# iBlock

# Person Verification Platform
# Software Architecture Document
# PID 6

## Version 1.0

## Mentor, Group members and contributions

**Mentor:** Dr. Thanuja Ambegoda

| **Index Number** | **Name** | **Contribution**<br>(Topics covered by each student) |
|---|---|---|
| 190239A | K. G. Akila Induranga | ● Architectural representation<br>● Architectural goals and constraints<br>● Use case realization<br>○ Sign-up<br>○ Recover account<br>○ Add new information field<br>● Logical view<br>● Process view<br>○ Sign-up<br>○ Recover account<br>○ Add new information field<br>● Data view |
| 190241X | M. I. M. Ishad | ● Use case realization<br>○ verify identity<br>● Logical view<br>● Process view<br>○ verify identity<br>● Deployment view<br>● Implementation view |
| 190242C | Kasun Isuranga | ● Introduction<br>● Use case realization<br>○ Monitor history<br>● Process view<br>○ Monitor history<br>● Size and performance<br>● Quality |

## Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 18/09/2022 | 1.0 | First Revision | K. G. Akila Induranga<br>M. I. M. Ishad<br>Kasun Isuranga |
|  |  |  |  |
|  |  |  |  |

## Table of Contents

# Software Architecture Document

## 1. Introduction

### 1.1 Purpose

This document provides a comprehensive architectural overview of the system, using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

### 1.2 Scope

The proposed system is a digital person verification platform using blockchain. The system aims to solve current issues associated with digital person verification. With blockchain, decentralized storage is provided, giving users full control over their credentials. A mobile/desktop application is used to store and manage credentials and an API is provided to integrate the system with third-party service providers.

### 1.3 Definitions, Acronyms, and Abbreviations

UI – User Interface

UX – User Experience

GUI – Graphical User Interface

API – Application Programming Interface

### 1.4 Overview

This document will provide architectural diagrams and design details of the iBlock digital person verification platform.

The following information is provided in this document.

- Architectural Representation
- Architectural Goals and Constraints
- Use case view
- Logical view
- Process View
- Deployment View
- Implementation View
- Data View
- Size and Performance

- Quality
- References

## 2. Architectural Representation

- Use Case View

  The Use Case View is represented with the UML Use Case Diagram. This view is intended for users who need an explanation of the higher-level functionality of the system such as the clients as well as the system designers.

- Logical View

  The Logical View consists of UML Class Diagram and UML State Diagrams, describing the important components of the system architecture such as classes and subsystems. This view is intended for system designers and system developers who will be needing more in-depth insight into the system.

- Process View

  The Process View consists of UML Activity Diagrams and UML Sequence Diagrams, describing the interactions between the components mentioned in the logical view. Also, this is the view where most of the methods and parameters that will be used in the system will be represented. This view is also intended for the system designers and system developers, who will need an in-depth understanding of the system.

- Deployment View

  The Deployment View is represented with the UML Deployment Diagram, describing the device that the system is using and the connections between such devices, also known as nodes. Also, this is the view where the required execution environments for the system will be represented. This view is intended for the members of the production team who are responsible for the release of the product.

## 3. Architectural Goals and Constraints

The distributed architecture of the product is considered to achieve the tamper-proofness of the system and to guarantee zero downtime whenever possible. The software requirements to develop the solution are any preferred code editor for writing and debugging, an emulator or a device for testing the mobile application, a web browser with web3 extensions or web3 support built in, and access to a private blockchain as the testing environment. Designed architecture not having a relational database setup will constrain some of the data querying functionalities and being a decentralized application may constrain the debugging of some functionalities to some extent.

# 4. Use-Case View

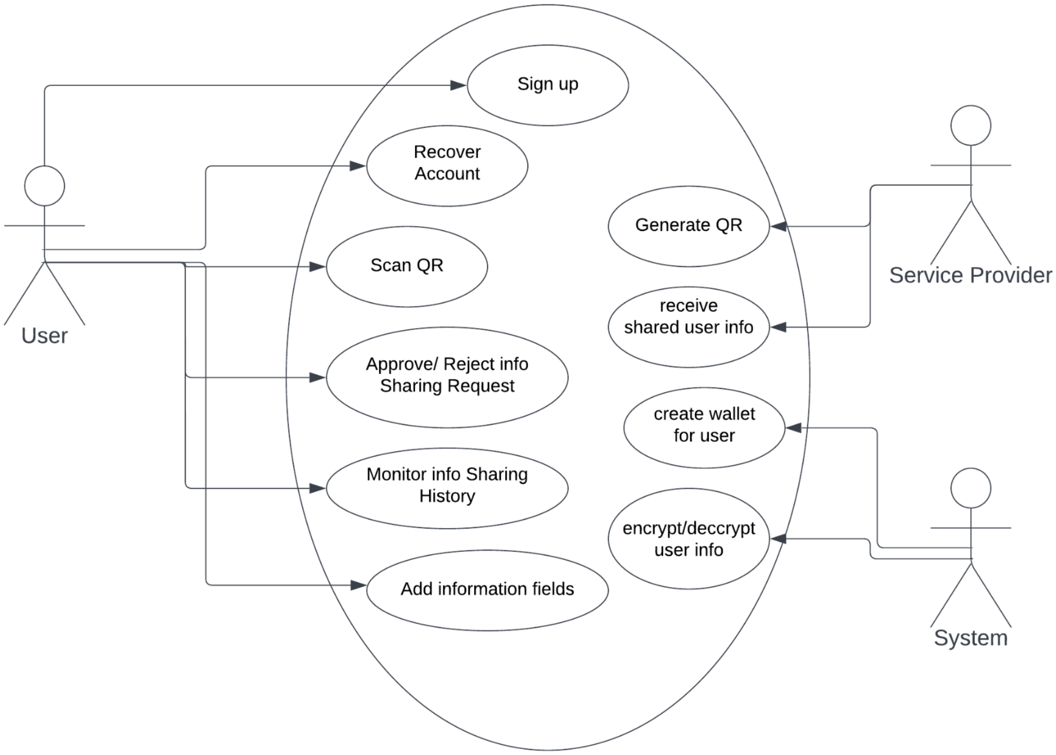


Figure 1. UML Use Case Diagram

UML Diagram for this project consists of three roles interacting with the platform performing ten use cases as shown in the diagram. A user can be any person who wants to use iBlock to verify his identity as a unique person. A service provider can be any vendor who wishes to use iBlock as a tool in their services to identify and verify people uniquely, and the role of the system is to perform the necessary use cases inside the platform.

## 4.1 Use-Case Realizations

### 4.1.1 Sign-up

| | |
|---|---|
| **Use case name** | SignUp |
| **Actor** | User |

| | |
|---|---|
| **Description** | Allows a user with the mobile application to create an account in the system and sign in. |
| **preconditions** | The mobile application has to be installed on the user's device, and the device should have a working internet connection. Also, the user is expected to have an accessible email account. |
| **Main flow** | The signup form is prompted to the user,<br>and the user enters the required details and submits,<br>The application validates the data,<br>and a wallet gets created for the user. |
| **Successful end/post condition** | Sign Up successfully, landing view of the home. |
| **Fail end/post condition** | Sign Up failed, back to the sign in view. |
| **Extensions** | Encrypt user data,<br>Create Wallet |

### 4.1.2 Recover Account

| | |
|---|---|
| **Use case name** | Recover Account |
| **Actor** | User |
| **Description** | Allows a user with the recovery phrase to restore the wallet and recover the account |
| **preconditions** | The mobile application has to be installed on the user's device, and the device should have a working internet connection. Also the user is expected to have the secret recovery phrase presented when the account is created. |
| **Main flow** | The recovery form is prompted to the user,<br>User enters the secret recovery phrase and submits,<br>The application validates the phrase,<br>A wallet gets restored for the user. |
| **Successful end/post condition** | Restored successfully, landing view of the home. |
| **Fail end/post condition** | Restore failed, back to the sign in view. |
| **Extensions** | Restore Wallet |

4.1.3. Verify Identity

| | |
|---|---|
| **Use case name** | Verify Identity |
| **Actor** | User, Service Provider |
| **Description** | Allows service providers to get the user's identity verified. |
| **preconditions** | This main use case consists of the following sub-use cases.<br>● Service provider generating the QR code.<br>● User scans the QR code provided by the service provider<br>● User approves verification information to be shared with service provider |
| **Main flow** | If a user needs to verify identity to a service provider to get a service done.<br>First, the user needs to scan the QR code provided by the service provider.<br>Then, the application shows what is the information requested by the service provider.<br>Finally, the user needs to approve or reject the information-sharing transaction. |
| **Successful end/post condition** | Identity verified and can proceed with next steps. |
| **Fail end/post condition** | Identity verification failed and cannot proceed with the next steps. |
| **Extensions** | N/A |

4.1.4. Monitor Sharing History

| | |
|---|---|
| **Use case name** | Monitor Sharing History |
| **Actor** | User |
| **Description** | Allows a user with a created wallet to view the approvals and denials he has made to different service providers on a single click. |

| | |
|---|---|
| **preconditions** | The mobile application has to be installed on the user's device, and the device should have a working internet connection. Also the user is expected to be logged in to the wallet. |
| **Main flow** | User requests to view his information sharing history,<br>The application calls the smart contract to get the transactions under the user's wallet address,<br>Application receives the transaction list, processes it and returns a map of information sharing history. |
| **Successful end/post condition** | View of all the transactions of information sharing. |
| **Fail end/post condition** | Fetching information history failed, back to home view |
| **Extensions** | N/A |

### 4.1.5. Add New Information Field

| | |
|---|---|
| **Use case name** | Add Information Fields |
| **Actor** | User |
| **Description** | Allows a user with the created wallet to add a new personal information field as preferred. |
| **preconditions** | The mobile application has to be installed on the user's device, and the device should have a working internet connection. Also the user is expected to be logged in to the wallet. |
| **Main flow** | The add information form is prompted to the user,<br>User enters the new information (key - value pair) and submits,<br>The application validates the information,<br>New information field gets stored in the wallet. |
| **Successful end/post condition** | Information added successfully, landing view of the home. |
| **Fail end/post condition** | Adding information failed, back to home view |
| **Extensions** | N/A |

# 5. Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.

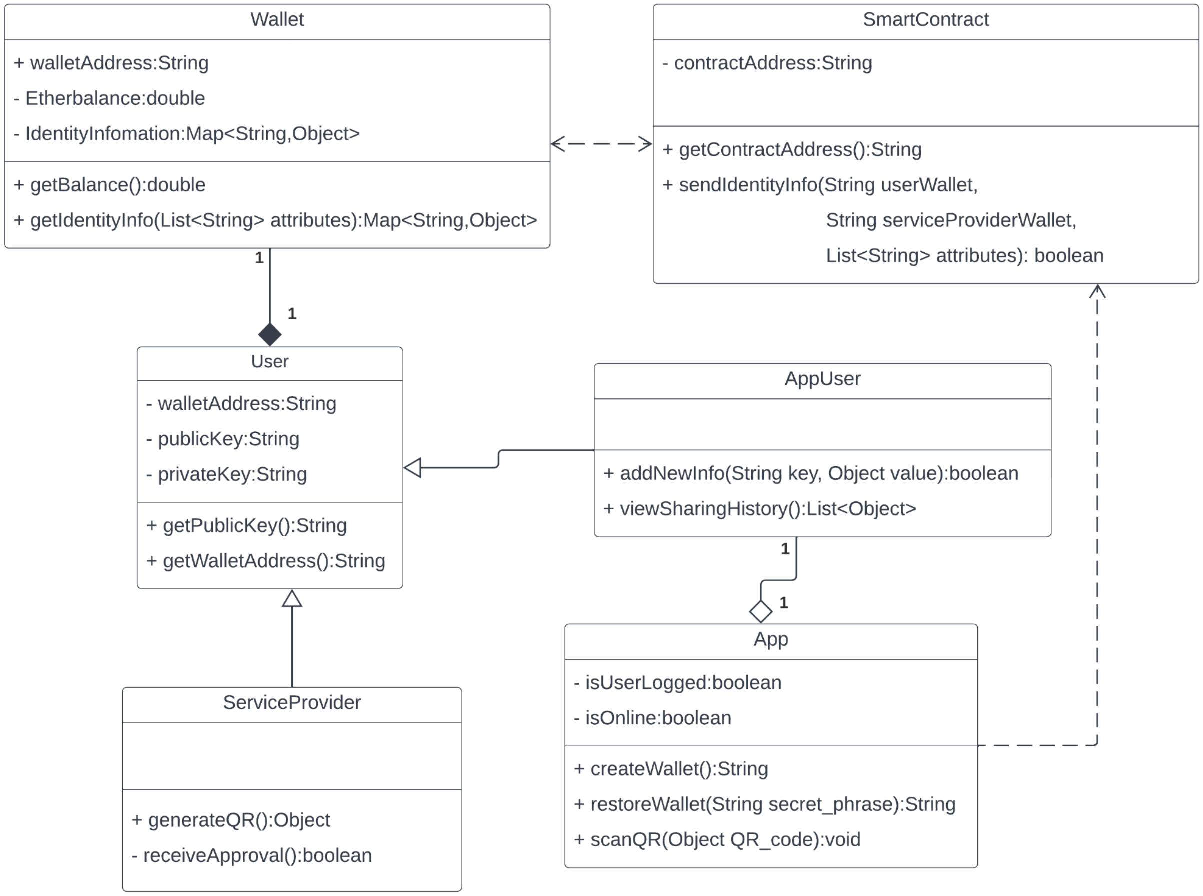
## 5.1 Overview

When decomposing the system following are the main classes identified in the system.

1. User
2. App user ( Inherited by User )
3. Service provider ( Inherited by User )
4. App
5. Wallet
6. Smart Contract

Here, app users and service providers are identified to have common attributes. It is the reason for creating a superclass called user.

## 5.2 Architecturally Significant Design Packages



Wallet
+ walletAddress:String
- Etherbalance:double
- IdentityInfomation:Map<String,Object>
+ getBalance():double
+ getIdentityInfo(List<String> attributes):Map<String,Object>
SmartContract
- contractAddress:String
+ getContractAddress():String
+ sendIdentityInfo(String userWallet,
String serviceProviderWallet,
List<String> attributes): boolean
1
1
User
- walletAddress:String
- publicKey:String
- privateKey:String
+ getPublicKey():String
+ getWalletAddress():String
AppUser
+ addNewInfo(String key, Object value):boolean
+ viewSharingHistory():List<Object>
1
1
App
- isUserLogged:boolean
- isOnline:boolean
+ createWallet():String
+ restoreWallet(String secret_phrase):String
+ scanQR(Object QR_code):void
ServiceProvider
+ generateQR():Object
- receiveApproval():boolean

Figure 2. UML Class Diagram

# 6. Process View

## 6.1. Sign-up

### 6.1.1. Activity Diagram (Sign-up)

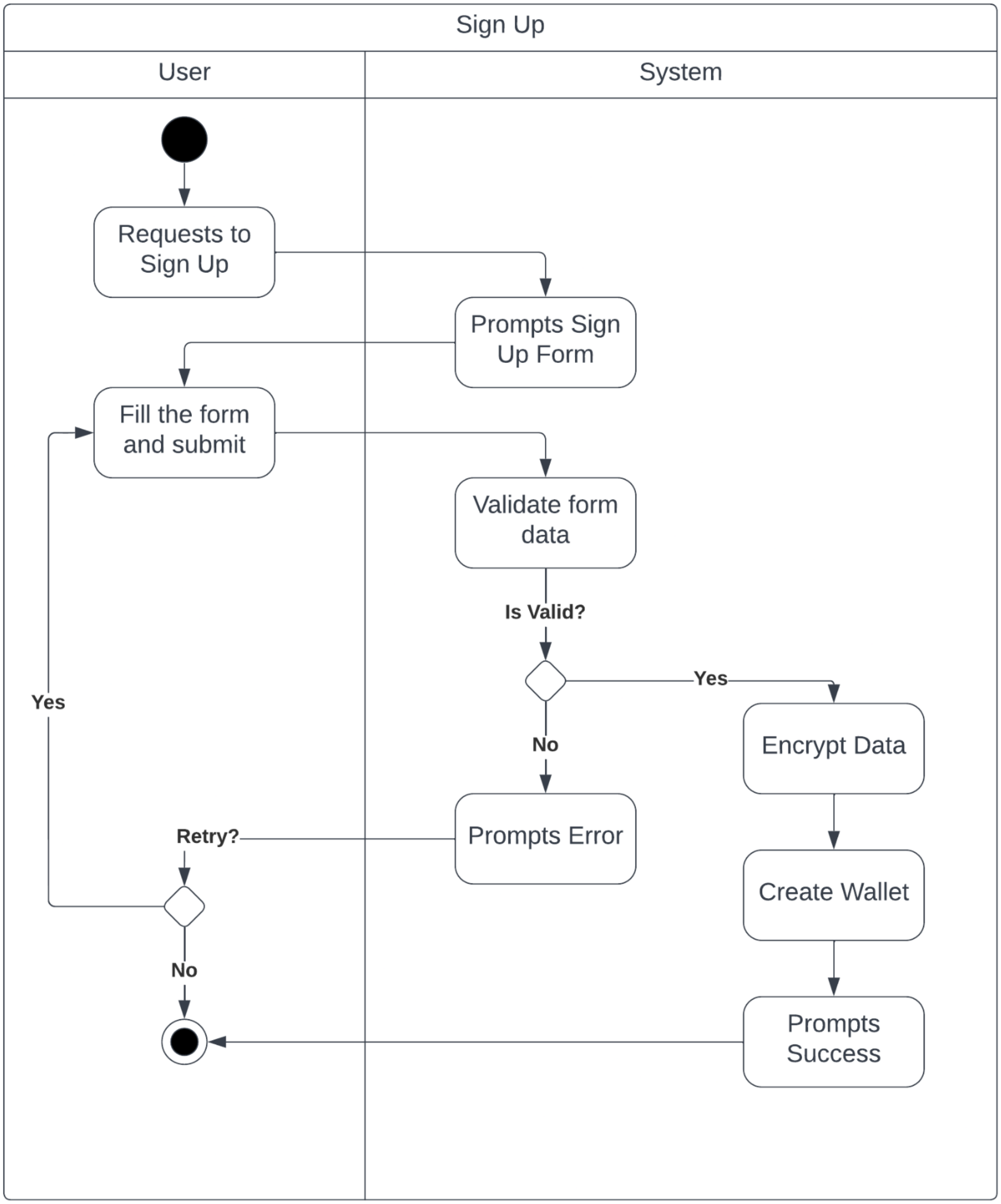


Figure 3. UML Activity Diagram for Sign Up

### 6.1.2. Sequence Diagram (Sign-up)

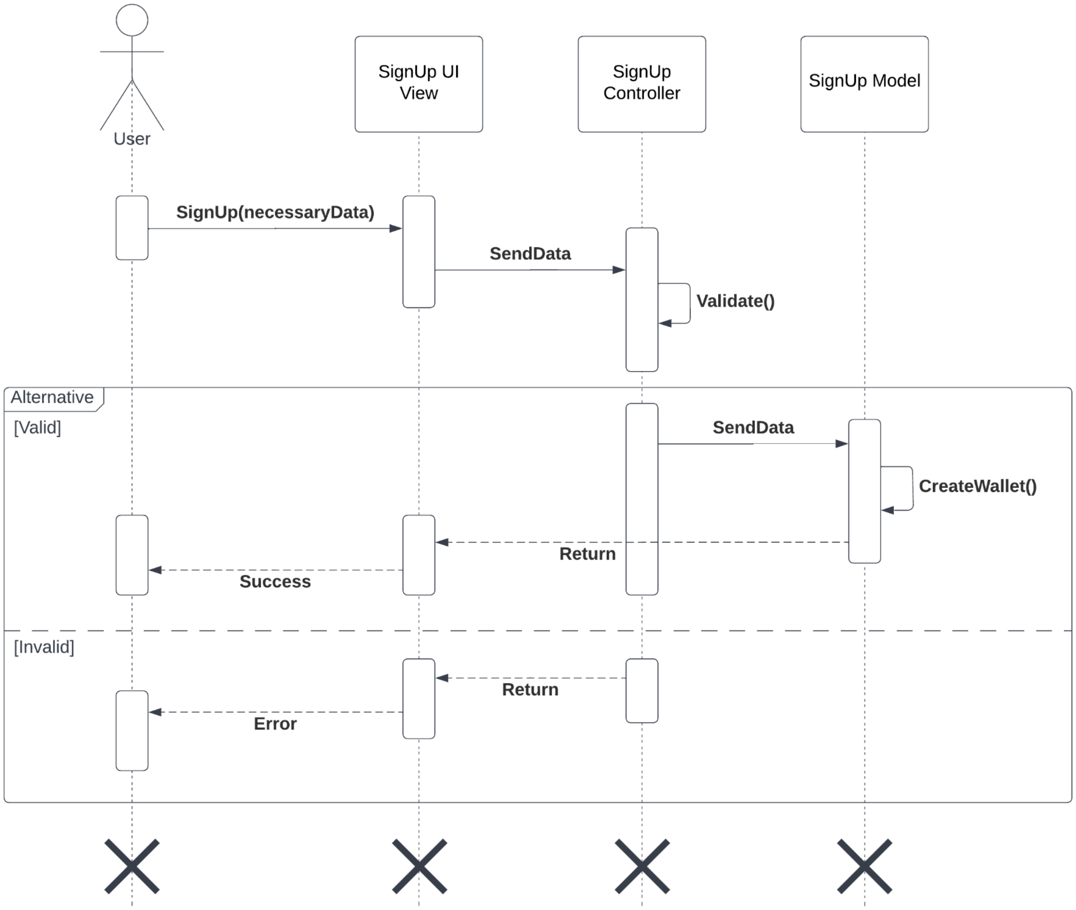


Figure 4. UML Sequence Diagram for Sign Up

### 6.2. Recover Account

6.2.1. Activity Diagram (Recover Account)

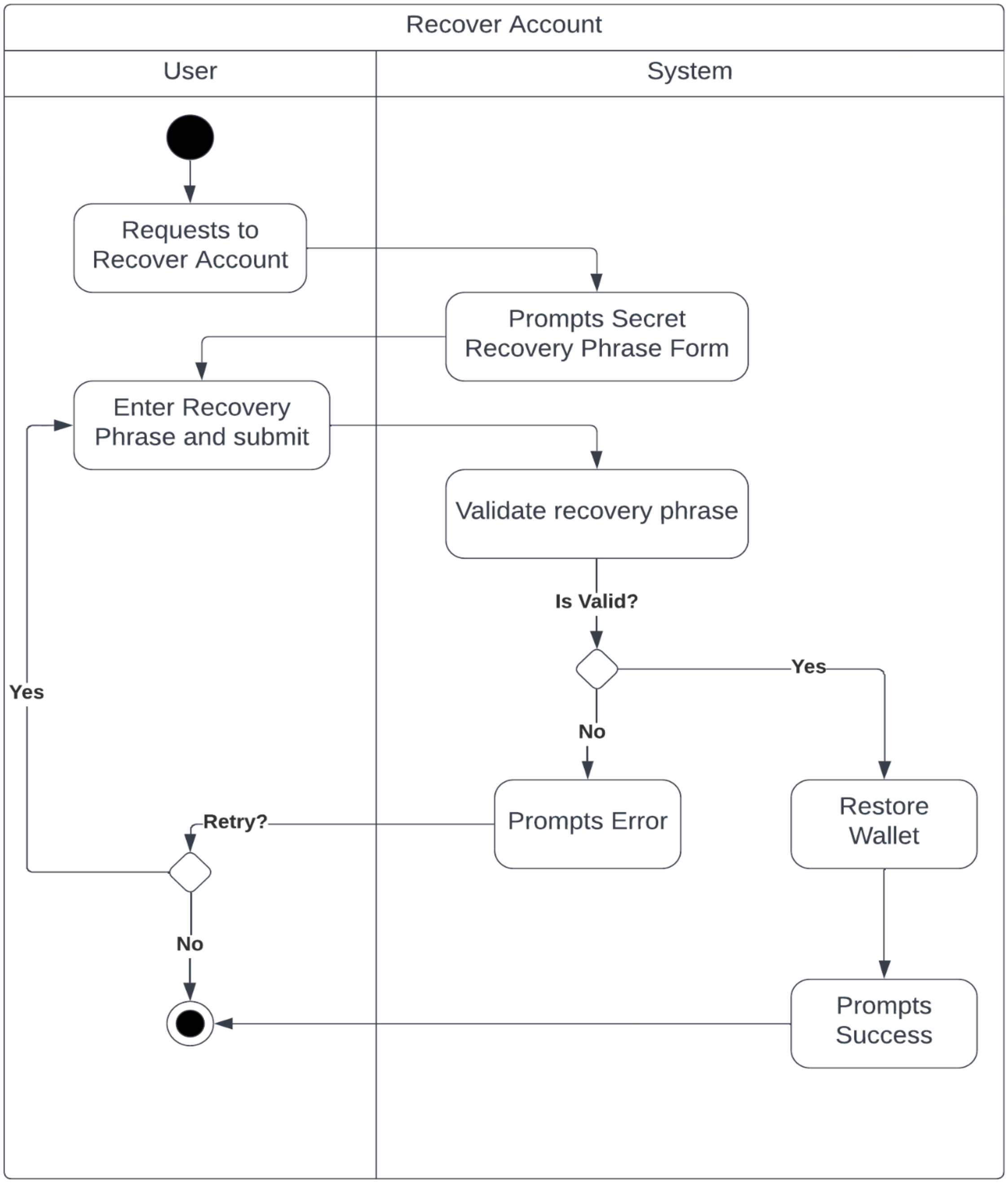


Figure 5. UML Activity Diagram for Recover Account

### 6.2.2. Sequence Diagram (Recover Account)

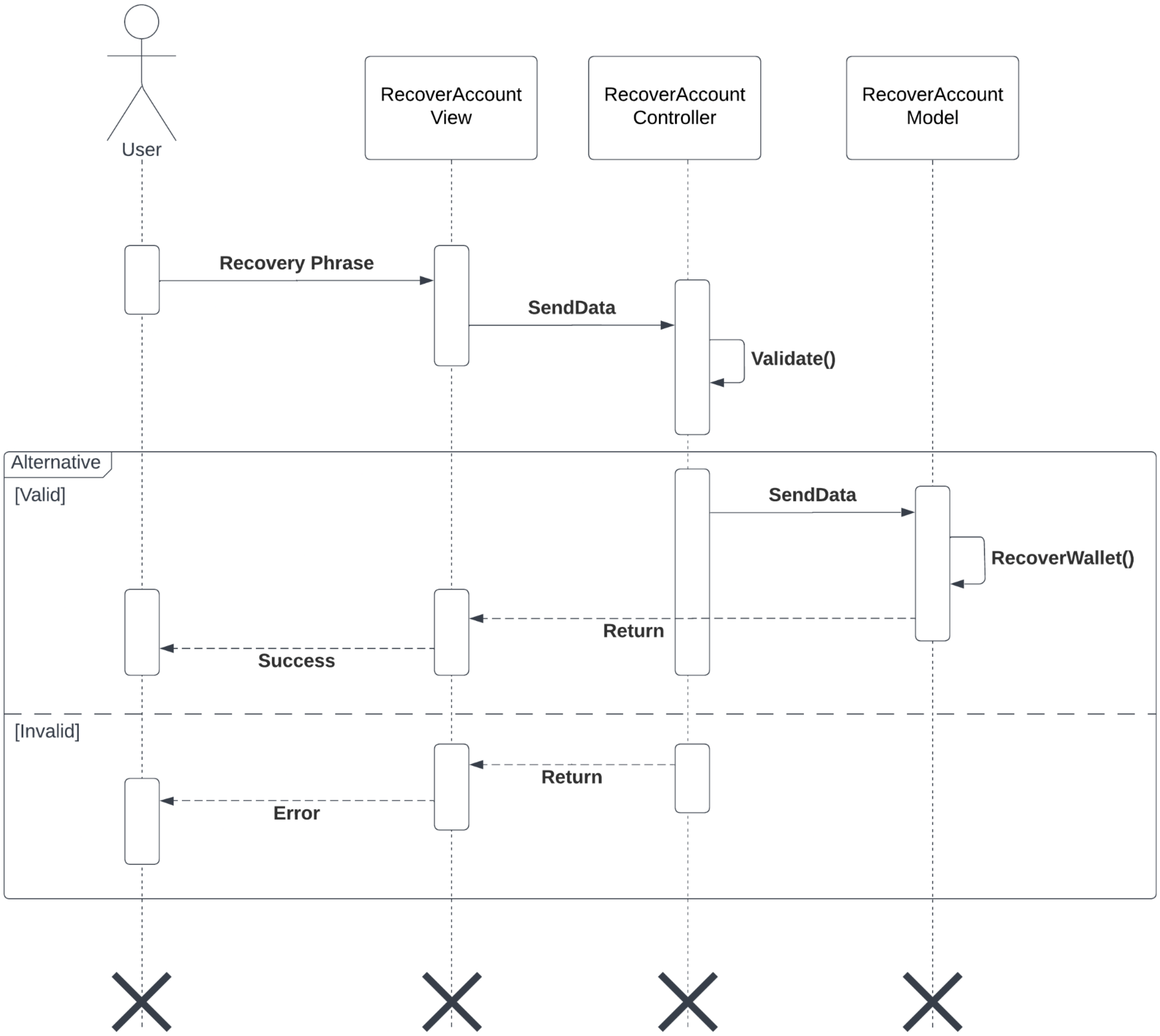


Figure 6. UML Sequence Diagram for Recover Account

## 6.3. Verify Identity

6.3.1. Activity Diagram (Verify Identity)

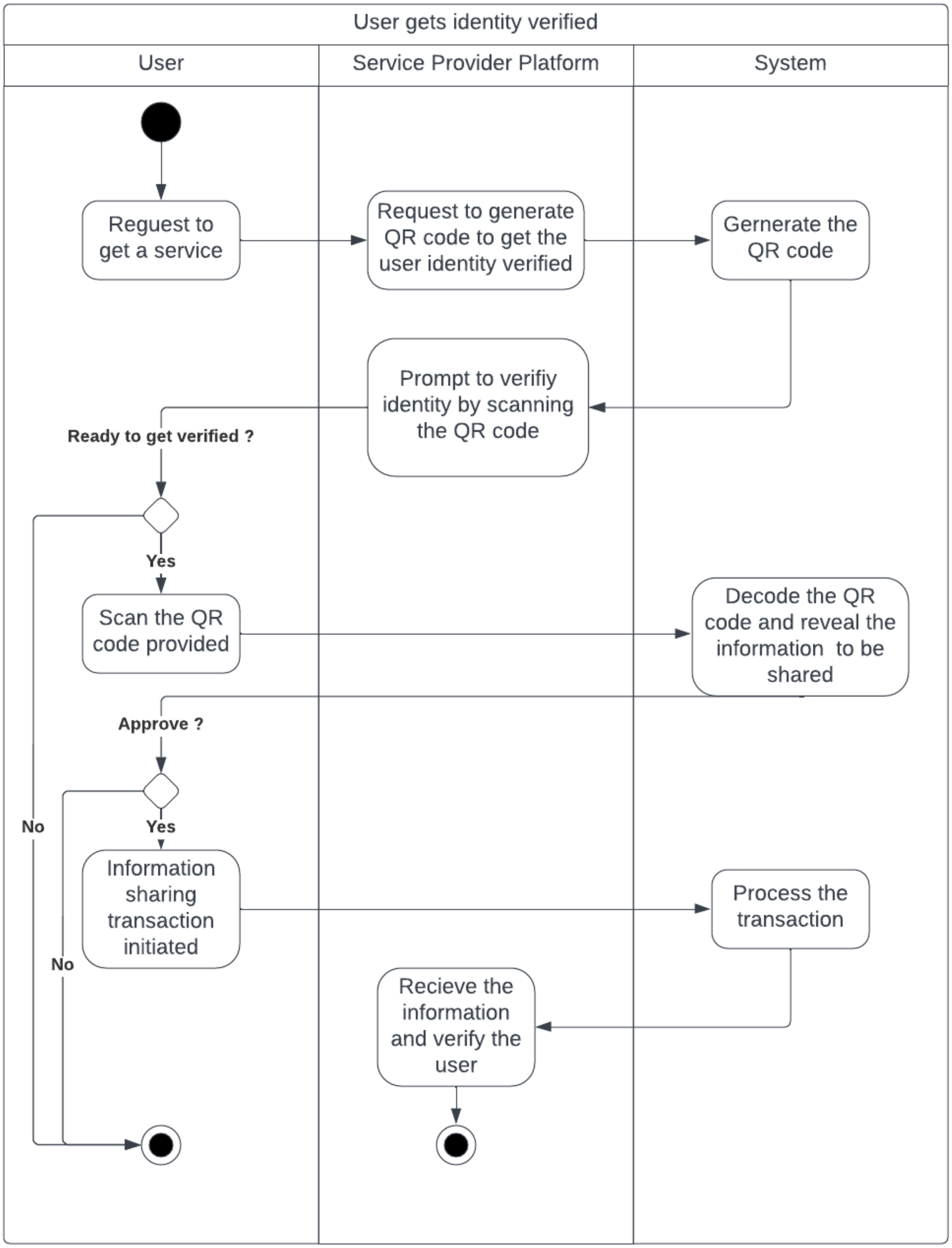

User gets identity verified
User
Service Provider Platform
System
Reguest to get a service
Request to generate QR code to get the user identity verified
Gernerate the QR code
Prompt to verifiy identity by scanning the QR code
Ready to get verified ?
Yes
No
Scan the QR code provided
Decode the QR code and reveal the information to be shared
Approve ?
Yes
No
Information sharing transaction initiated
Process the transaction
Recieve the information and verify the user

Figure 7. UML Activity Diagram for Verify Identity

### 6.3.2. Sequence Diagram (Verify Identity)

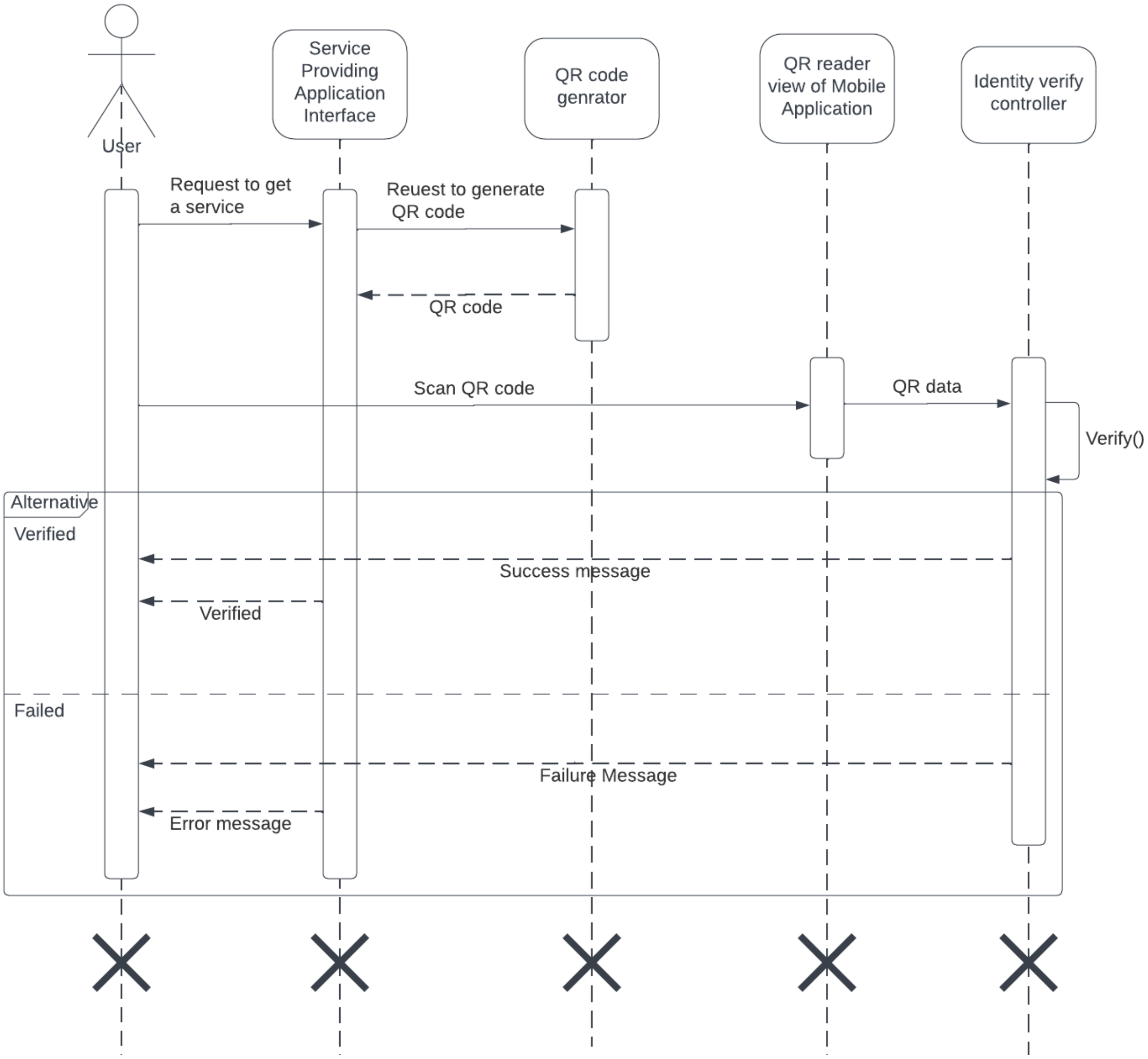
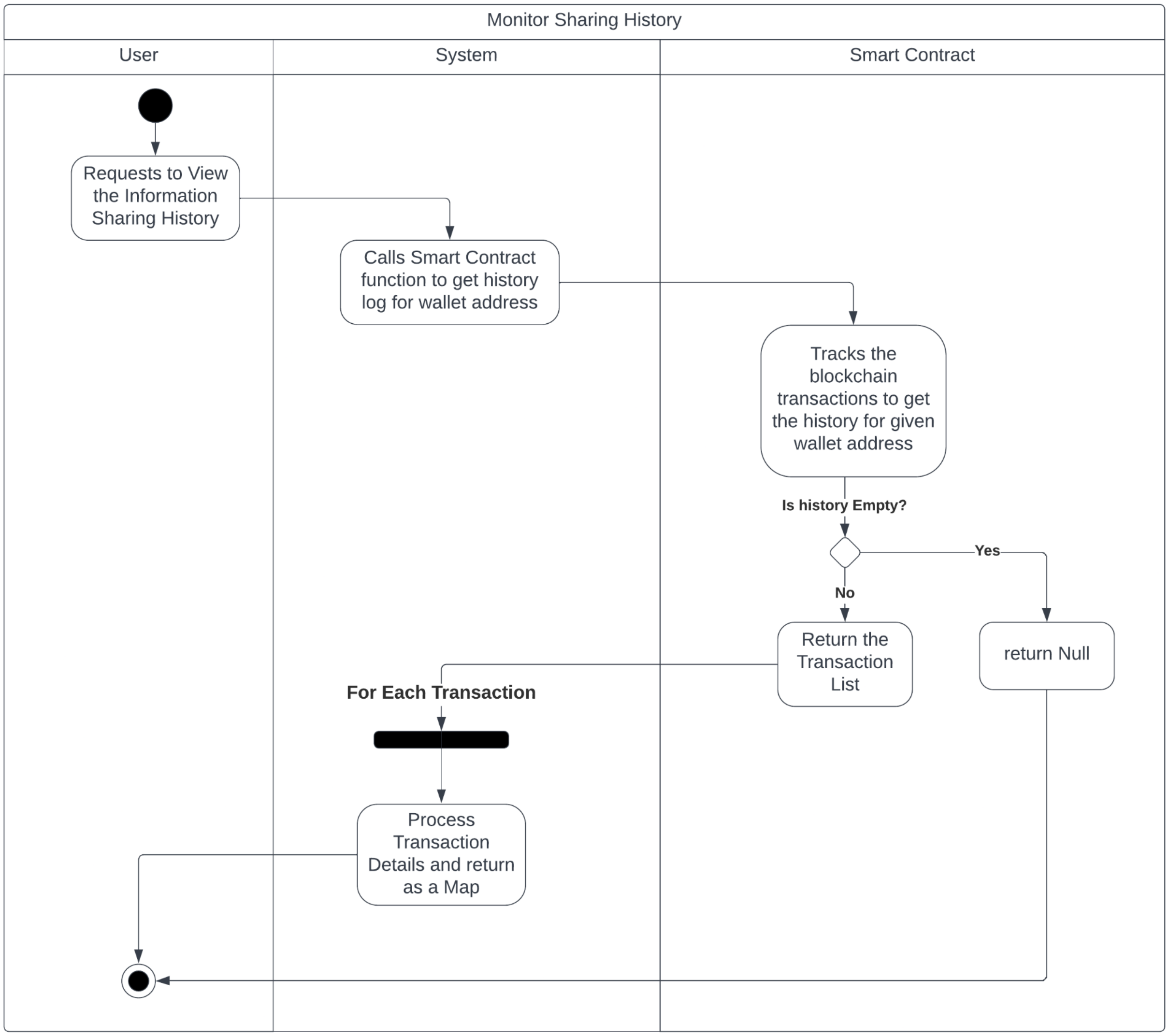


Figure 8. UML Sequence Diagram for Verify Identity

### 6.4. Monitor Sharing History

#### 6.4.1. Activity Diagram (Monitor Sharing History)



Monitor Sharing History
User
System
Smart Contract
Requests to View the Information Sharing History
Calls Smart Contract function to get history log for wallet address
Tracks the blockchain transactions to get the history for given wallet address
Is history Empty?
Yes
No
Return the Transaction List
return Null
For Each Transaction
Process Transaction Details and return as a Map

Figure 9. UML Activity Diagram for Monitor Sharing History

### 6.4.2. Sequence Diagram (Monitor Sharing History)

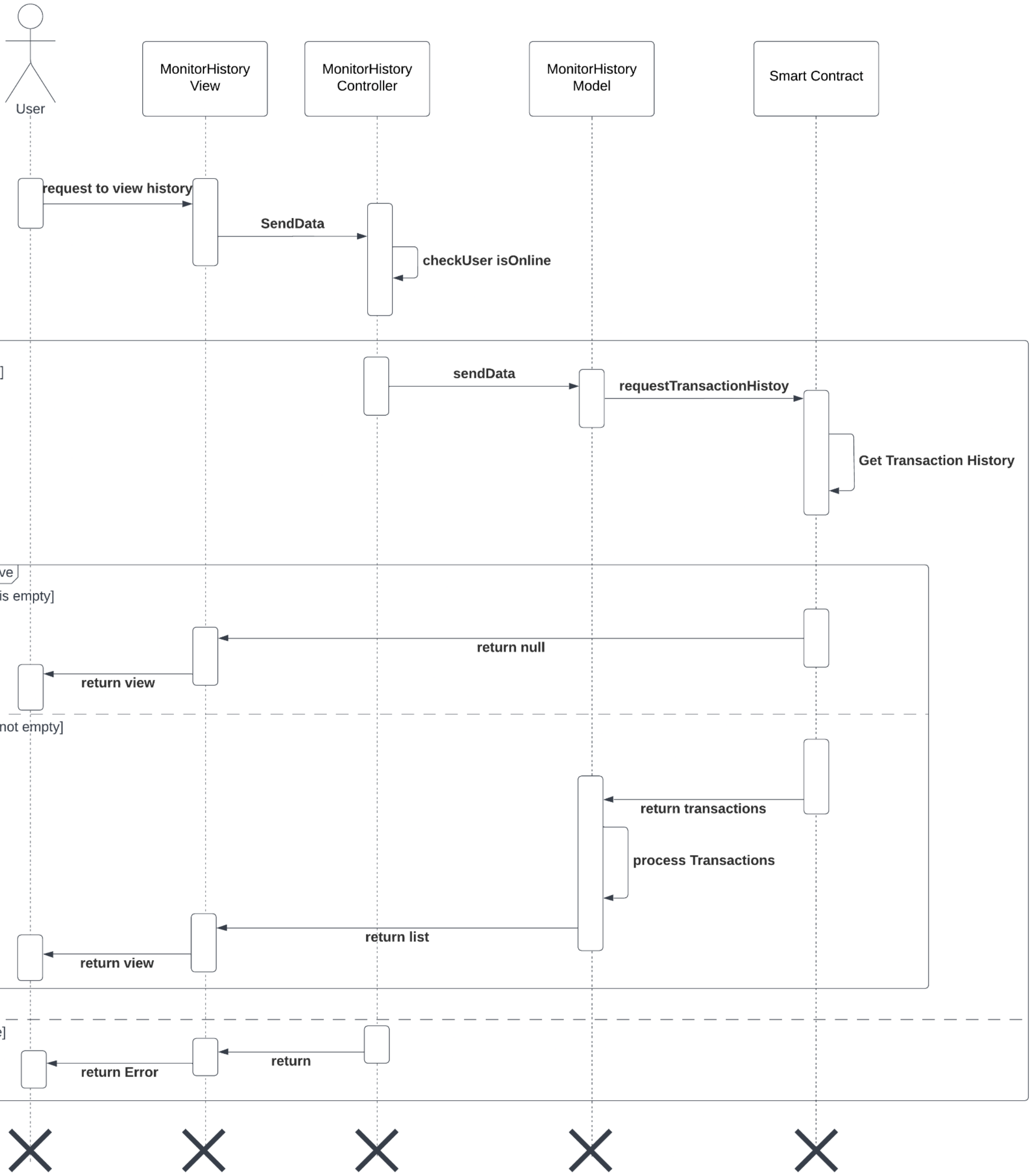
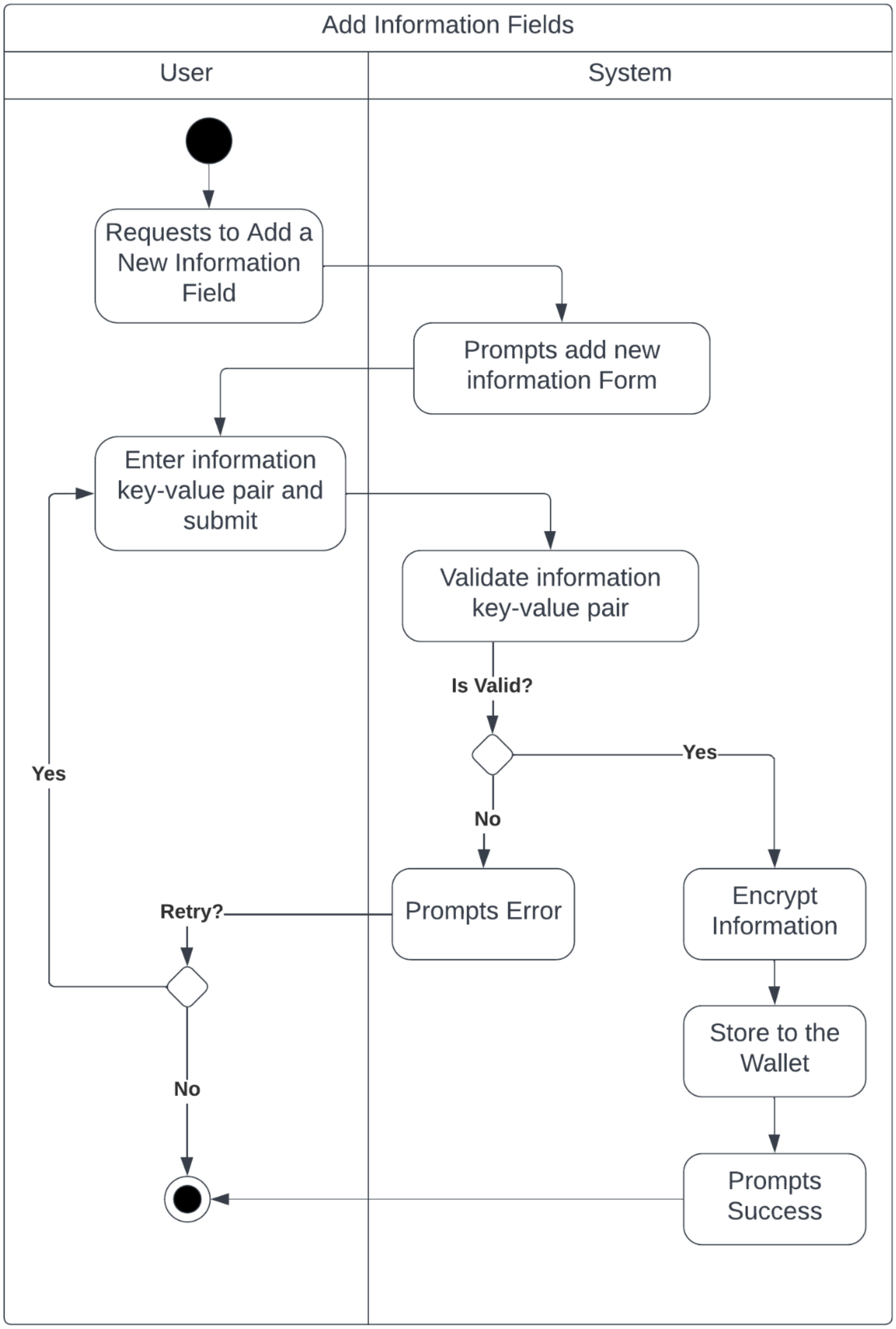


Figure 10. UML Sequence Diagram for Monitor Sharing History

## 6.5. Add New Information Field

### 6.5.1. Activity Diagram (Add New Information Field)



Add Information Fields
User
System
Requests to Add a New Information Field
Prompts add new information Form
Enter information key-value pair and submit
Validate information key-value pair
Is Valid?
Yes
No
Prompts Error
Encrypt Information
Store to the Wallet
Prompts Success
Retry?
Yes
No

Figure 11. UML Activity Diagram for Add New Information Field

### 6.5.2. Sequence Diagram (Add New Information Field)

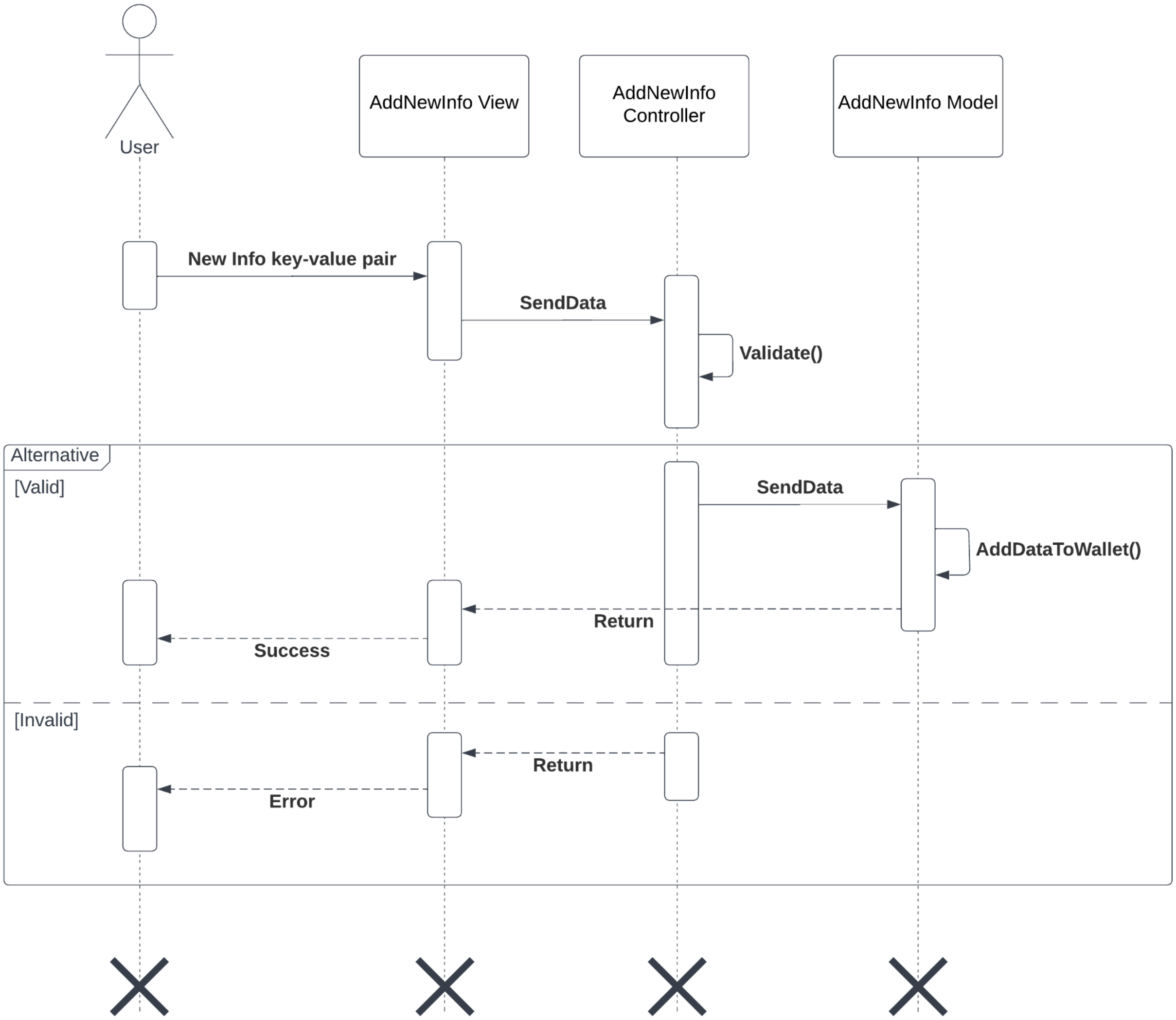


Figure 12. UML Sequence Diagram for Add New Information Field

# 7. Deployment View

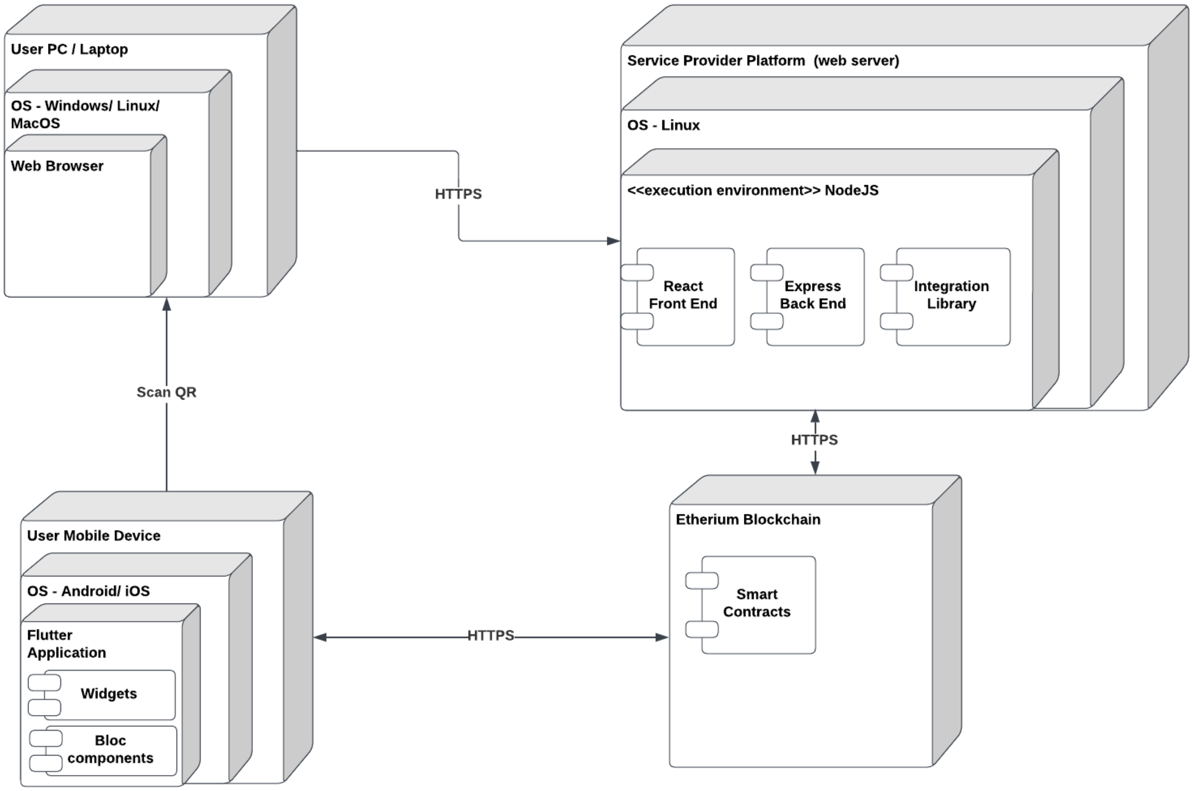


Figure 13. UML Deployment Diagram

iBlock will be deployed as a mobile application that can be installed on both android and iOS devices for users to verify their identity. The smart contracts deal with both users' and service providers' wallets will be deployed on the Ethereum Blockchain (a testent), and the integration API will be provided to service vendors deployed on Github.

# 8. Implementation View

## 8.1. Overview

Implementation of the iBlock project is divided into the following components.

1. iBlock application
2. Demo application and Integration library

## 8.2. Layers

### 8.2.1. Implementation view of iBlock application

Bloc design pattern[4] is proposed to use for the implementation of the application. This design pattern separates the user interface from the business logic of the application. The business logic further makes use of events to make transitions between application states. The view is mapped with the state of the business logic

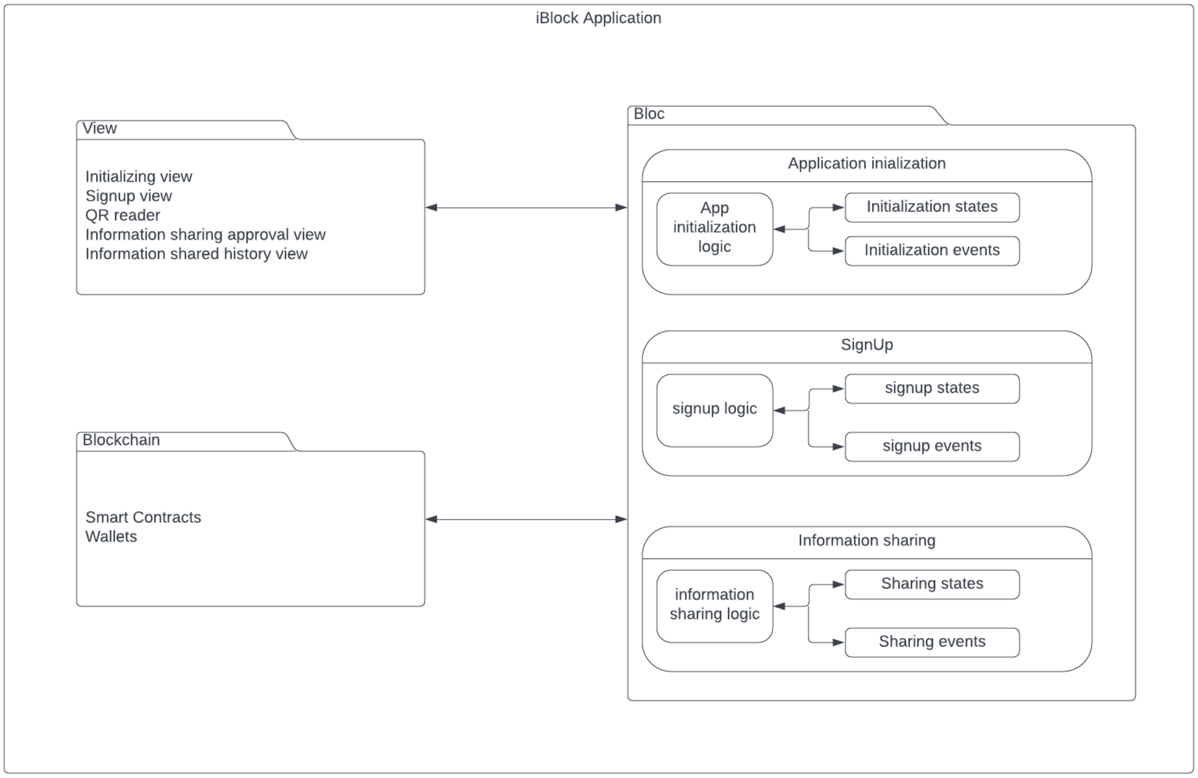


Figure 14. UML Package Diagram for Mobile Application

### 8.2.2. Implementation view of demo application and integration library

Proposed to develop the demo application as front-end and back-end. The front end is to be built with react and the back end with expressJS. The back-end will interact with the blockchain.

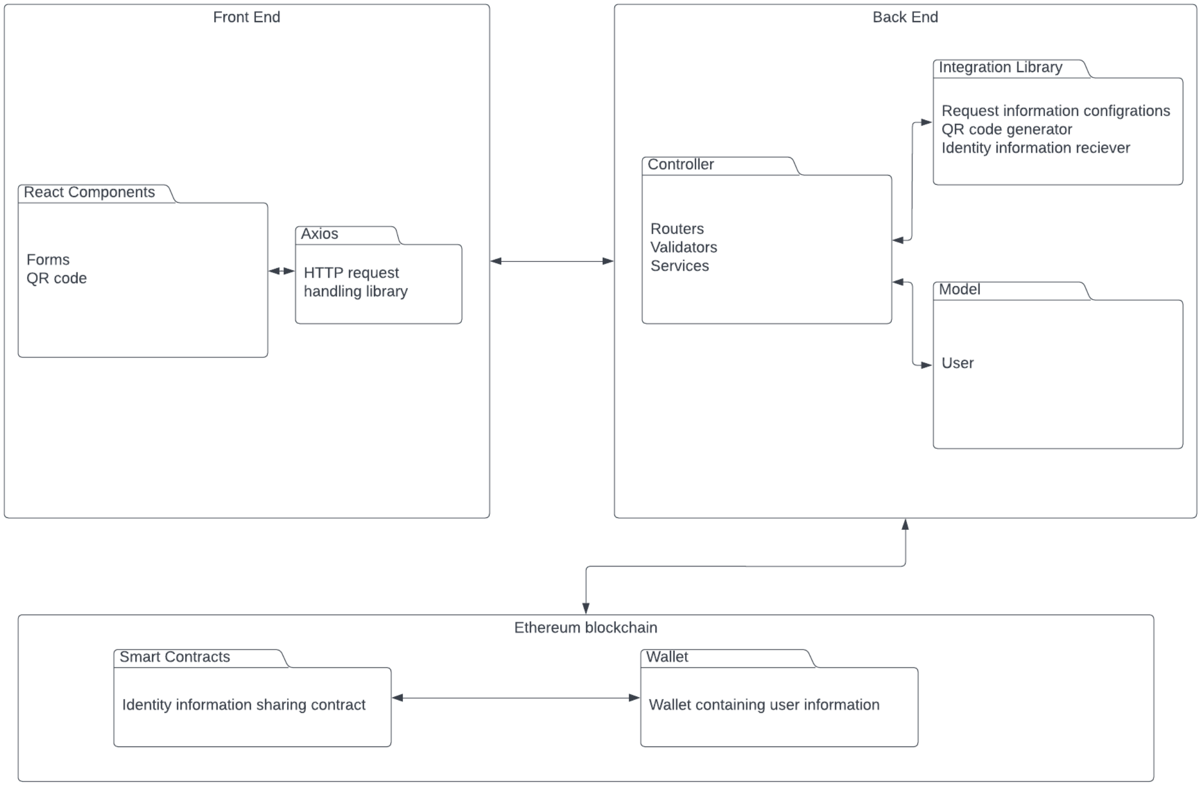


Figure 15. UML Package Diagram for Service Provider and Integration API

## 9. Data View

Being a decentralized application, iBlock is not storing users' personal information on any kind of centralized database server. All the information is stored in the user's wallet which can only be accessed by the owner itself and is allowed to be shared by the owner only.

## 10. Size and Performance

### Security

As this software system is related to confidential information and security sensitive activities, a very high priority should be placed on security.

### User Experience

The proposed software solution intends to provide useful services to normal people.

Therefore a high emphasis needs to be on user experience.

### Processing latencies

Minimizing processing latencies can provide a smoother experience for the users.

### Concurrency

A major part of the software system can benefit from concurrent operations, as in Smart Contracts and API. Improving the performance by using a maximum number of concurrent operations should be prioritized.

## 11. Quality

### Usability

A simple UI with steps taken to improve the user experience will be used in the iBlock Application Program. This application is cross platform which further enhances the usability.

The software interface of the API will be simple providing easier integration with third-party service providers web interfaces.

### Reliability

The modular architecture used in the system will provide better maintainability and a simpler design. These characteristics will provide a reliable end-product.

### Scalability

As the system is modularized, each module can be improved separately which will make the system highly scalable.

## 12. References

[1] Lucid chart (https://lucid.app/)

[2] Ethereum (https://ethereum.org/en/)

[3] Flutter (https://flutter.dev/)

[4] Bloc package (https://pub.dev/packages/bloc)