I3301
CryptoBank

55825 Jaisson Abi Saad, 56053 Elie Aoun

Summary

Contents

1- Introduction

.During the pandemic we saw a rise in the use of cryptocurrencies and many companies such as PayPal adopted the trade of Bitcoin which is one of the principals and most used cryptocurrencies. These events procured a quick rise in the price of various currencies and especially Bitcoin. Various apps exist that acts as a digital wallet to keep, trade, sometimes buy and sell cryptocurrencies. The underlying mechanism of these digital currencies is blockchain which is a complex and specific type of database that acts in a very different way compared to a traditional one. Therefore, we decided to build an android app using ANDROID STUDIO that embodies a modern UI which represents a simulation of what a crypto wallet app would look like with an underlying backend written in PHP connected to a traditional database MYSQL to perform various operations for the user in order to provide a good user experience. In addition, we wanted to provide a quick and easy way to add new currencies for the app to support. How can such thing be implemented? Do we hardcode it every time we want to add a new one? After many thoughts we decided to build a simple and straightforward web interface using HTML, CSS, JS, and PHP for the admin to add new currencies, ban users in case of complaints and send emails to users who requested customer support. In order to support many currencies everything in our app must be dynamic, ranging from adding a currency to updating in real time the price of all the supported cryptos therefore to achieve that we used APIs such as the binance API to provide real time prices and market statistics.

2- Analysis

2-1- Business Rules

CryptoBank can have two entries:

- User access (Android).
- Administrator access (Web).

In the user module:

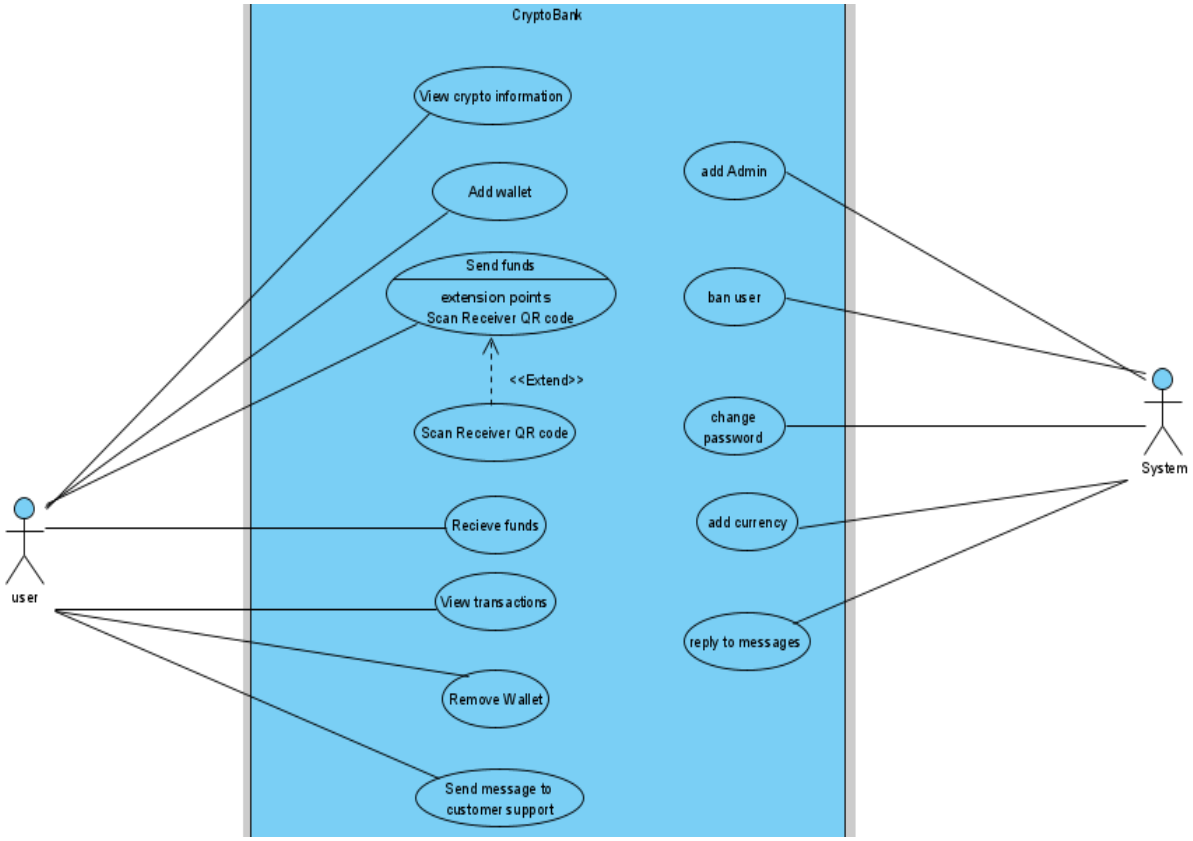
- Each User must register before using the app by entering a valid email, password, and date of birth.
- Users can create wallets. Each wallet has a type (Cryptocurrency), a creation date and a default balance of 0.01.(for the simulation purpose)
- Once the wallet is created, users can now send and receive funds.
- After each send or receive, a transaction is saved. A transaction contains the amount (in crypto and USD), a date, the wallet addresses of the sender and receiver.

For more information about something, users can contact customers support by sending a message. They will have to wait 24hours before sending another one.

In the administrator module:

- An admin can add/remove another admin (user with administrator privileges).
- Ban users in case of any suspicious activity.
- Insert/remove cryptocurrencies. These currencies will be the supported wallet types in the app. A currency has a name, symbol, and logo.
- Reply to users' messages. Only the non-replied messages will be displayed to the admins.

2-2- Use Case



2-2-1- Add Wallet Use Case:

Scenario 1: Wallet is successfully added.

- User chooses the Add Wallet action.
- User selects the wallet type.
- Check if the wallet already exists.
- Create wallet.

Scenario 2: Wallet addition fails since the wallet already exists.

- User chooses the Add Wallet action.
- User chooses the wallet type.
- Check if the wallet already exists.
- Return error message to the user.

2-2-2- Remove Wallet Use Case:

Scenario 1: Wallet is successfully removed.

- User chooses the Remove Wallet action.
- Check if the wallet balance is 0.
- Remove Wallet.

Scenario 2: Wallet is successfully removed after user's confirmation that balance will be lost.

- User chooses the Remove Wallet action.
- Check if the wallet balance is 0.
- Display warning to the user that balance will be lost.
- User agrees.
- Remove wallet.

Scenario 3: Wallet remove fails after the user cancels the process.

- User chooses the Remove Wallet action.
- Check if the wallet balance is 0.
- Display warning to the user that balance will be lost.
- User cancels the process.

2-2-3- Send message to customer support Use Case:

Scenario 1: Message is successfully sent.

- User enters the message.
- User chooses the send message action.
- Message sent.
- Display warning that the user must wait 24 hours before sending a another one.

Scenario 2: Cannot send the message since the user did not wait 24 hours after sending the last one.

- User cannot enter a message.

2-2-4- Send funds Use Case:

Scenario 1: The Funds are successfully sent.

- User chooses the send funds action.
- User selects the wallet from which he wants to send.
- -User enters the amount.
- -User enters/scans the address.
- -Check if the wallet has sufficient balance.
- -Check if the address is valid.
- -Send funds to receiver.

Scenario 2: Sending fails since the wallet balance is insufficient.

- User chooses the send funds action.

- User selects the wallet from which he wants to send.
- User enters the amount.
- User enters/scans the address.
- Check if the wallet balance is sufficient.
- Return error message to the user.

Scenario 3: Sending fails since the receiver address is not valid.
- User chooses the send funds action.
- User selects the wallet from which he wants to send.
- User enters the amount.
- User enters/scans the address.
- Check if the wallet has sufficient balance.
- Check if the address is valid.
- Return error message to the user.

2-2-5- Add Admin Use Case:

Scenario 1: Admin successfully added.
- Admin chooses the Add Admin action.
- Admin enters the name, email, password, and date of birth.
- Check if the fields are valid.
- Check if the admin already exists.
- Add admin.

Scenario 2: Addition fails since the admin already exists.
- Admin chooses the Add Admin action.
- Admin enters the name, email, password, and date of birth.
- Check if the fields are valid.
- Check if the admin already exists.
- Return error message.

Scenario 3: Addition fails since one or more fields are not valid.
- Admin chooses the Add Admin action.
- Admin enters the name, email, password, and date of birth.
- Check if the fields are valid.
- Return error message.

2-2-6- Change password Use Case:

Scenario 1: Password successfully changed.

- Admin chooses the change password action.
- Admin enters the current password.
- Admin enters the new password.
- Check if the current password exists.
- Check if the new password is valid.
- Change password.

Scenario 2: Password change fails since the password does not exist.

- Admin chooses the change password action.
- Admin enters the current password.
- Admin enters the new password.
- Check if the current password exists.
- Return error message.

Scenario 3: Password change fails since the new password is not valid.

- Admin chooses the change password action.
- Admin enters the current password.
- Admin enters the new password.
- Check if the current password exists.
- Check if the new password is valid.
- Return error message.

2-2-7- Add Currency Use Case:

Scenario 1: Currency successfully added.

- Admin chooses the Add Currency action.
- Admin enters the name, logo, and description.
- Check if the fields are valid.
- Check if the currency already exists.
- Add currency.

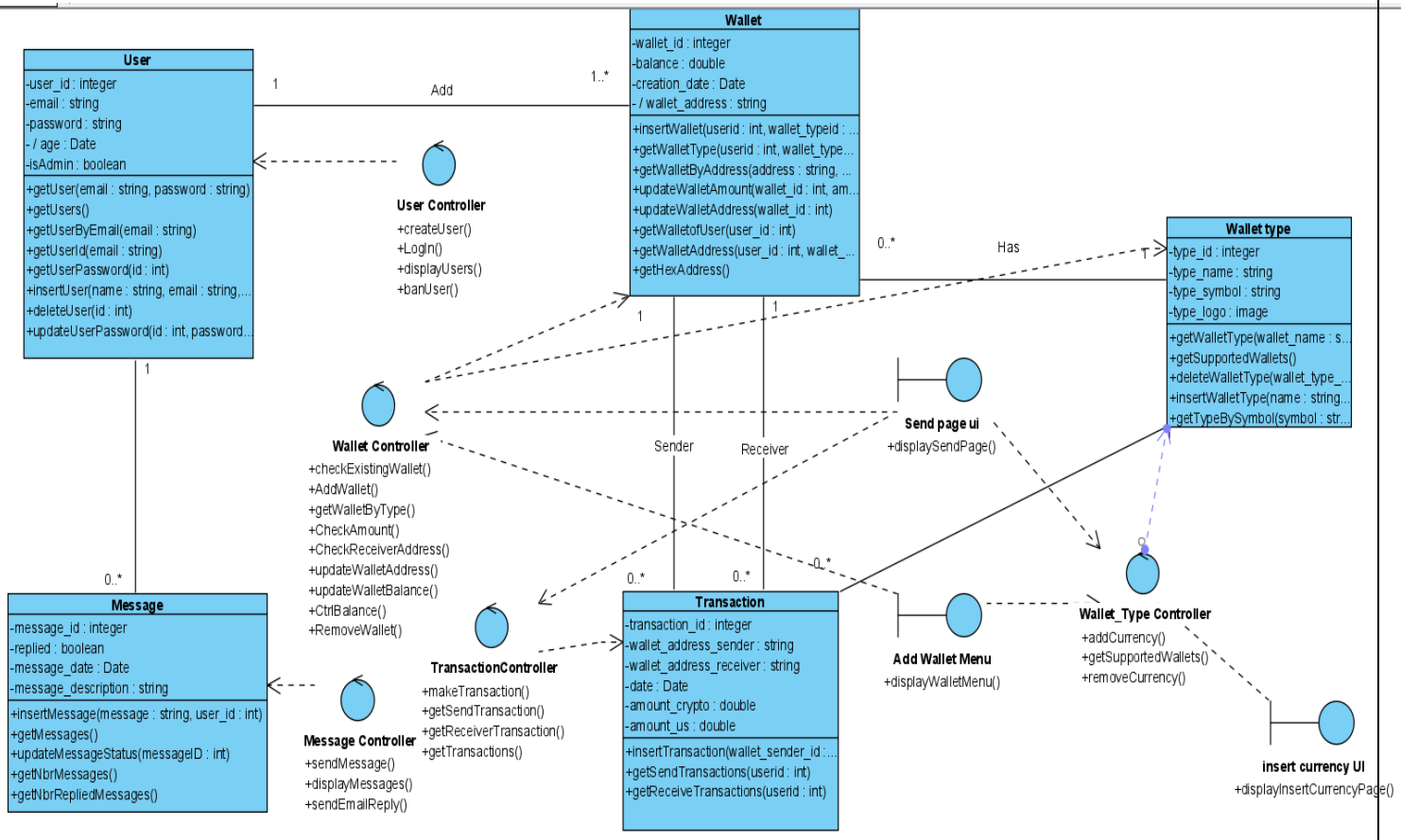
Scenario 2: Addition fails since the currency already exists.

- Admin chooses the Add Currency action.
- Admin enters the name, logo, and description.
- Check if the fields are valid.
- Check if the currency already exists.
- Return error message.

Scenario 3: Addition fails since one or more fields are not valid.

- Admin chooses the Add Currency action.
- Admin enters the name, logo, and description.
- Check if the fields are valid.
- Return error message.

2-3- Class diagram

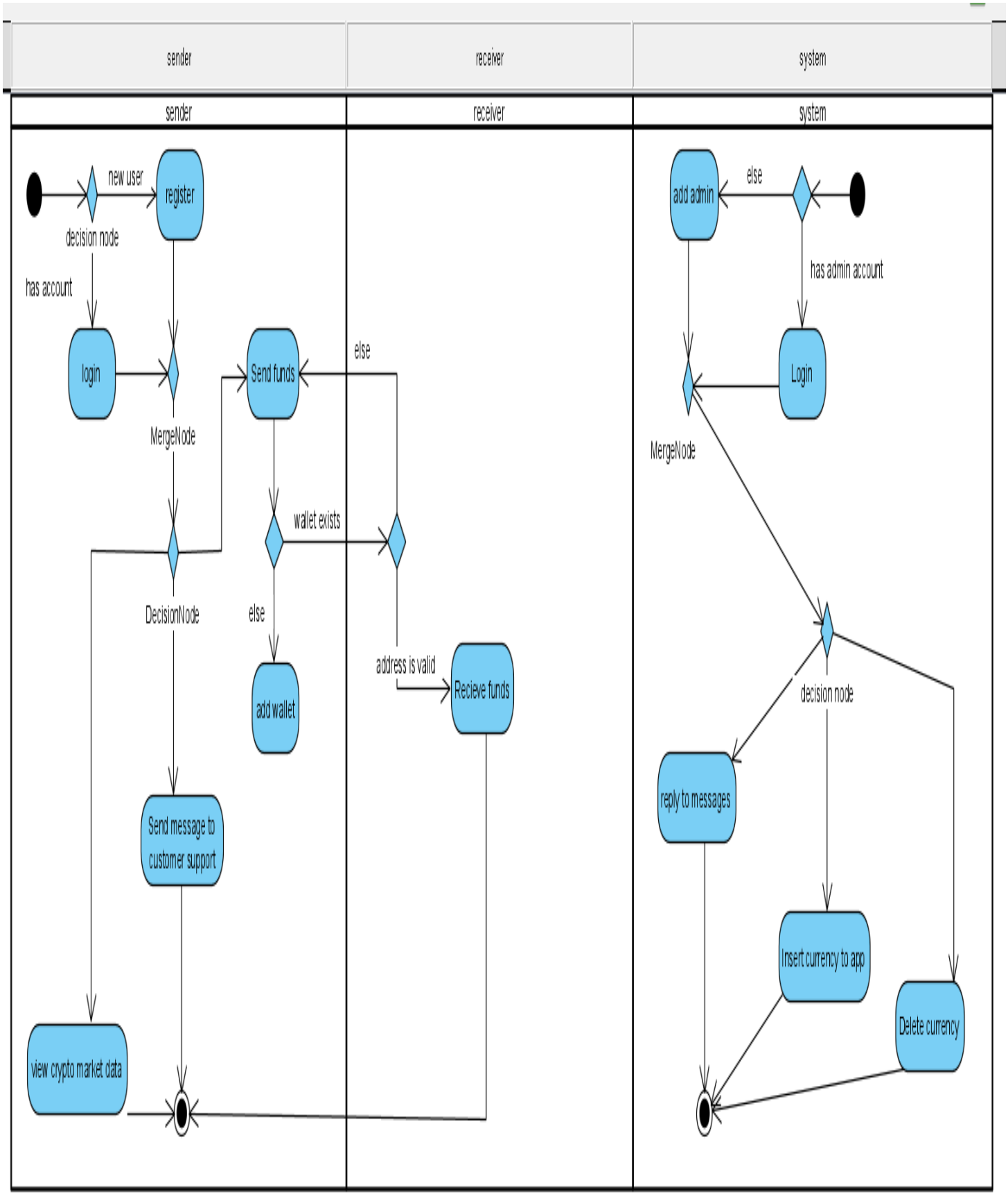


As we can see in the diagram, every entity has its corresponding controller, and a controller communicates not only with its corresponding entity but can sometimes require communicating with other models. The transaction entity contains two relationships with the wallet which means every transaction made will be between 2 wallets hence we register from one side the sender and from the other the receiver.

2-4- Activity Diagram

As we can see here in the activity diagram there exists 2 actors, users which are composed of a sender and receiver for a certain case and admin who can have a certain level of control for the app. Starting by the user we can see that after registering for the first time or logging in, the user can be directed by different flow hence we used a decision node here to give the user different choice of what to do. He can start by adding a wallet or sending fund to another user which is labeled by the receiver, he can also send customer support request in case he needs help, or he can view a cryptocurrency's daily data such as its price market cap or the variation of its price in a month.

As for the admin side, he can add a new currency for the app to support by providing a list of cryptos through the CoinMarketCap API, he can also remove one. The admin can also reply to users who requested help by sending them emails.

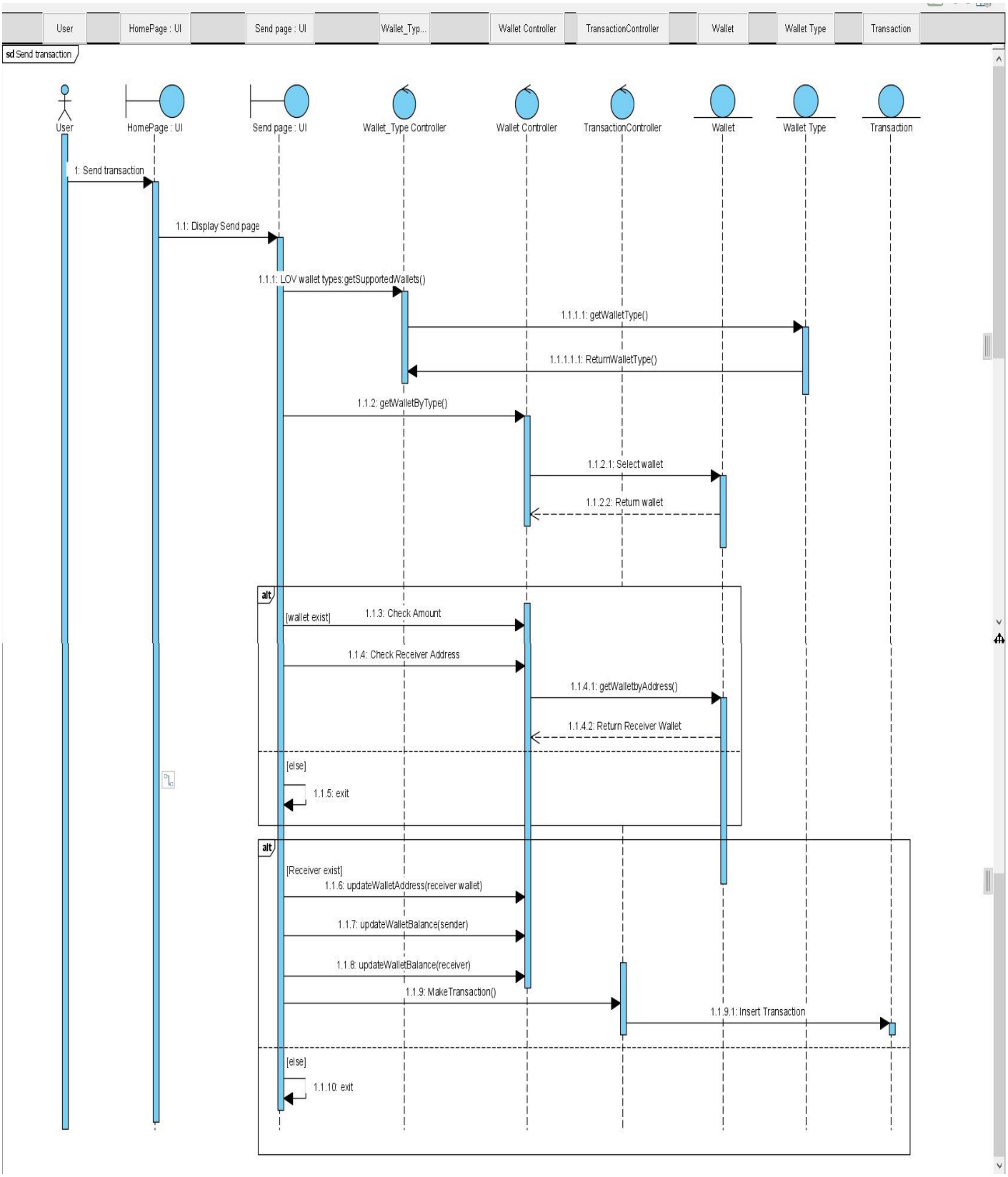

sender
receiver
system
sender
receiver
system
new user
register
decision node
has account
login
MergeNode
Send funds
else
wallet exists
else
add wallet
address is valid
Recieve funds
DecisionNode
Send message to customer support
view crypto market data
add admin
else
has admin account
Login
MergeNode
decision node
reply to messages
Insert currency to app
Delete currency

*2-5- **Sequence Diagram***

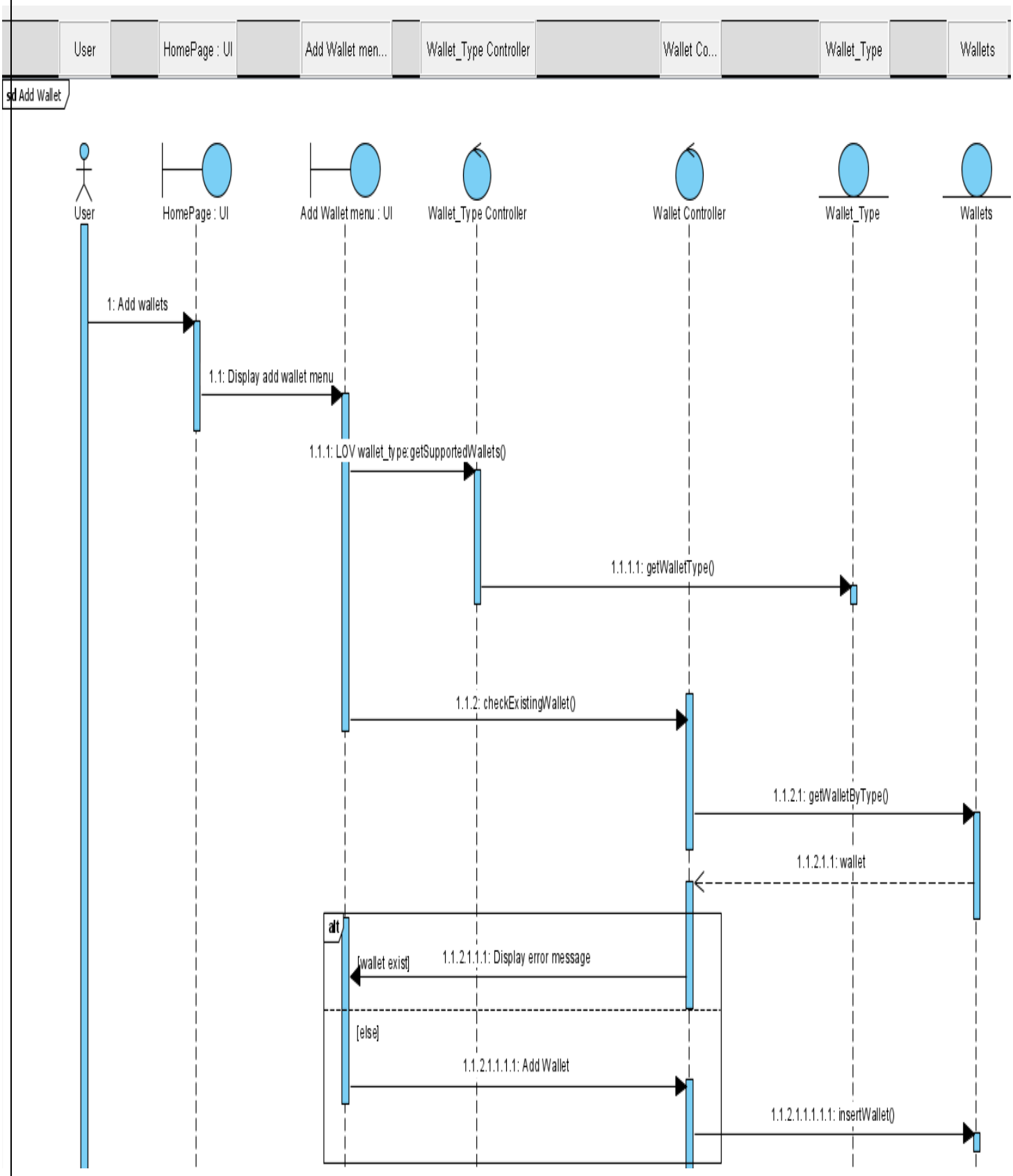
2-5-1- Send transaction.

- Boundaries:
  - Homepage
  - Send Page
- Controllers:
  - Wallet_Type
  - Wallet
  - Transaction
- Entities:
  - Wallet
  - Wallet_Type
  - Transaction

The user is directed to the send page where he must choose the type of currency to send, in order to do that we must get all the wallet types from the Wallet_Type entity via the Wallet_Type controller and displaying them in our send page view. Then the user selects the desired wallet to send from, if it exists then we do a control over the amount to send in order to check if the user's balance is enough and, we check if the receiver address is correctly entered either by using the built in QR scanner or by copying it. In case all these tests return true we update the sender's and receiver's balance, and we insert a transaction into the entity Transaction via the Transaction Controller.


User
HomePage : UI
Send page : UI
Wallet_Typ...
Wallet Controller
TransactionController
Wallet
Wallet Type
Transaction
sd Send transaction
User
HomePage : UI
Send page : UI
Wallet_Type Controller
Wallet Controller
TransactionController
Wallet
Wallet Type
Transaction
1: Send transaction
1.1: Display Send page
1.1.1: LOV wallet types:getSupportedWallets()
1.1.1.1: getWalletType()
1.1.1.1.1: ReturnWalletType()
1.1.2: getWalletByType()
1.1.2.1: Select wallet
1.1.2.2: Return wallet
alt
[wallet exist]
1.1.3: Check Amount
1.1.4: Check Receiver Address
1.1.4.1: getWalletbyAddress()
1.1.4.2: Return Receiver Wallet
[else]
1.1.5: exit
alt
[Receiver exist]
1.1.6: updateWalletAddress(receiver wallet)
1.1.7: updateWalletBalance(sender)
1.1.8: updateWalletBalance(receiver)
1.1.9: MakeTransaction()
1.1.9.1: Insert Transaction
[else]
1.1.10: exit

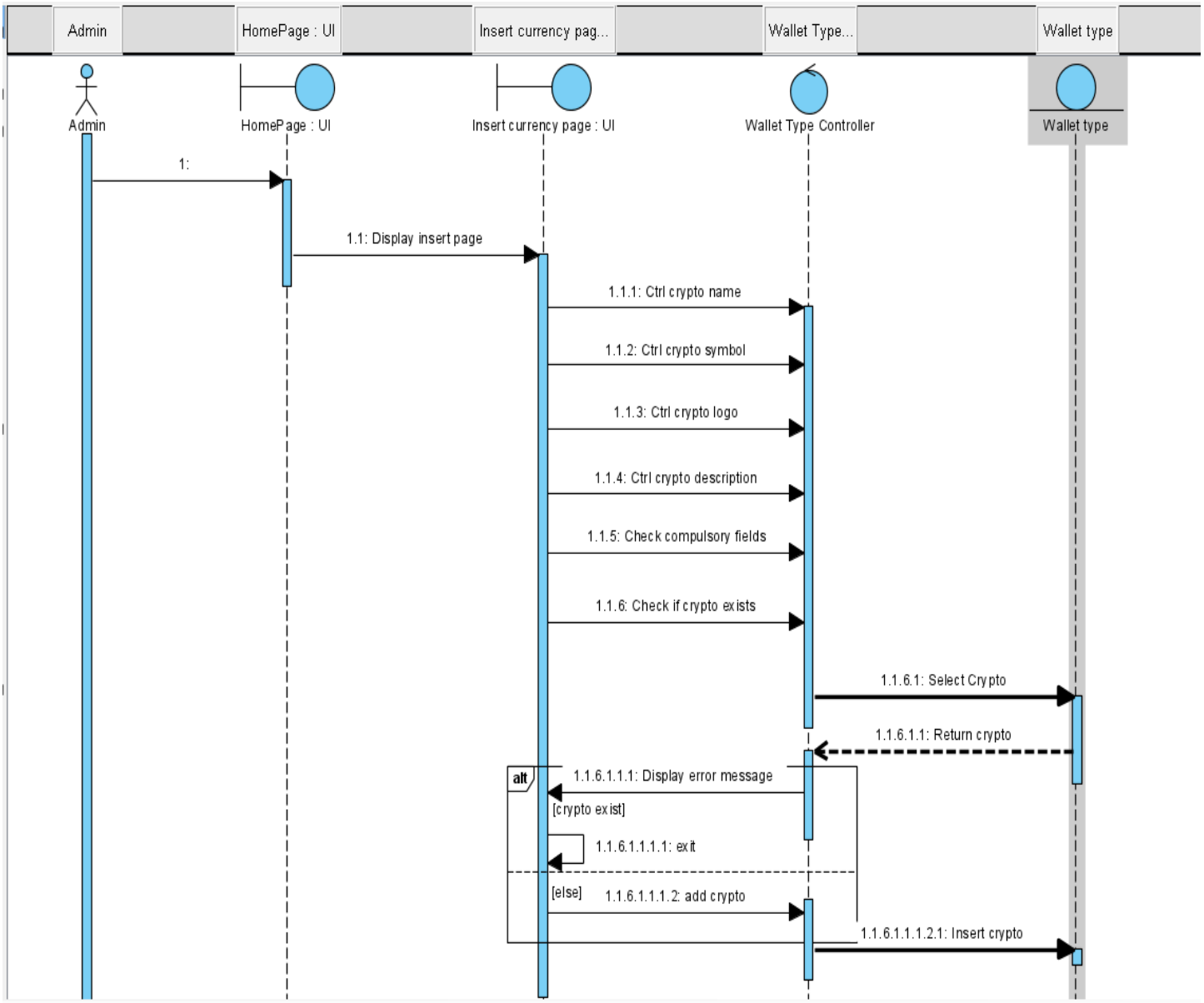
2-5-2- Add Wallet



- Boundaries:
  - Homepage
  - Add Wallet Menu
- Controllers:
  - Wallet_Type
  - Wallet
- Entities:
  - Wallet_Type
  - Wallet

The Add wallet menu is displayed from the homepage where we can find a list of supported currencies to choose from, we get the list from the Wallet_Type entity through the Wallet_Type controller and display them. After the user chooses which currency to add tests must be made. We check if the wallet already exists by querying the wallet from the Wallet entity via the Wallet Controller. If the wallet already exists an error is displayed for the user, otherwise we insert the wallet into the Wallet entity via its corresponding controller.

2-5-3- Insert Currency



- Boundaries:
  - Homepage
  - Insert Page
- Controller:
  - Wallet_Type
- Entities:
  - Wallet_Type

For the admin to add a currency he must choose a cryptocurrency name with its corresponding symbol provided by an API, then he must add a description and a logo. We check all the compulsory fields and then check if the currency is already added by querying the chosen one from the Wallet_Type entity via the Wallet_Type controller. If the currency already exists, then we display an error for the admin otherwise we insert it into the Wallet_Type entity.

3- Implementation

3-1- Technical environment used.

In order to have a well-structured and reusable code between different platforms including the android app and the admin webpage the MVC design pattern was adopted. During development time we decided that each Model must correspond to a controller therefore for each created model we implemented its controller. Moreover, a controller can perform operations from various models depending on the view. For example, the wallet controller can perform operations from the wallet and wallet_type controller to provide the necessary control for the user's input. Sharing code and reusing code was a primary objective to achieve because time is gold therefore all controls over a user input are made in server side meaning we had to make requests from the android app to our php scripts and then a json response is returned to the app with the corresponding error messages to give the user feedback if the operation was a success or not.

The models which connect with the database are the following:

- Wallet
- User
- Wallet_Type
- Transaction
- Message

The Controllers which perform operations to control user actions are:

- WalletController
- Wallet_TypeController
- UserController
- TransactionController
- MessageController
- AdminController

For the views we have the android app and the admin webpage.
For the admin we had some differences in some methods compared to the User therefore we decided to differentiate between the user and admin controller but sharing the same model which is the user model.
Each model extends the connection to the database which is represented by the Dbh class and each Controller extends the Controller class which contains the basic functions such as redirecting to a view and some shared methods between different controllers.
To be able to perform operations directly from the class and not create for each specific operation to use a different small script where we initiate a class, we decided to construct a routing method that will allow us to choose which controller and which method to perform directly in the URL. We created an index page that for any form submission or method call will redirect through it and initiate an object of type App which is a class we implemented that will route through the specific method in the URL and perform its operation.

*3-2- **Screens description***

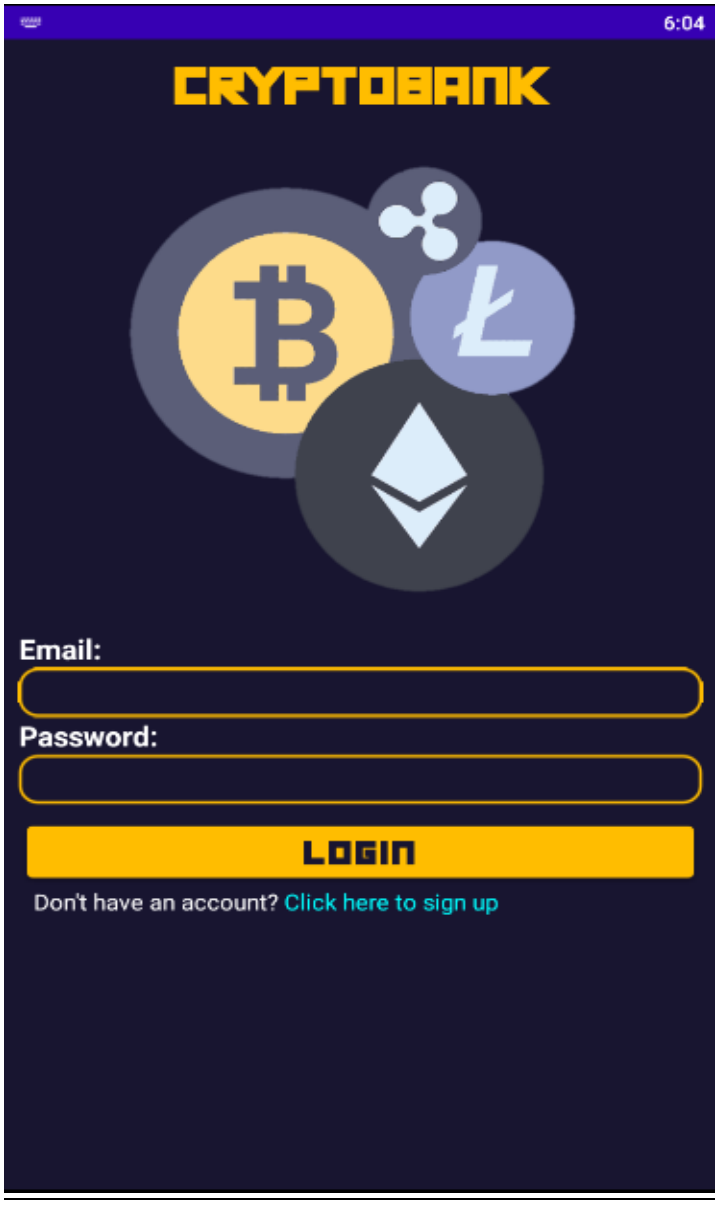
3-2-1- User's views

3-2-1-1- Login

To use the app, users will need to login by entering their email and password. Once logged in, home page is displayed.

Class used in Login page: User.

Controller used in Login page: UserController.

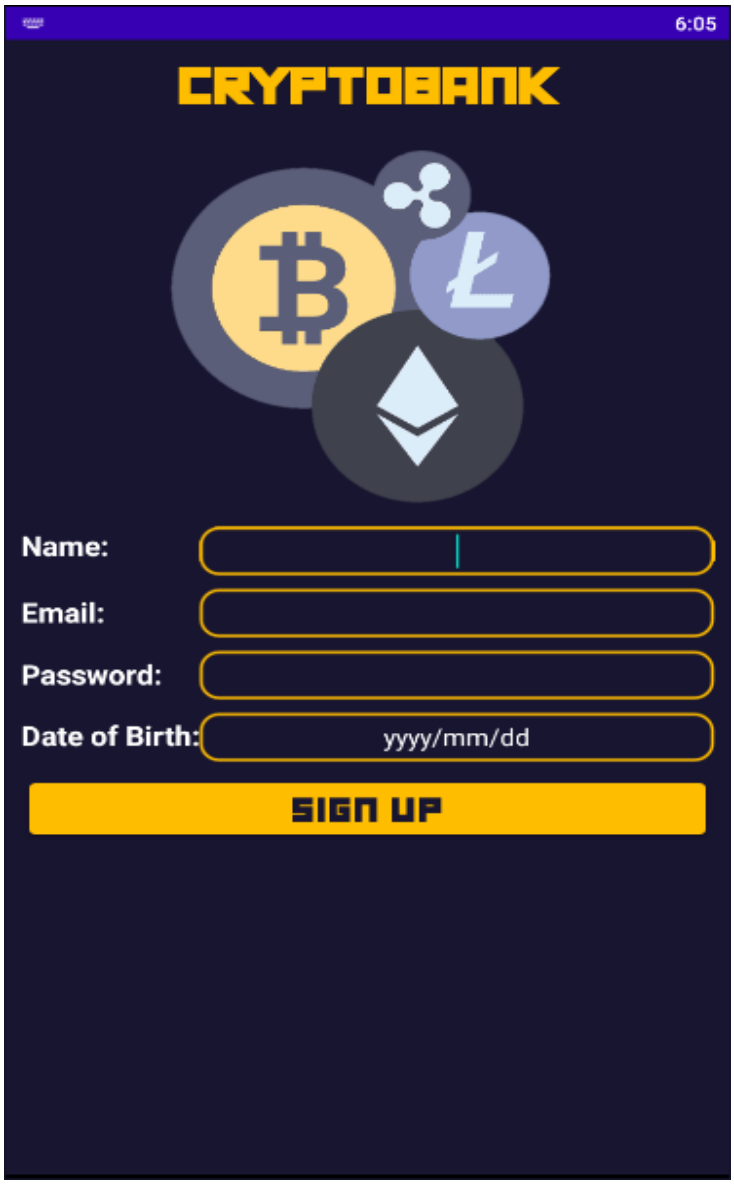


3-2-1-2- Register:

For new users, a registration form must be filled. They will have to give their name, valid email, valid password, and their date of birth.

Class used in register page: User.

Controller used in register page: UserController.

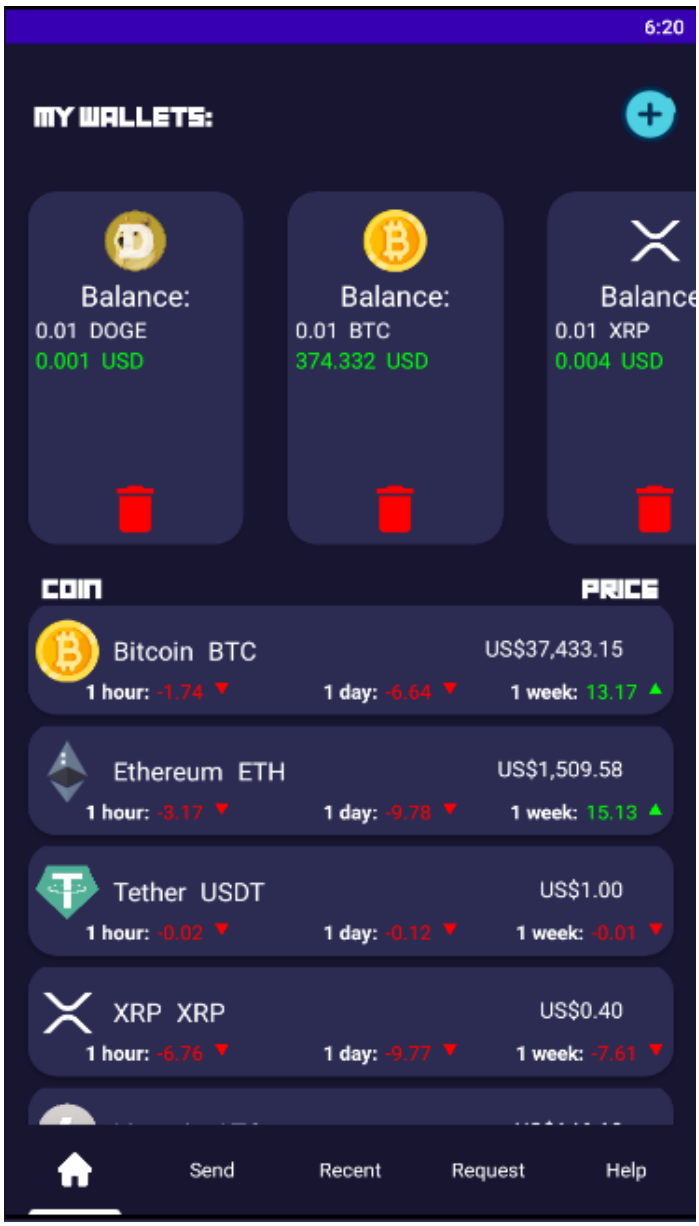


3-2-1-3- Home:

In the home page, users can check the prices variation in percentage during the last hour, day and week. Users can also create wallets (by clicking the + button) to start buying and selling. For testing purposes, the default wallet balance is 0.01.

Classes used: Wallet and Wallet_Type

Controllers used: WalletController and Wallet_TypeController



3-2-1-4- Crypto Info:

When the user clicks on a currency, a page is displayed where the variation is represented in a graph with a description of the currency and market stats. These information are extracted from an API.

Class used: Wallet_Type.

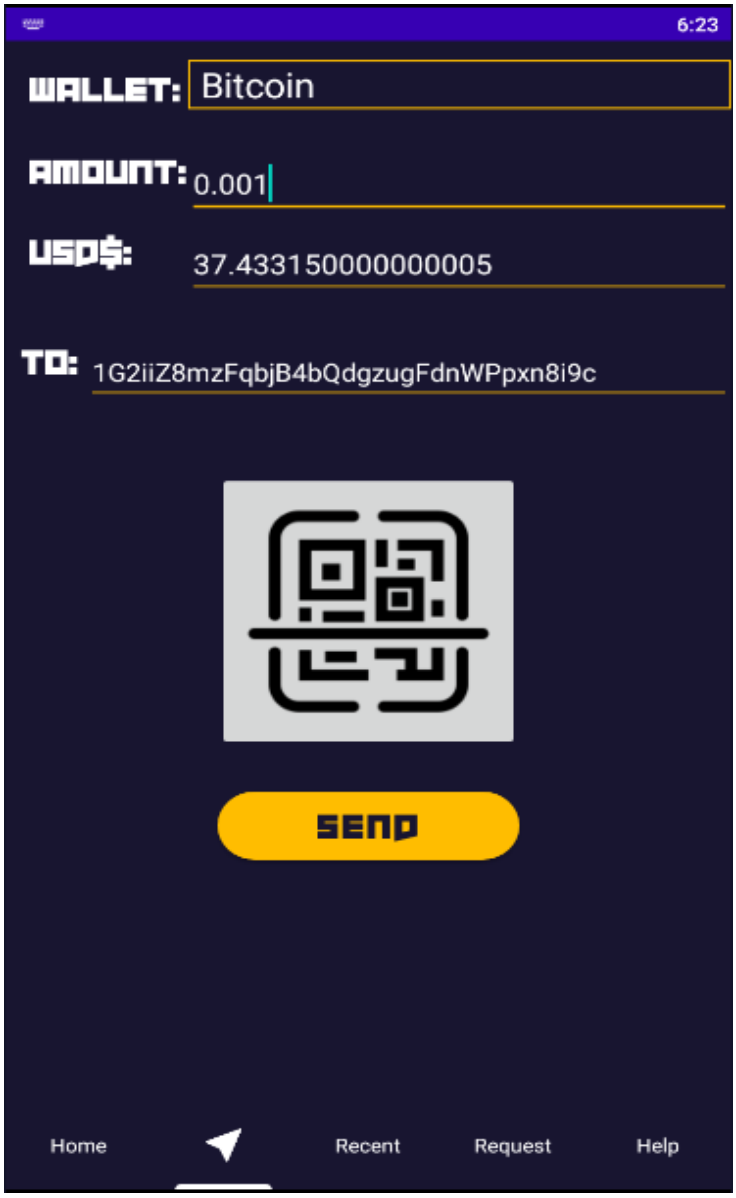
Controller used: Wallet_TypeController.

3-2-1-5- Send:

To Send funds, the user must select the wallet from which he wants to send,and input the amount and the receiver's address (or scan it).

Classes used: Wallet and Transaction.

Controller used: WalletController.

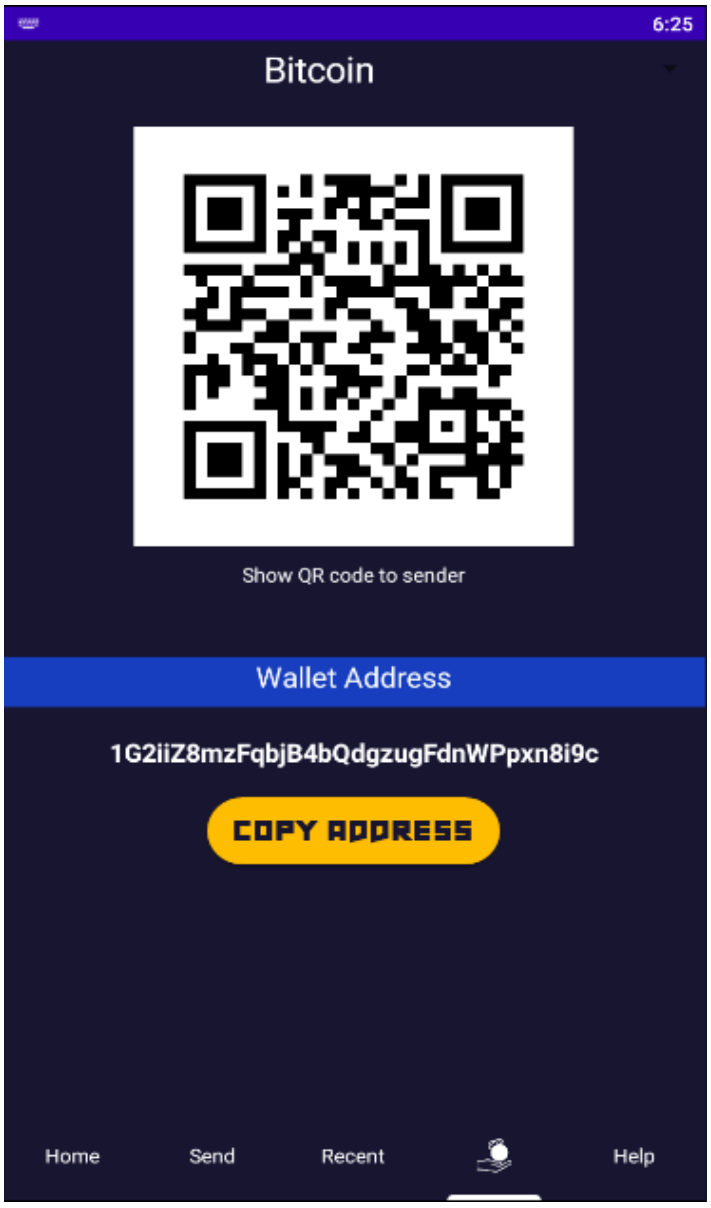


3-2-1-6- Request:

The sender scans the receiver's address from the request page, or the receiver can just copy and send him the address.

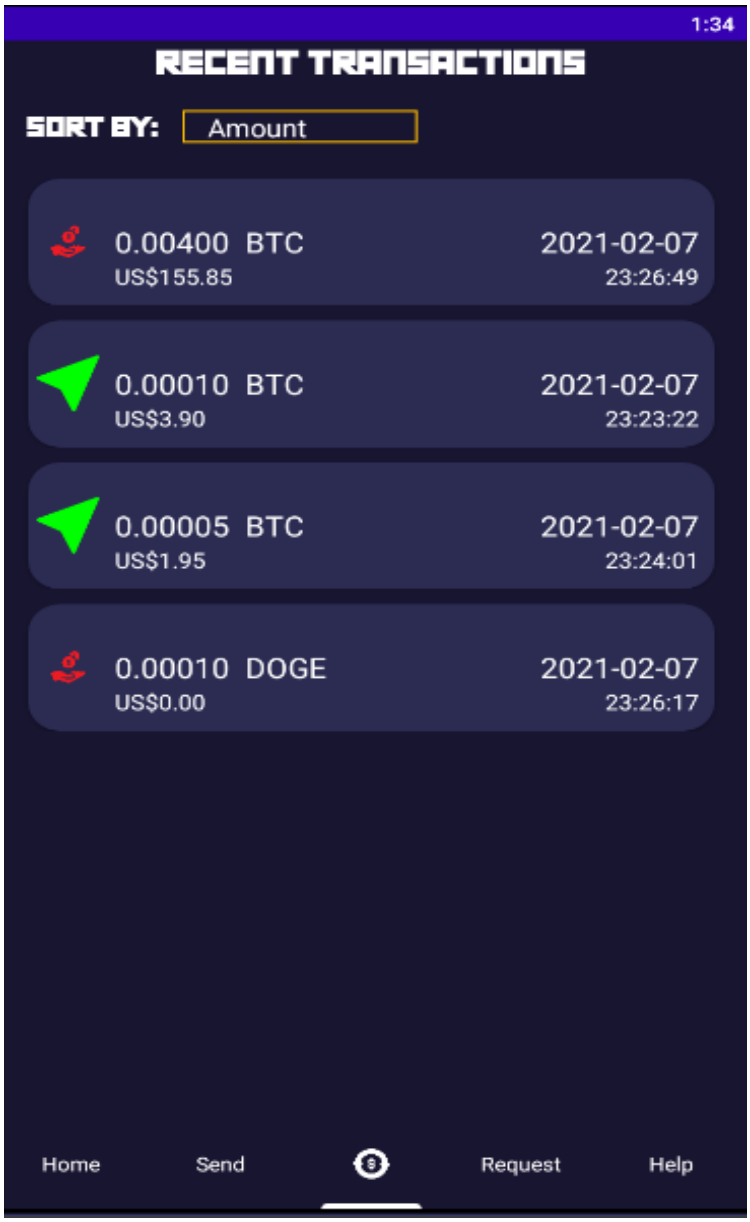
Classes used: Wallet and Transaction.

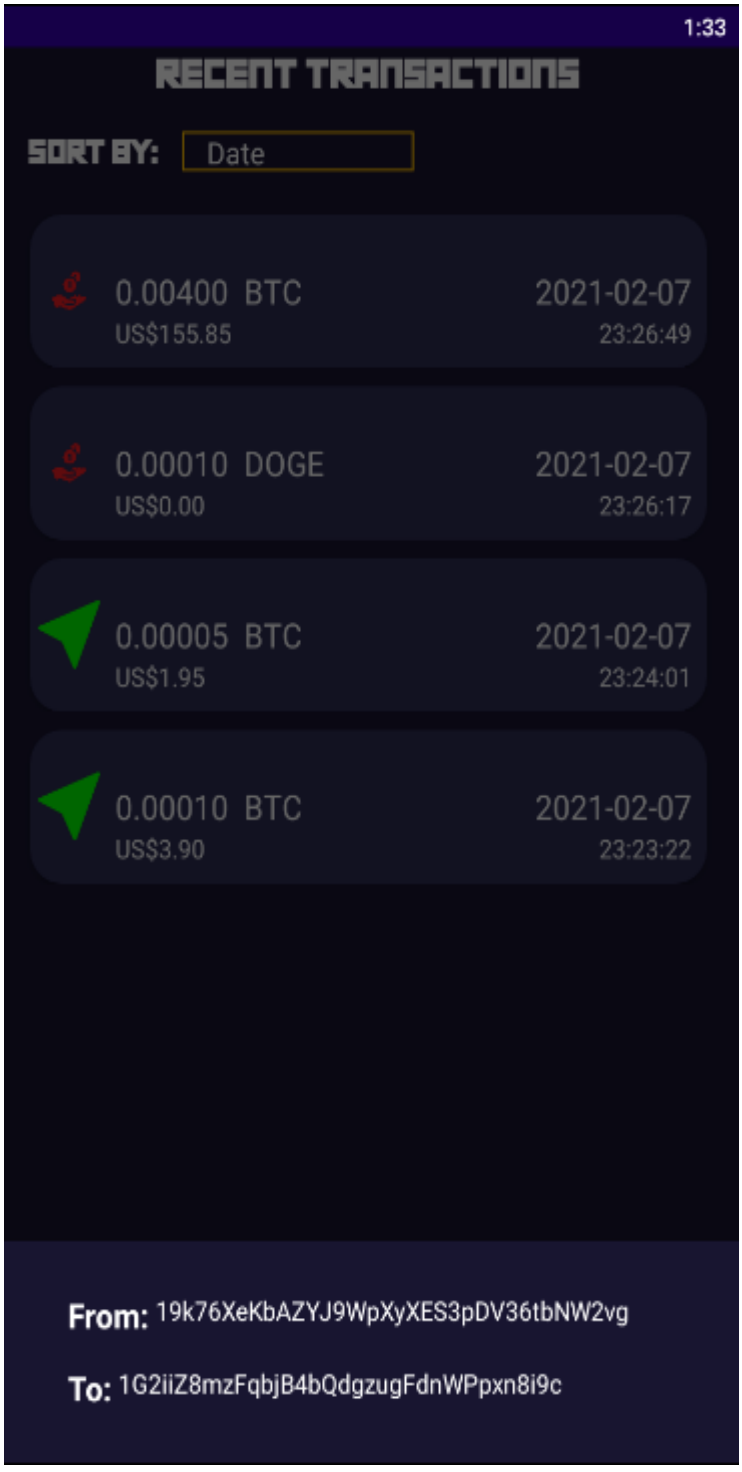
Controller used: WalletController.



3-2-1-7- Recent Transactions:

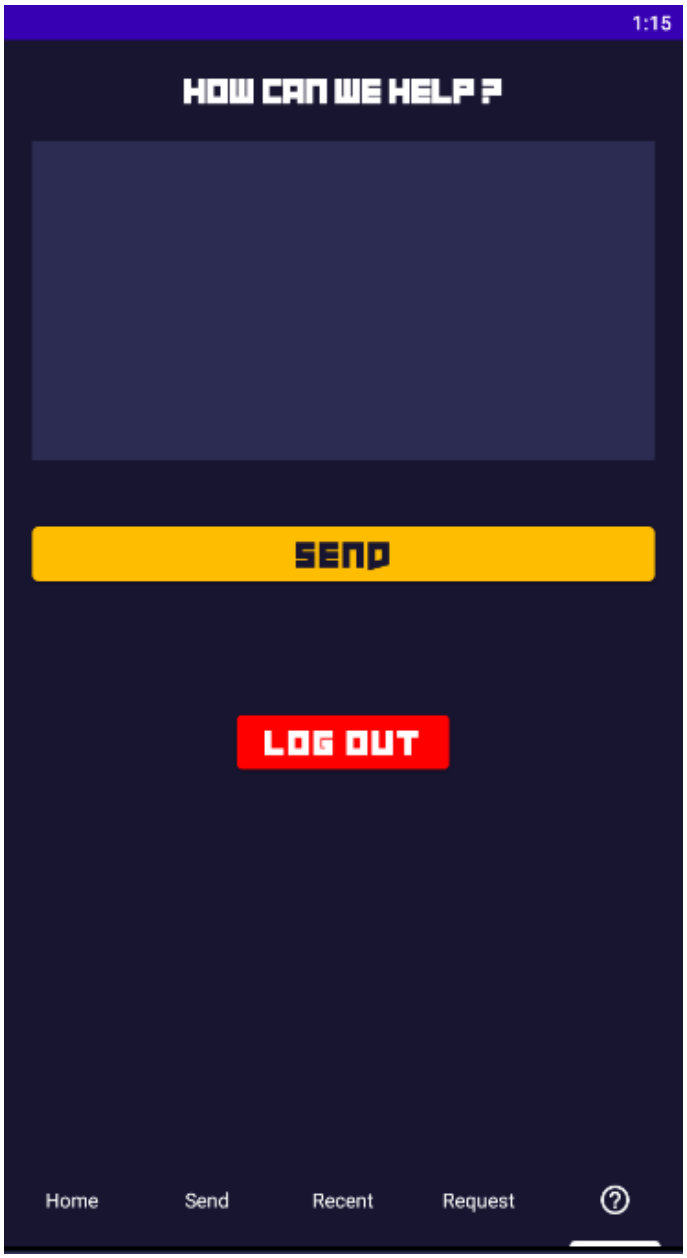
In this page, users can check recent transactions. These transactions can be sorted by date or amount. Users can also view transactions details (sender and receiver addresses) by clicking on it.




1:33
RECENT TRANSACTIONS
SORT BY:
Date
0.00400 BTC
US$155.85
2021-02-07
23:26:49
0.00010 DOGE
US$0.00
2021-02-07
23:26:17
0.00005 BTC
US$1.95
2021-02-07
23:24:01
0.00010 BTC
US$3.90
2021-02-07
23:23:22
From: 19k76XeKbAZYJ9WpXyXES3pDV36tbNW2vg
To: 1G2iiZ8mzFqbjB4bQdgzugFdnWPpxn8i9c

3-2-1-8- Help:

In the Help page, users can send a message to customers support (1 every 24hours), and finally logout.



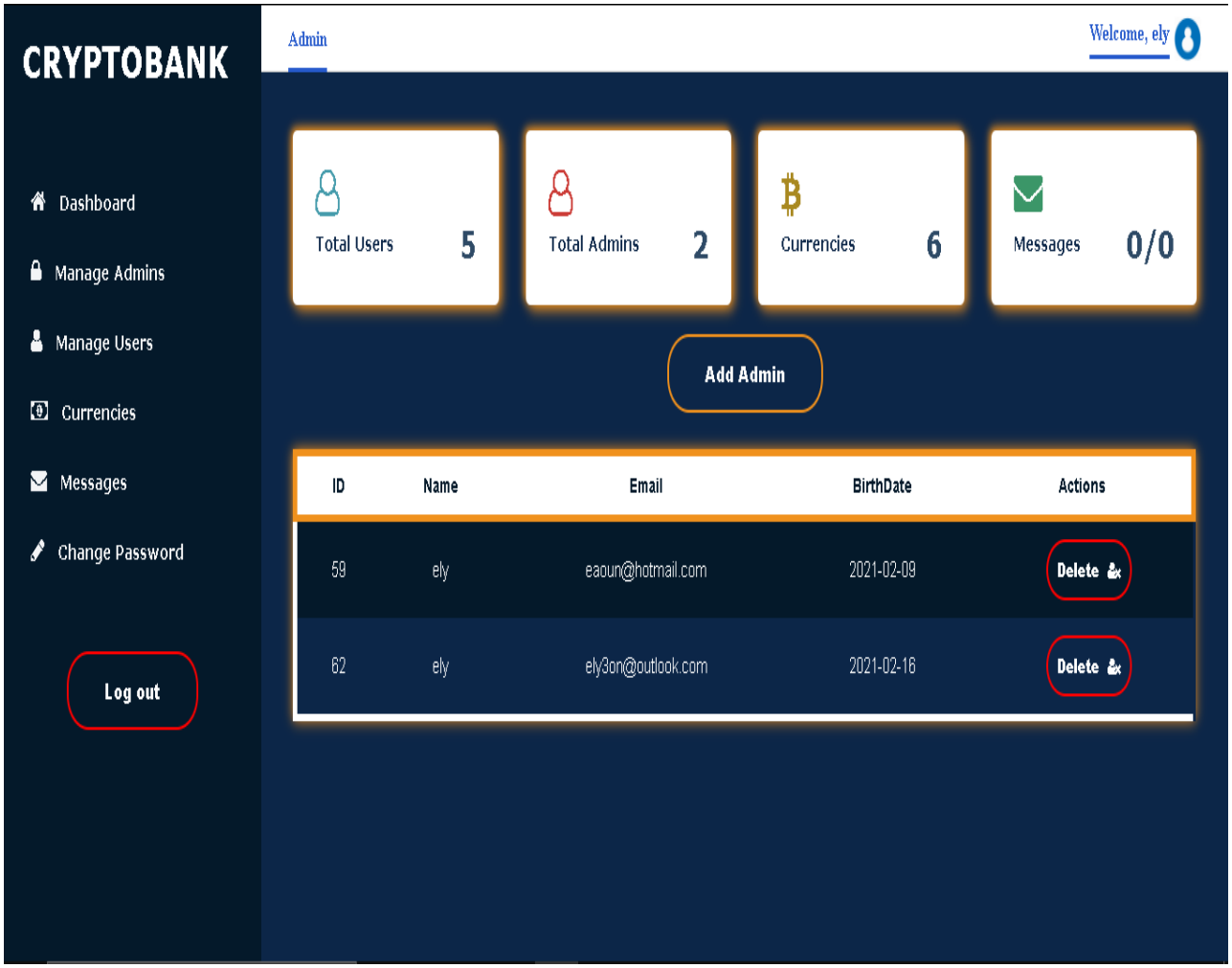
3-2-2- Admin's views

3-2-2-1- Admin Log In

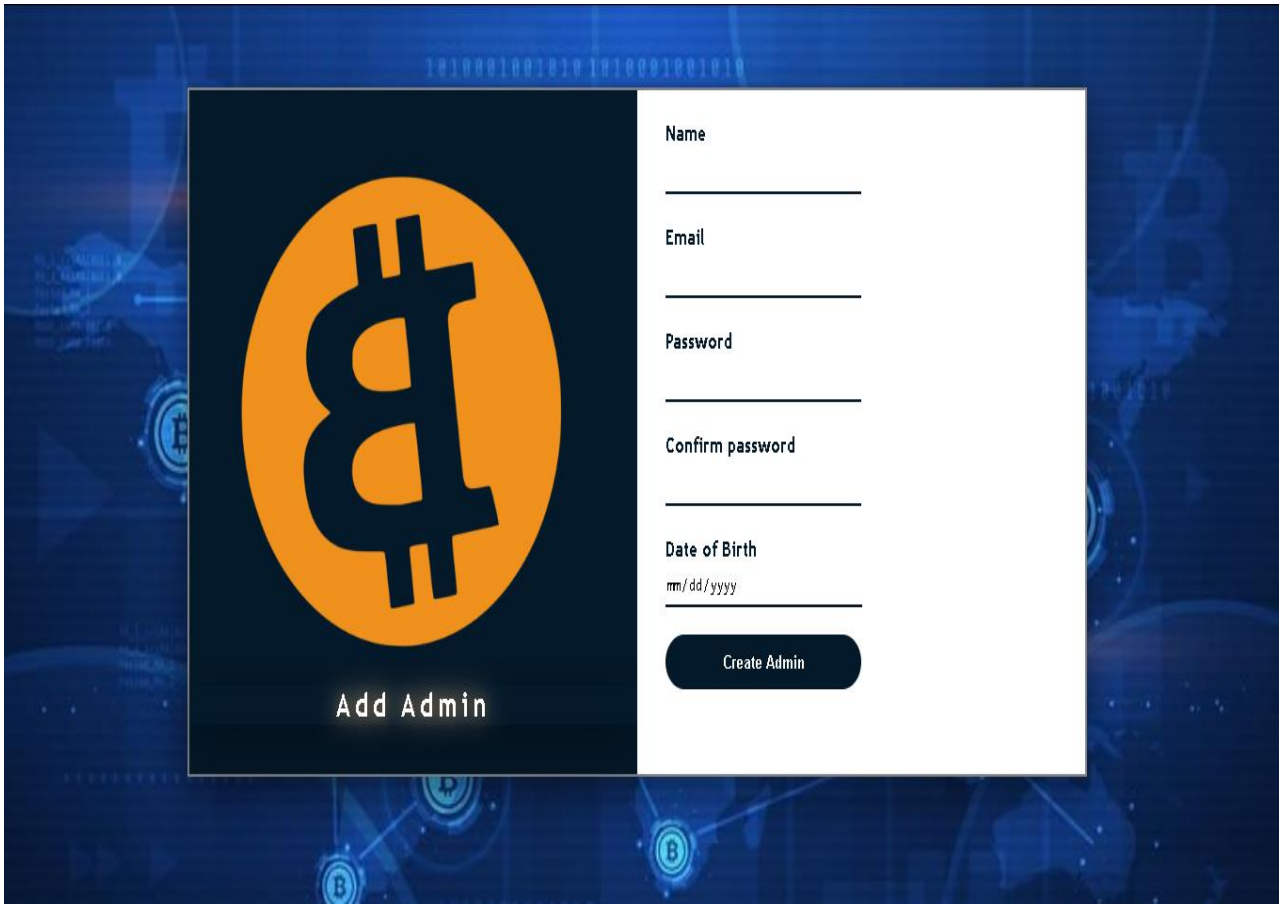
The admin logs using his email and password.
Controller used: Admin Controller.
Model used : User Model

3-2-2-2- Manage Admins:

In this page, we can check the admins that exists, and we can remove one or add one by clicking “Add Admin”.
Controller used: Admin Controller.
Model used: User model.

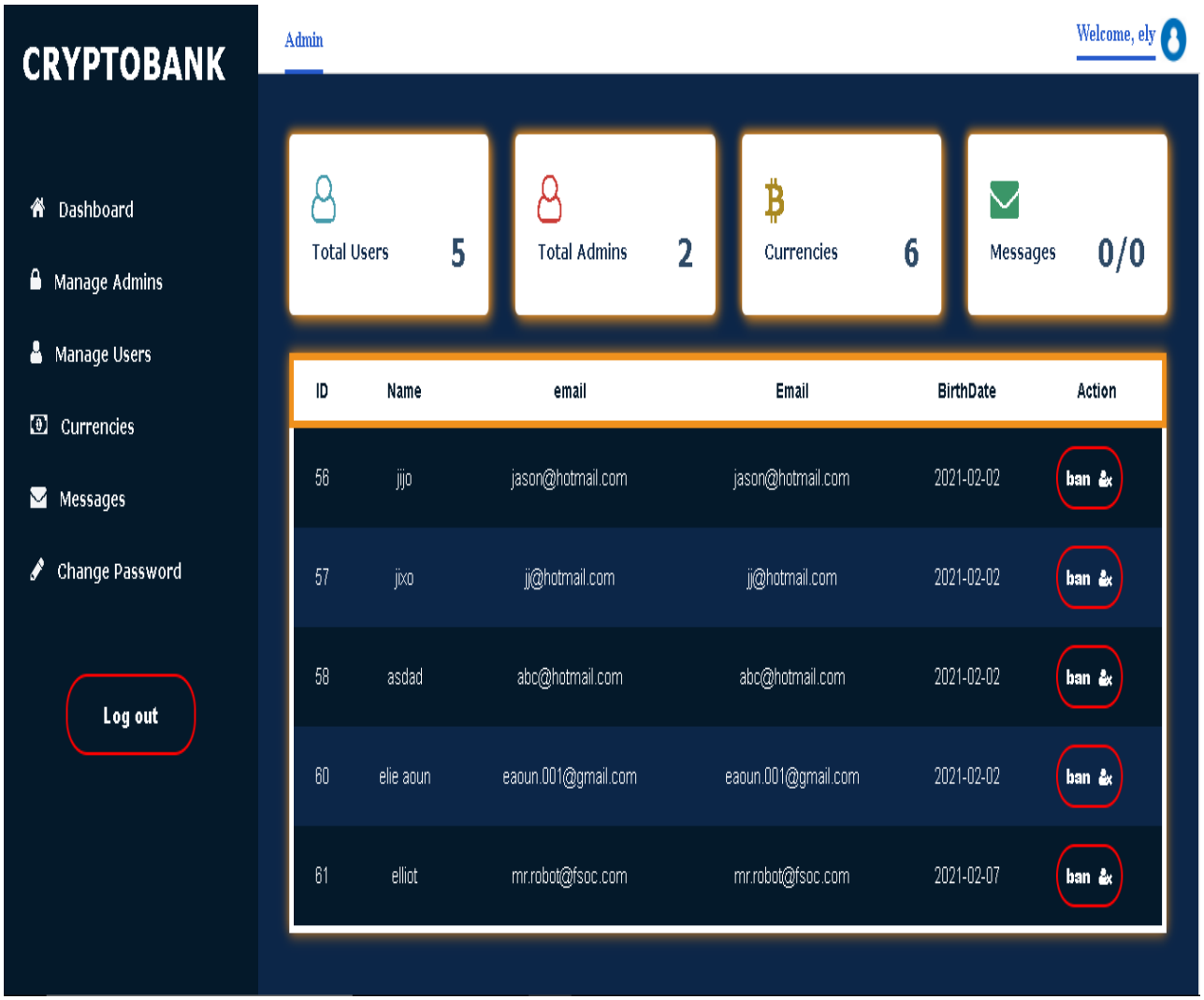


This page presents the form in which we fill to add an admin.



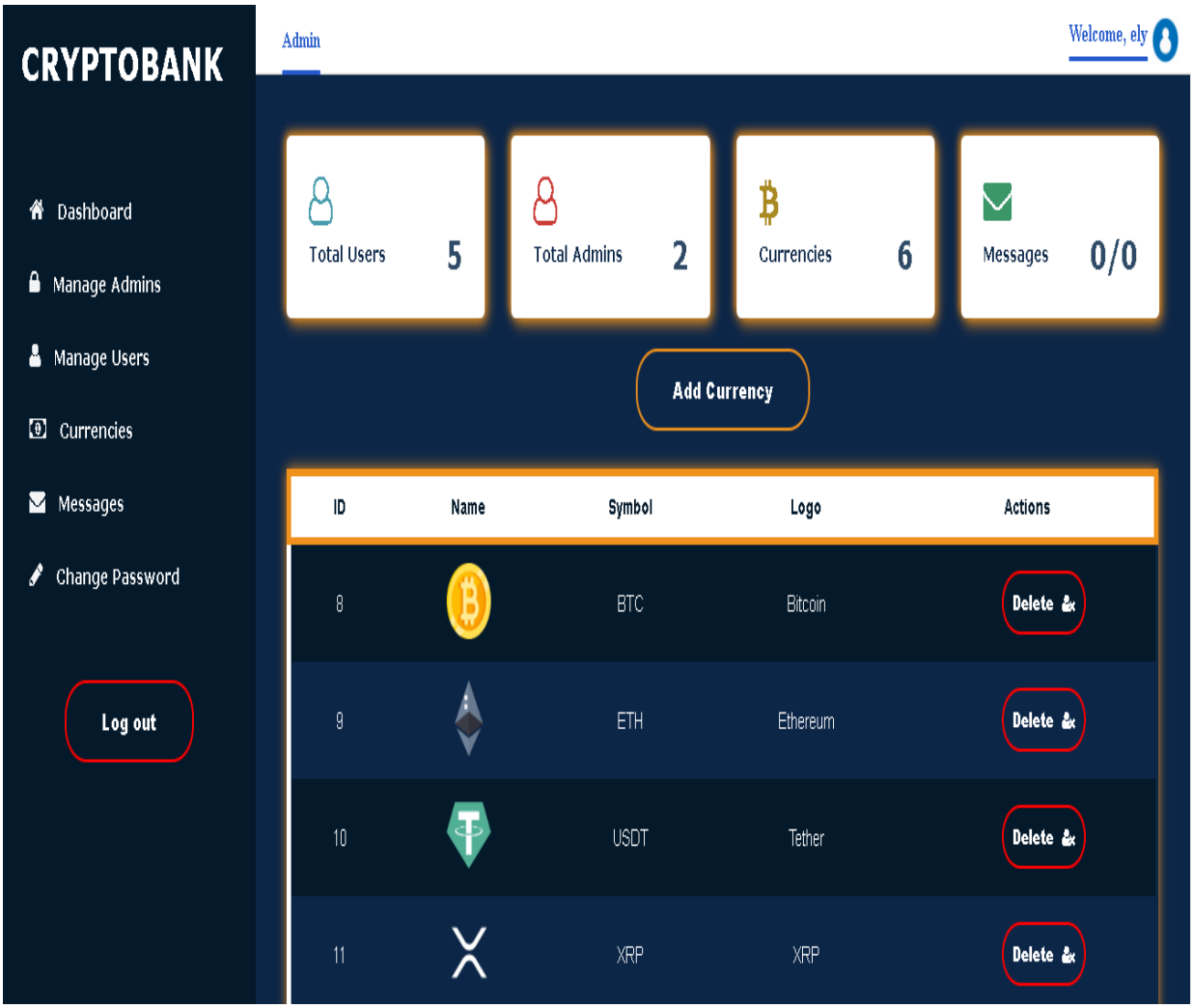
3-2-2-3- Manage Users

In this page, the list of users is displayed. An admin can ban a user by clicking on “ban”.
Controller used: Admin Controller.
Model Used: User Model.



3-2-2-4- Currencies

An admin can view the current currencies supported in the android app. He can either delete or add one.
Controller used: Wallet Type Controller
Model used: Wallet Type Model



In order to add a currency the admin must fill the form by choosing the currency name and choose a logo and fill in a description.
The list of currencies with their corresponding symbols are provided by the CoinMarket API

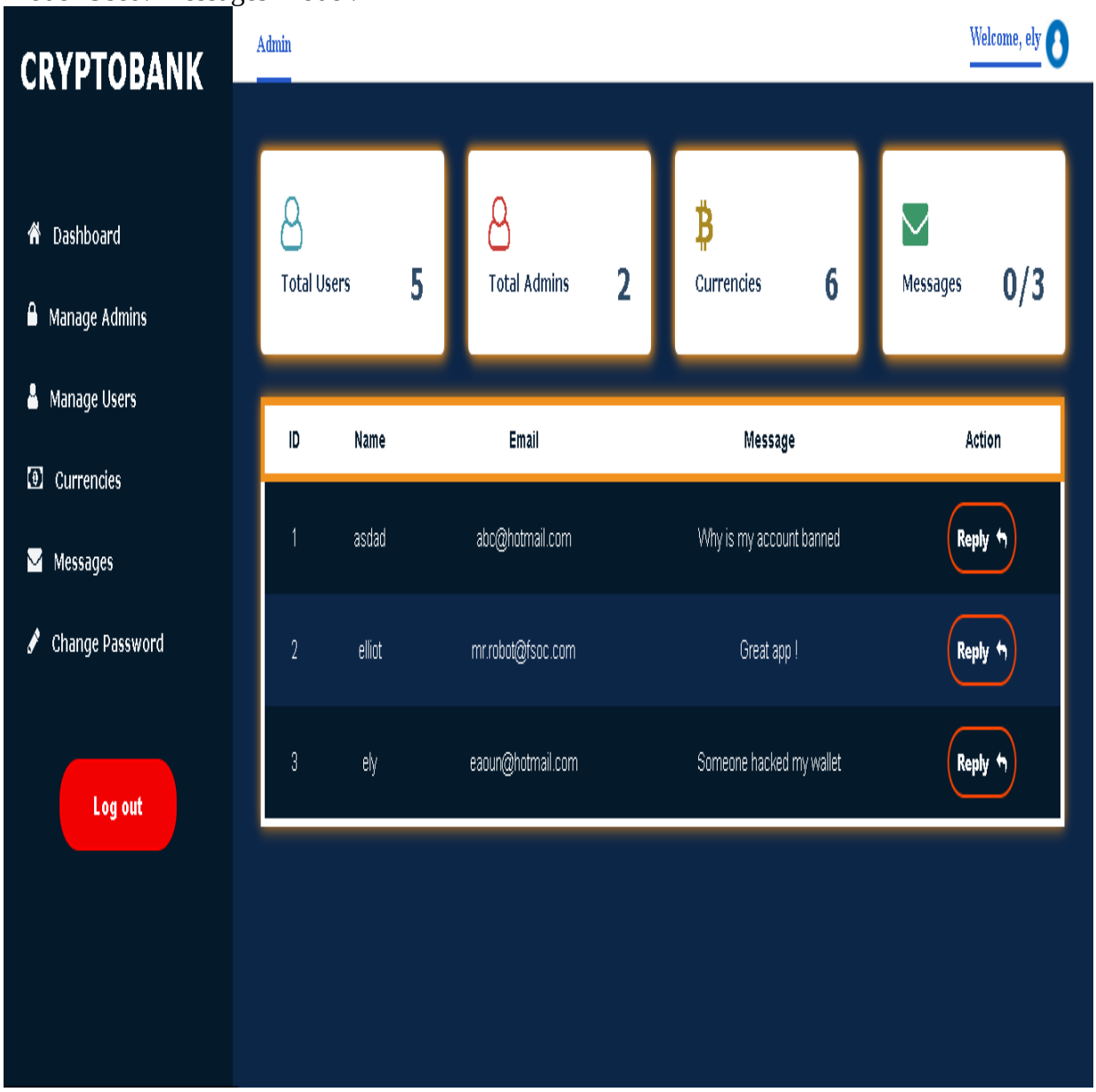
.

3-2-2-5- Messages

In this page, users' messages are displayed. An admin can reply by clicking reply.

Controller used: Messages Controller.

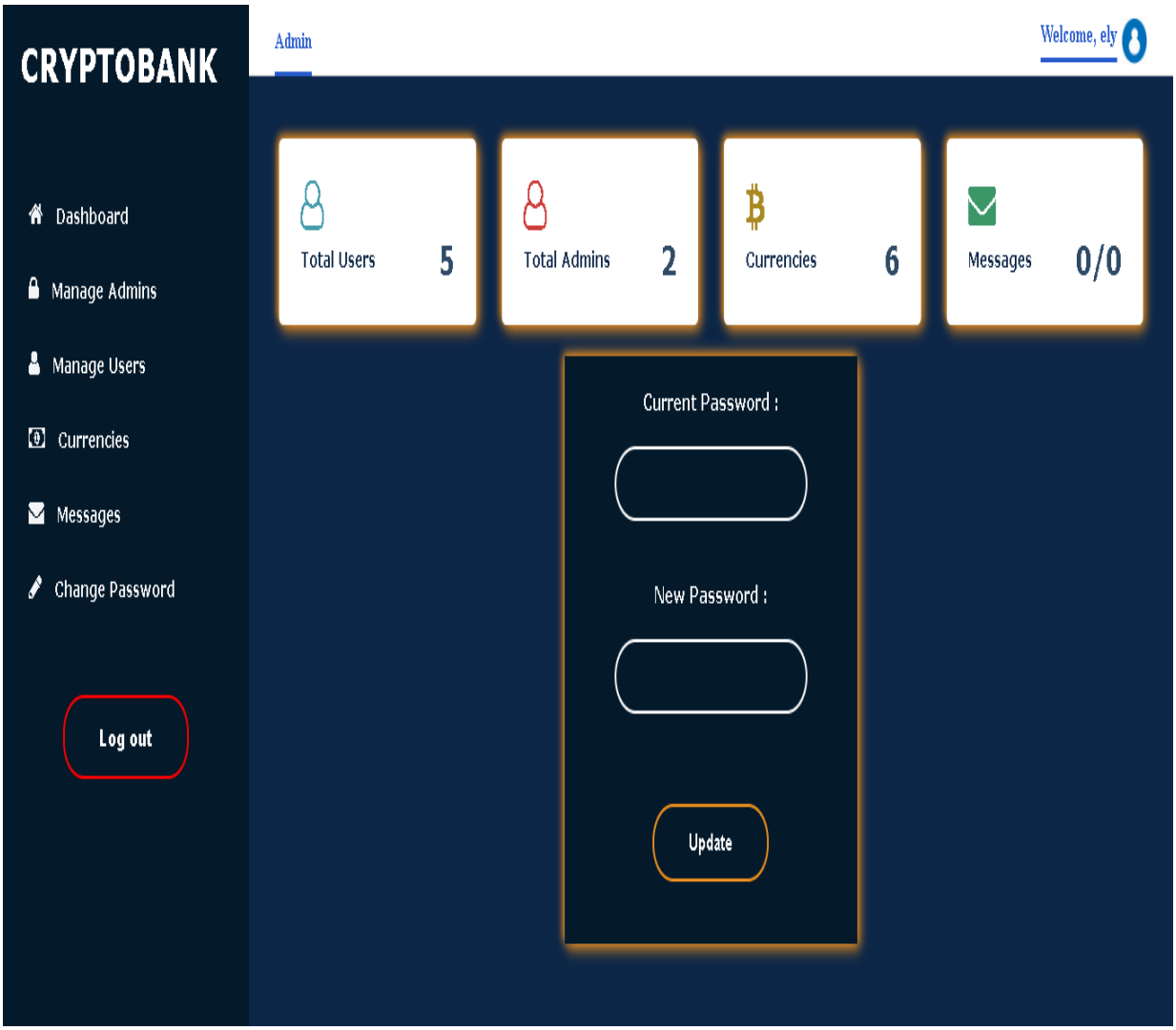
Model Used: Messages Model.



When Clicking on reply the admin is presented on a page where he must fill what he wants to send to the user via email. The user's email is supplied in the form.

3-2-2-6- Change Password

An admin can change his account's password providing his current password to make sure he's the one who is making this decision.
Controller used: Admin Controller.
Model used: User model.



4- Conclusion

To Conclude, this project gave us a good understanding on how to plan and execute a project and moreover helped us improve our skills in PHP, Android development and UML diagrams. What is left to implement is a way to fund a wallet with a real credit card from the user and to use the blockchain technology. Therefore, our simulation provides a clear understanding of how a wallet app might work and in what form it should be presented to provide the user a clean interface. We would consider developing the missing parts for a real-world environment since we felt excited about our work.