit testing method is usually followed by white box testing. It is considered the initial phase of testing before the integration. It increases the quality of the system for maintaining the code and future changes. The code can be reused for future extensions. It makes the development fast and easy with more confidence to build a quality application.

12.3 Security

The android application provides the built-in robust security that decreases security issues to a large extent. The application requires to have permissions such as location, file read/write, etc. to run it in a secure manner. In order to make the application highly secure, we followed the app security best practices defined by the android developer community for preserving the trust and integrity of the device. It also includes secure communication with the printer to safeguard the encrypted hash code that is exchanged between the application and printer. The android application allows communicating with the p2p module in a secure manner. In addition, the android application communicates with the back-end server that has a certificate issued by a trusted CA, using HTTPS.

12.4 Robustness

The food supply chain tracking platform deals with the user's data and financial transactions. The robustness of the system plays a significant role. This section evaluates the robustness of front-end and back-end which measures the tolerance of the system in case of any failures such as a loss of connectivity or abrupt failure.

Back-end

If for some reason some of the services on the back-end side stop working, the restart policy set to **always** will try to bring them up again. Since on the AWS environment, the AWS RDS service is used, it is highly unlikely that the server can lose the connection to the database. Also, the data is backed up every week so in case of a failure or if the system should be redeployed, the data can be recovered.

Front-end

If the back-end stops working, all the endpoints used by the front-end application go into not responding state. For example, new producers cannot register with the system, existing users cannot log into the application which halts them to use the functionality of the application. It will be required to restart the server to recover from the failure.

P2P Module

Although the P2P module is deployed independently on the server, the android application provides the entry point for accessing the OySy wallet. Any failure on the backend server does not functionally stop the P2P module. Android users will be able to access the OySy wallet if the producer is already logged into the application.

Printing Module

This module does not have any dependency on the back-end server. If the server is not responding which means producers cannot generate QR codes or they cannot see the history of previously generated QR codes, consequently, producers will not be able to print the tags through the printer. However, if the producer is already logged into the android application, the printing module can be accessed easily.

12.5 Challenges

We fixed the reported compatibility issues such as location services at different devices that were reported during the demonstration of the platform. We received positive feedback and suggestions for improvements related to design changes, and we implemented these changes in the system. In addition, we received a change request to add Multi QR functionality in the system. We already had developed the basic underlying architecture and components on the front-end and back-end. It was very challenging to enable this change modifying the basic flow of the application and making the system compliant with the new requirements. We faced severe technical difficulties during the onsite live demonstration in January 2019. We enabled the platform fully functional after receiving

the feedback and fixed the vulnerable points such as deploying the system to AWS for high stability and full compatibility of front-end and back-end. It also helped us to test the integration of front-end and backend, and execute the different test scenarios in order to make the system more robust and stable.

12.6 Onsite-Experiments

We tested the food supply chain platform with different onsite settings to completely utilize the functionality in a real scenario. A successful evaluation was performed at the Federal Office of Agriculture, Bern showing the required functionalities. We registered products at four different points on the way to Bern for demonstration purpose and we displayed the tracking of each product with complete information by scanning the final product. It eventually led to the successful submission of ICBC 2019 - Demo paper 193028.

Chapter 13

Summary and Future Considerations

The main objective of this Master Project was to design and develop a blockchain-based supply chain tracking platform that meets the security and scalability challenges. It allows industries to make themselves more transparent and traceable by registering their products on a blockchain. The blockchain-based supply chain tracking platform is developed as a proof of concept with the aim to build trust in consumers by allowing them to track the real history of products anytime. The basic functional prototype was developed in the independent study module, where the android was able to track the history of products with minimal features.

The producers and consumers can easily use it by installing the application on their smartphones. It basically allows the producers to create QR codes, perform actions on products. On the other side, consumers can scan the QR code to track the history of products on a very simple map[17].

We extended the features of the food supply chain in this project, it is connected with the blockchain test network where the hashes of QR codes are stored for tracking the complete history of a product in a transparent way. A data server on the back-end stores information such actions, time, date, location again each hash value. A printing module was added to allow the producers to print the QR tag immediately after performing actions on the product. To this end, we used a label printer that is connected with the android platform in order to print the tags. The system allows printing the QR codes using different means of communication.

A complete interactive map view is developed to see the tracking of a product using markers at different locations of producers in ascending order. The multi QR feature in the system allows us to scan and create multiple QR codes, where each QR code can be connected with a previous single or chain of QR codes. The system computes recursively the complete tracking of a product. This feature is developed to enable the industrial use case where one product can be combined with products from many other sources. The system allows us to visualize the history of currently generated QR codes by a producer, and it also allows to print or share the QR codes using any third-party application installed on the phone.

The OySy wallet is integrated with the supply chain tracking platform to enable P2P payments. To this end, the OySy wallet is connected with Ethereum test network, and it allows producers to send payments to another producer using their public address. In addition, it manages the accounts of a producer and transaction history. However, it can be extended in the future by adding other features such as supporting multiple crypto-currencies, currency conversions, dynamic fee calculations.

Finally, the website is created which allows the final consumers to validate the products without installing any additional components.

13.1 Future Considerations

The blockchain-based supply chain tracking platform is fully capable to function as proof of concept in the food supply chain industry. This master project has the potential to be generalized for many other business use cases in the supply chain industry. The evolution of blockchain is going beyond the FinTech industry creating a cascading impact across several domains in the industry. In today's highly competitive environment, getting the correct products to the right place at the right time via the most cost-effective manner is paramount for business success. It's imperative for businesses to innovate new ways to streamline their supply chain and optimize productivity. The blockchain has the potential to take the power of information from the food monopolies and place it into the hands of the direct customer.

Through the use of a simple QR code and a smartphone, customers can scan a package and receive a full and complete history of the journey of their food. Smart contracts can play a significant role in digitizing the whole supply chain by leveraging the potential of IoT. They would solve the complicated problems such as fraud, corruption, tampering and falsifying information by malicious parties. It will build trust among customers in the supply chain industry by showing the provenance of products.

Bibliography

- [1] Provenance: Introductio to Blockchains for supply chain transparency,[online] Available: <https://www.provenance.org/tracking-tuna-on-the-blockchain>, Last Visit: 6.18.2019
- [2] CISION: The 2016 Food Revolution Study,[online] Available: <https://www.prnewswire.com/news-releases/study-ninety-four-percent-of-consumers-say-food-product-transparency-from-brands-and-manufacturers-is-important-impacts-purchase-300287311.html>, Last Visit: 6.08.2019
- [3] M. Weber and M. Boban, "Security Challenges of the Internet of Things," *9th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2006, pp. 638-643.
- [4] Abramowicz, Michael, "Cryptocurrency-Based Law," *GWU Law School Public Law Research*, 2015, pp. 2015-9. [online] Available: <https://ssrn.com/abstract=2573788> or <http://dx.doi.org/10.2139/ssrn.2573788>, Last Visit: 6.08.2019.
- [5] Minhaj Ahmad Khan, Khaled Salah, "IoT security: Review, Blockchain Solutions, and Open Challenges" *Future Generation Computer Systems*, (Vol. 82), 2018.
- [6] Satoshi Nakamoto, "Bitcoin: A peer-to-Peer Electronic Cash" [online] Available: <https://bitcoin.org/bitcoin.pdf>, Last Visit: 6.08.2019.
- [7] Aung, M. Min, & Chang, Y. Seok, "Traceability in a food supply chain: Safety and quality perspectives," *Food control*, 2013, pp. 172-184.
- [8] Boyacia, I.H., Temiza, H.T., Uysala, R.S., Veliogluc, H.M., Yadegaria, R.J., Rishkana, M.M. "A novel method for discrimination of beef and horse-meat using Raman spectroscopy," *Food Chemistry*, 2014, pp. 148, 37-41, Available: <http://www.labguide.com.tw/user/w016267551/upload/file/beef-horse%20food%20chem.pdf>, Last Visit: 6.08.2019.
- [9] Archana Prashanth Joshi, Meng Han and Yan Wang , " A Survey on Security and Privacy Issues of Blockchain Technology," (Vol.1, No.2) 2018, pp. 121-147, Available: https://www.researchgate.net/profile/Archana_Joshi9/publication/325173502_A_survey_on_security_and_privacy_issues_of_blockchain_technology/links/5b40d165a6fdccbcf9079f73/A-survey-on-security-and-privacy-issues-of-blockchain-technology.pdf, Last Visit: 6.08.2019.

- [10] H. Gross, M. Holbl, D. Slamanig, and R. Spreitzer, "Privacy-Aware Authentication in the Internet of Things," *Cryptology and Network Security. Springer International Publishing* (Vol.1, No.2) 2015, pp. 32-39.
- [11] Skarmeta, Antonio F., Jose L. Hernandez-Ramos, and M. Moreno, "A decentralized approach for security and privacy challenges in the internet of things," *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, 2015.
- [12] Codepath: Consuming APis with Retrofit , Available :<https://guides.codepath.com/android/consuming-apis-with-retrofit>, Last Visit: 6.08.2019.
- [13] INNOVATION Technology Solution: Request-Response Handling APis with Retrofit , Available :<http://blogs.innovationm.com/retrofit-library-in-android/>, Last Visit: 6.08.2019.
- [14] Manoj Kumar Choubey (2012) IT Infrastructure and Management (For the GBTU and MMTU). p. 53
- [15] Android Developers: Application Fundamentals, Google, <https://developer.android.com/guide/components/fundamentals>, 28.01.2019
- [16] Frank Ableson: Introduction to Android development, IBM, <https://developer.ibm.com/articles/os-android-devel/>, 28.01.2019
- [17] Atif Ghulam Nabi: Design and Development of an Android-based Supply Chain Tracking Application, University of Zurich, https://files.ifi.uzh.ch/CSG/staff/Rafati/Atif_IS.pdf, 20.03.2019
- [18] Brother:QL-820NWB,Brother,<https://www.brother.ch/de-ch/etikettendrucker/ql-820nwb>, 30.03.2019
- [19] Brother's Developers: Application Development Tool for Android, Brother, <https://www.brother.co.jp/eng/dev/mobilesdk/android/index.aspx>, 03.04.2019
- [20] Jan von der Assen: A Progressive Web App(PWA)-based Mobile Wallet for Bazo, University of Zurich,
- [21] Jan von der Assen: Web-based Wallet for Bazo , University of Zurich, <https://github.com/bazo-blockchain/bazo-wallet> <https://files.ifi.uzh.ch/CSG/staff/bocek/extern/theses/BA-Jan-von-der-Assen.pdf>, 03.04.2019
- [22] Author: Ethereum JavaScript API, Web3js, <https://web3js.readthedocs.io/en/1.0/>, 09.04.2019
- [23] Hannu Jaakkola and Bernhard Thalheim. (2011) "Architecture-driven modelling methodologies." In: Proceedings of the 2011 conference on Information Modelling and Knowledge Bases XXII. Anneli Heimbäcker et al. (eds). IOS Press. p. 98
- [24] Wikipedia website, Representational state transfer. Available at: https://en.wikipedia.org/wiki/Representational_state_transfer. Last visited on: 13.04.2019

- [25] Wikipedia website, Separation of concerns. Available at: https://en.wikipedia.org/wiki/Separation_of_concerns. Last visited on: 13.04.2019
- [26] Web3j website, WHERE JAVA MEETS THE BLOCKCHAIN. Available at: <https://web3j.io/>. Last visited on: 13.04.2019
- [27] Oracle Java website Available at: <https://go.java/index.html>. Last visited on: 13.04.2019
- [28] Nginx website. Available at: <https://www.nginx.com>. Last visited on: 13.04.2019
- [29] JavaScript Mozilla website, Java Script. Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Last visited on: 13.04.2019
- [30] Wikipedia website, MySQL. Available at: <https://en.wikipedia.org/wiki/MySQL>. Last visited on: 13.04.2019
- [31] Wikipedia website, Spring Security. Available at: https://en.wikipedia.org/wiki/Spring_Security. Last visited on: 13.04.2019
- [32] Spring website, Spring Boot. Available at: <https://spring.io/projects/spring-boot>. Last visited on: 13.04.2019
- [33] JWT website. Available at: <https://jwt.io/>. Last visited on: 13.04.2019
- [34] Wikipedia website, Swagger. Available at: [https://en.wikipedia.org/wiki/Swagger_\(software\)](https://en.wikipedia.org/wiki/Swagger_(software)). Last visited on: 13.04.2019
- [35] Wikipedia website, Jackson. Available at: [https://en.wikipedia.org/wiki/Jackson_\(API\)](https://en.wikipedia.org/wiki/Jackson_(API)). Last visited on: 13.04.2019
- [36] Wikipedia website, Google Guava. Available at: https://en.wikipedia.org/wiki/Google_Guava. Last visited on: 13.04.2019
- [37] ProjectLombok website, Project Lombok. Available at: <https://projectlombok.org>. Last visited on: 13.04.2019
- [38] Martin Fowler, Data Transfer Object. Available at: <https://martinfowler.com/eaaCatalog/dataTransferObject.html>. Last visited on: 13.04.2019
- [39] Developer Mozilla Website, Cross-Origin Resource Sharing (CORS). Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. Last visited on: 13.04.2019
- [40] C2 Website, Persistence Layer. Available at: <http://wiki.c2.com/?PersistenceLayer>. Last visited on: 13.04.2019
- [41] Wikipedia Website, Smart Contract. Available at: https://en.wikipedia.org/wiki/Smart_contract. Last visited on: 13.04.2019
- [42] Solidity Website, Solidity. Available at: <https://solidity.readthedocs.io/en/v0.5.6>. Last visited on: 13.04.2019

- [43] Rinkeby Website, Rinkeby. Available at: <https://www.rinkeby.io>. Last visited on: 13.04.2019
- [44] Wikipedia Website, Single-page application. Available at: https://en.wikipedia.org/wiki/Single-page_application. Last visited on: 13.04.2019
- [45] React Website, React - A JavaScript library for building user interfaces. Available at: <https://reactjs.org>. Last visited on: 13.04.2019
- [46] Material-UI Website, React components that implement Google's Material Design. Available at: <https://material-ui.com>. Last visited on: 13.04.2019
- [47] Redux Website, A predictable state container for JavaScript apps. Available at: <https://redux.js.org>. Last visited on: 13.04.2019
- [48] Bitbucket FooChain Repository, food-chain-platform. Available at: <https://bitbucket.org/foodchainmp/food-chain-platform/src/master>. Last visited on: 13.04.2019
- [49] Docker Website, Enterprise Container Platform for High Velocity Innovation. Available at: <https://www.docker.com>. Last visited on: 13.04.2019
- [50] Docker Website, Docker Compose. Available at: <https://docs.docker.com/compose>. Last visited on: 13.04.2019
- [51] Medium Website, Nilesch Jayanandana, Build a Docker Image just like how you would configure a VM. Available at: <https://medium.com/platformer-blog/practical-guide-on-writing-a-dockerfile-for-your-application-89376f88b3b5>. Last visited on: 15.04.2019
- [52] Nearsoft Website, Jayro Salgado, Docker Compose v3 â Production Ready. Available at: <https://nearsoft.com/blog/docker-compose-v3-production-ready>. Last visited on: 15.04.2019
- [53] Apache Maven Project, Maven. Available at: <https://maven.apache.org>. Last visited on: 13.04.2019
- [54] NPM Website, Node Package Manager. Available at: <https://www.npmjs.com>. Last visited on: 13.04.2019
- [55] H2 Website, H2 Database Engine. Available at: <https://www.h2database.com/html/main.html>. Last visited on: 13.04.2019
- [56] Docker Docs Website, Multistage Build. Available at: URL. Last visited on: 13.04.2019
- [57] AWS Website, Amazon Web Services. Available at: <https://aws.amazon.com>. Last visited on: 13.04.2019
- [58] AWS Website, Amazon Elastic Compute Cloud (Amazon EC2). Available at: <https://aws.amazon.com/ec2>. Last visited on: 13.04.2019
- [59] AWS Website, Amazon Relational Database Service (RDS). Available at: <https://aws.amazon.com/rds>. Last visited on: 13.04.2019

- [60] GitHub Website, TMUX. Available at: <https://github.com/tmux/tmux>. Last visited on: 13.04.2019
- [61] AWS Website, Amazon Route 53. Available at: <https://aws.amazon.com/route53>. Last visited on: 13.04.2019
- [62] Letsencrypt Website, Letsencrypt. Available at: <https://letsencrypt.org>. Last visited on: 13.04.2019
- [63] Certbot Website, Automatically enable HTTPS on your website with EFF's Certbot, deploying Let's Encrypt certificates. Available at: <https://certbot.eff.org>. Last visited on: 13.04.2019
- [64] Infura Website, YOUR ACCESS TO THE ETHEREUM NETWORK. Available at: <https://infura.io>. Last visited on: 13.04.2019
- [65] R. Shirley: Internet Security Glossary, Version 2, RFC 4949 (Informational), IETF, Fremont, CA, USA, August 2007. Available at: <https://www.rfc-editor.org/rfc/pdf/rfc4949.txt.pdf>. Last visited on: 10.08.2019
- [66] SANS Institute: IT security Resources. Available at: <https://www.sans.org/it-security/>. Last visited on: 10.08.2019
- [67] The National Academies Press, Computers at Risk: Safe Computing in the Information Age. Chapter: Concepts of Information Security Available at: <https://www.nap.edu/read/1581/chapter/4>. Last visited on: 10.08.2019
- [68] Geeks for Geeks: Types of Security Attacks| Active and Passive attacks. Available at: <https://www.geeksforgeeks.org/types-of-security-attacks-active-and-passive-attacks/>. Last visited on: 10.08.2019
- [69] Medium : Why MVC Architecture ,[online] Available at : <https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76>. Last visited: 10.08.2019
- [70] MDN we docs : MVC Architecture ,[online] Available at : https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture. Last visited : 10.12.2018
- [71] Android Developers : Device Compatibility Overview. Available at : <https://developer.android.com/guide/practices/compatibility> Last visited: 10.08.2019
- [72] Android Developers : Layouts. Available at : <https://developer.android.com/guide/topics/ui/declaring-layout> Last visited: 10.08.2019
- [73] Android Developers : Linear-Layout. Available at : <https://developer.android.com/guide/topics/ui/layout/linear> Last visited: 10.08.2019
- [74] Android Developers : Relative-Layout. Available at : <https://developer.android.com/guide/topics/ui/layout/relative> Last visited: 10.08.2019

Abbreviations

REST	Representational State Transfer
API	Application Programming Interface
SQL	Structured Query Language
JVM	Java Virtual Machine
URL	Uniform Resource Locator
ER	Entity Relationship
QR Code	Quick Response Code
JPA	Java Persistence API
CRUD	Create-Read-Update-Delete
WSDTO	Web Service Document Transfer Object
DB	Database
CORS	Cross-Origin Resource Sharing
MVC	Model-View-Controller
JSON	JavaScript Object Notation
JWT	JSON Web Token
SPA	Single Page Application
UI	User Interface
IDE	Integrated Development Environment
NPM	Node Package Manager
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
IP	Internet Protocol
HTTPS	Hyper Text Transfer Protocol Secure
SSL	Secure Sockets Layer
SSH	Secure Shell
CPU	Central Processing Unit
RDS	Relational Database Service

List of Figures

2.1	Context Diagram	5
2.2	System architecture.	6
2.3	Container Diagram for Food Chain Tracking System.	7
2.4	Folder Structure of the Food Chain Tracking System.	8
3.1	Server's folder structure	11
3.2	Server Layers	12
3.3	ER Diagram	13
3.4	Previous Product Tag Validation	17
3.5	Create Product Tag - Sequence Diagram	24
5.1	Dalvik VM versus Android runtime [16]	29
5.2	System Diagram [17]	30
5.3	Use Case Diagram	31
5.4	Code Architecture	33
5.5	Architectural Components	34
5.6	Home Page	37
5.7	First QR Code	38
5.8	Discard Previous QR Code	38
5.9	Product Data is Valid	40
5.10	Map View	41
5.11	Product's Information on map	41

5.12 History of QR Codes	42
5.13 French Translation	43
5.14 German Translation	43
5.15 Italian Translation	43
5.16 Portuguese Translation	43
5.17 Processing Flow of Printer[19]	45
5.18 Printer	45
5.19 Printer Settings	45
5.20 Basic Settings	46
5.21 Bluetooth Settings	46
5.22 P2P Module	48
5.23 Accounts	49
5.24 P2P Module [20]	49
6.1 Visualization of a view hierarchy with layout parameters associated with each view [72].	52
6.2 Login View	53
6.3 Registration View	53
6.4 List of Categories	58
6.5 Selected Category is Tea	58
6.6 Certificates List related to Tea	60
6.7 Selected Certificates	60
6.8 Action and Processes List	60
6.9 Selected Action	60
6.10 All Action List	64
6.11 All Certificate List	64
6.12 New certificate is created	64
6.13 New certificate belongs to category Coffee	64
6.14 New action is created	65

6.15	New action belongs to category Coffee	65
7.1	MVC Architecture	70
7.2	Encryption Process	73
7.3	Decryption Process	73
7.4	Requets-Response Handling [13]	75
8.1	Sample of Google Nexus 5x	80
8.2	Sample of google Nexus 7 (Tablet)	80
8.3	Sample of landscape mode	81
8.4	Memory Utilization	82
8.5	CPU Utilization	82
8.6	Response time for login api	82
9.1	Homepage	85
9.2	QR Scanner Page	85
9.3	Invalid QR Code	85
9.4	Map View on successful scan	85
9.5	Product Details	86
9.6	German translation	86
9.7	Change Language Button and French Translation	86
10.1	Flow from a docker file to a running container [51]	88
10.2	Docker-compose illustration [52]	89
10.3	Listing running containers	89
10.4	AWS Component Diagram	93
10.5	Listing running docker containers	93
10.6	Landing page	94

List of Tables

Appendix A

Installation Guidelines

” Specifications for Android App installation:

API level (Platform version): There have been many major changes between API's in android. Food Chain Android App currently supports minimum API level 19 (android version=4.4) and highest API level 27 (android version=8.1) for device compatibility.

Device capabilities: Your android device should have following capabilities: GPS, Integrated Camera, Internet Connection.

Device Permissions: Food Chain app requires following permissions before installation: Internet, location, media storage.

Installation:

Please install 'Food Chain' android application from a shared location. By default, Android devices cannot install apps from sources other than the Google Play Store. To install apps manually, you'll need to allow your device to install apps from other sources. You may need following steps to enable manual installation of android APK.

- Please open the Settings on your android device.
- Tap "Security". it will open the Security menu.
- Scroll down and check the "Unknown sources" box. Tap "OK" to confirm that you want to enable this.
- Download Food Chain APK directly to your device from shared location.
- Navigate to the location of downloaded APK file installation.
- Tap the APK. This will immediately open the installer for the app. You can review the permissions before the app is installed, you will see a list of the
- services that it will have access to.

- Tap "Install" to install the app. The app will automatically install, and an icon for it will be added to your Home screen. You can tap the "Open" button in the installer to open it immediately." [17]

P2P Module

In order to connect the app with P2P module, it is required to run the web wallet locally using an IDE e.g. IntelliJ as mentioned in [21] . It would also be required to provide the IP of local system (where the P2P module is running) in the wallet fragment of android.

Appendix B

Contents of the CD

- A copy of all Food Chain Bitbucket Repositories:
 - food-chain-platform project containing the `fc-backend`, `fc-web`, `fc-proxy`, `docker-compose` and `env` files
 - Android Application
 - P2P Module
- A PDF file of the Master Project report
- The LaTeX source code of the Master Project report
- Slides used for the final presentation