Design and Implementation of Mobile applications

POLITECNICO
MILANO 1863

CryptoSimulator

Design Document

Francesco Montanaro - 10749086

Amedeo Pachera - 10750281

Index

1. Introduction

1.1 Purpose of the document

1.2 Project description

1.3 Goals

1.4 Requirements

1.5 Domain assumptions

2. Architectural design

2.1 Overview: High-level components and their interaction

2.2 Components view

2.3 Runtime view

2.4 Component interfaces

2.5 Selected architectural style and patterns

2.6 Other design decision

3. Requirements traceability

4. User interface design

5. Flow of Implementation and Integration

5.1 Implementation and Integration strategies

5.2 Technologies Involved

5.3 External APIs

5.4 Mobile Application

5.5 Server

6. Testing

6.1 Static Analysis

6.2 Unit Testing

6.3 Integration Testing

6.4 End-to-end Testing

7. References

1. Introduction

1.1 Purpose of the document

The purpose of this document is to give a description of the project, and the architectural choices made to implement all the functionalities required.

It provides guidelines and materials which are intended to assist technical staff to perform system testing and implementation, and to facilitate future maintenance and features extension.

1.2 Project description

CryptoSimulator is a cross-platform mobile application for monitoring and forecasting the trend of cryptocurrencies over time.

It allows users to be always up to date on the value of the most important cryptocurrencies by exploiting interactive graphs and plots. By the implementation of AI algorithms it allows users to monitor and predict cryptocurrency's trends and simulate the future of their preferred assets. In other words, thanks to an artificial intelligence engine, based on a recurrent neural network trained to predict future trends, CryptoSimulator helps users to make decisions about when to sell or buy the desired assets in order to maximize profits and reduce losses.

The application expands its functionalities by allowing users to authenticate and to create and manage their personal wallet with the cryptocurrencies owned. In particular, users can always be up to date on the current value of their wallet and understand how it could evolve over time thanks to the predictions made by the AI.

1.3 Goals

The goals that the system aims to achieve are listed and described below:

- **G1:** The application allows users to retrieve and visualize data on their preferred cryptocurrencies.
- **G2:** The application allows users to retrieve and visualize predictions on future trends of the selected cryptocurrencies.
- **G3:** Users can create and manage their personal wallet of their preferred cryptocurrencies.
- **G4:** The system allows users to visualize the current value of their wallet and its future trend.
- **G5:** The system allows users to save and share their preferences among multiple devices.

1.4 Requirements

This section aims to define and list all the system's requirements in order to achieve the goals set.

- **R1:** The System allows users to authenticate into the application.
- **R2:** Users are allowed to retrieve information regarding the preferred cryptocurrencies.
- **R3:** The system builds predictions on future trends of the cryptocurrencies.
- **R4:** The application presents data to users in the form of interactive graphs and plots.
- **R5:** Users are allowed to interact with graphs and plots by setting different time granularities and by visualizing specific periods' values.
- **R6:** The system allows users to create and manage their own wallet by adding, editing and deleting their cryptocurrencies.

- **R7:** The system allows users to visualize the current value of their wallet.
- **R8:** The application allows users to visualize future predictions on how their wallet value may evolve.
- **R9:** The system allows users to store information on the cloud in order to recover them from multiple devices.

1.5 Domain assumptions

All the assumptions under which the the system is supposed to operate and which are necessary to guarantee its correct functioning are listed and described below:

- **D1:** The user has a device capable of supporting all the APIs, libraries and frameworks on which the system is built.
- **D2:** The user has a working internet connection.

2. Architectural Design

2.1 Overview: High-level components and their interaction

The software to be is designed as a client-server system with a 3-tier architecture: client, application server and a database. In particular, the client provides an interface to the users and it incorporates only a part of the application's logic. The main logic features of the software are delegated to the server, which transitively communicates with the database, whose purpose is to store the data used by the application itself. This architecture has been chosen in order to make the software scalable and flexible to future integrations and modifications, without giving up a fair robustness and a good user experience. The communication between the various components is managed through the exchange of requests and responses, so, some of the main design choices have been taken in order to minimize the response time and optimize the expenditure of the available resources.

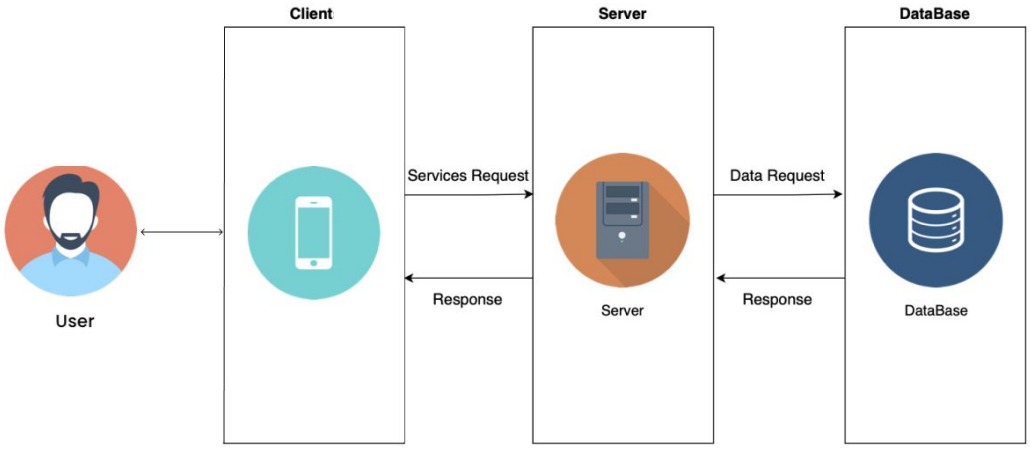


Figure 1.1: Representation of the three tier architecture

The client software (mobile application) acts in many cases only as an interface to the server. The server is used to manage the main application logic; moreover, it contains the management of some of the functions that users do not need to access, or the ones that require a high expenditure of resources. Some of the main purposes of the server are to send requests to the database (in particular, by communicating with its DBMS it can handle, when necessary, all the operations of reading, writing and uploading of data concerning users' account), to send requests to all the API used to retrieve data about cryptocurrencies and to compute predictions using a neural network.

The three tiers contain:

- **Tier 1:** The first tier contains the presentation logic for Users, in fact the Application contains the interfaces to authenticate, to create a wallet inserting the desired cryptocurrencies and to analyze their historical and future trends through interactive plots.

- **Tier 2:** Here is deployed the logic to retrieve and store data on the DB, which means all the cryptocurrencies in users' wallets. The server also contains executables to retrieve historical data about the cryptocurrencies, and to compute their predicted trends running a Deep learning model.

- **Tier 3:** The DB Server is conceived to execute a non-relational DBMS. The choice of a non-relational DB allows a lighter computation than a relational one, without risks for data integrity, because data are not split as in a relational DB. Moreover it allows horizontal scalability of data and it is suitable for large recorded amounts of data that have to be retrieved.

2.2 Components view

In this paragraph are presented the logical and physical components of the system, how they are structured and the way in which they interact with each other.

In particular, the components will be described first in a general way in order to highlight how they are integrated into the main structure of the system and how they are connected with the others. Then, a more detailed description is provided for each one in order to identify their subcomponents and their logical features.

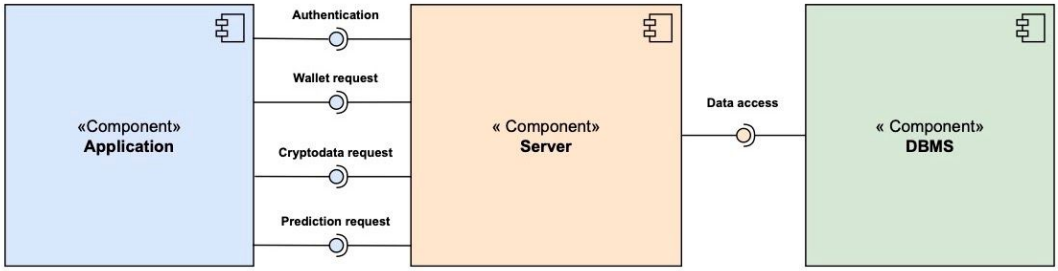


Figure 2.1: Application's high level components

The system is based on three main physical components: Application (UI), Server and Database. The entire system depends on the server component which is the core of the application. Its purpose is to handle the main logic features such as the users' authentication, the management of communications with the APIs and the management of the neural network which predicts future trends of cryptocurrencies.

The users' data are stored into a database which is managed by a DBMS that allows the creation, manipulation and querying of data.

The UI is characterized by a mobile application, which presents data retrieved by the server and allows users to manage their account.

As mentioned before, the communication between components is managed by the exchange of requests and responses. In particular, The UI communicates with the server in order to get the main application services, while the server communicates with the DBMS in order to upload, get, or edit the data. More details regarding the logical subcomponents of each main component are described below:

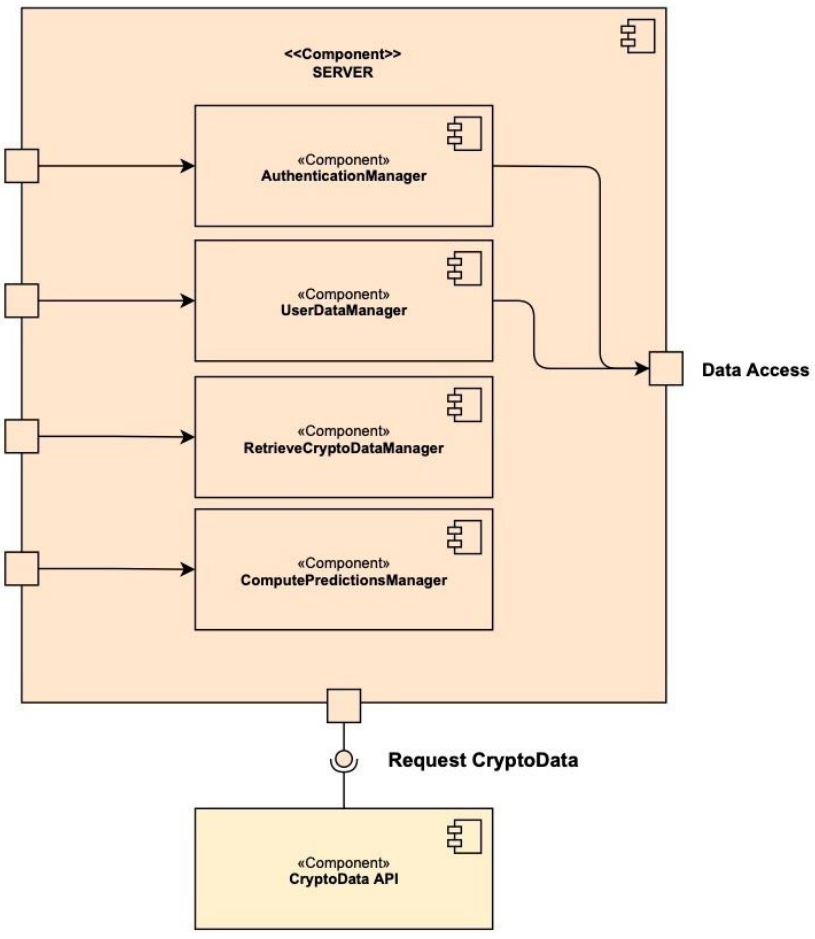


Figure 2.2: Explosion of the Server component

- **AuthenticationManager**: The purpose of this component is to collect the requests coming from the Application

containing the credentials inserted by users and to check for their correctness. Then, it sends back either a positive or negative response which determines the possibility to log into the system and the access to the services made for them.

- **UserDataManager**: When it receives a request coming from the Application concerning user's data it sends a request to the DBMS component in order to get them. Then, it sends back a response containing the information requested.
- **RetrieveCryptoDataManager**: Its task is to query the API in order to get data about the cryptocurrencies requested by the user in the Application and return them back.
- **ComputePredictionsManager**: The purpose of this component is to compute predictions on future trends of cryptocurrencies using a neural network and return them to the Application.

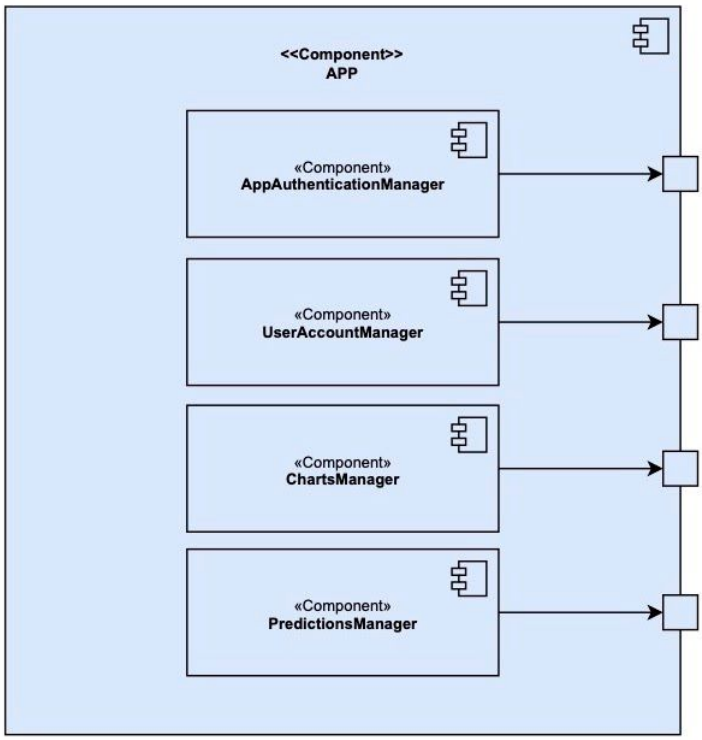


Figure 2.3: Explosion of the APP component

- **AppAuthenticationManager**: This component allows users to input their own credentials in order to log or register into the system. Then, it sends a request to the server component and waits for its response.
- **UserAccountManager**: It sends a request to the server component to get the data about the user's wallet and waits for its response.
- **ChartsManager**: This component sends a request to the server component to get the data about users' cryptocurrencies, it also contains an interface to allow data filtering through interactive plots.
- **PredictionsManager**: This component sends a request to the server to get the data about users' cryptocurrencies' predicted trends, and contains also an interface to allow data filtering through interactive plots.

2.3 Runtime view

This section describes the flow of interaction between the components of CryptoSimulator using sequence diagrams.

In particular four use-cases are taken into consideration explaining the manner in which the components are used:

- **User Authentication:** The sequence diagram in the figure below shows the flow of the user authentication.

  The AppAuthenticationManager sends the user's credentials to the AuthenticationManager on the Server, which checks their correctness communicating with the DBMS and returns a positive message in case of success, a negative one otherwise.

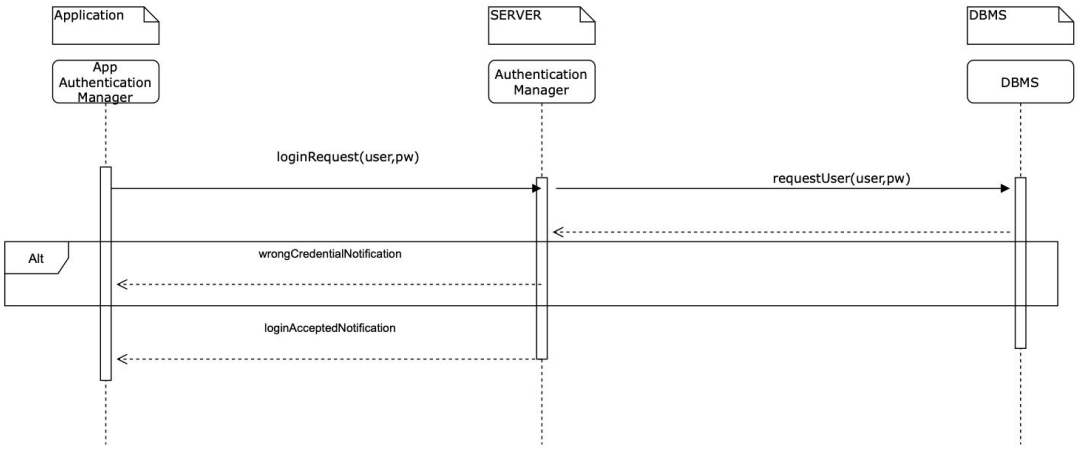


Figure 2.4: User authentication runtime view.

- **Wallet creation:** The sequence diagram in the figure below shows the flow of the wallet creation.

  The UserAccountManager asks the user to insert a list of pairs <cryptocurrency_name, amount>, then sends it to the UserDataManager on the Server, which forwards them to the DBMS that updates the database.

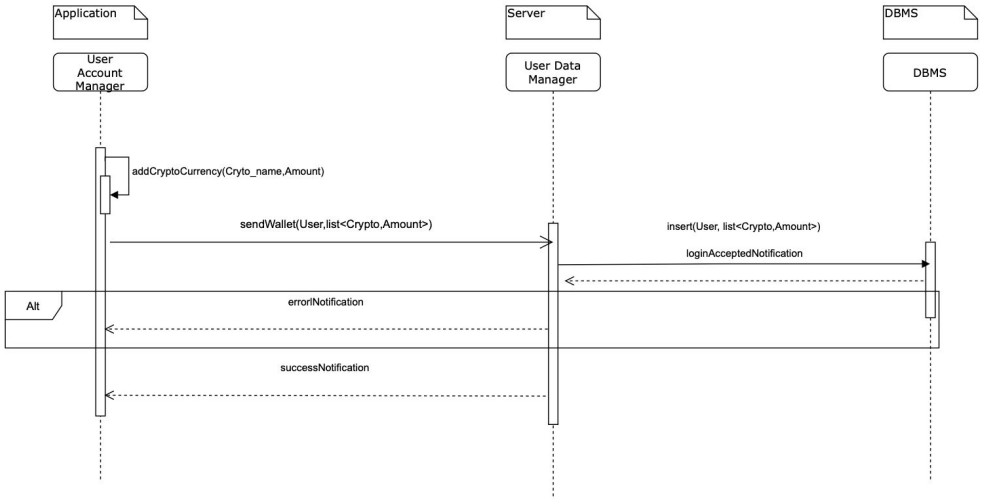


Figure 2.5: Wallet creation runtime view.

- **Charts retrieval:** The sequence diagram in the figure below shows the flow of the charts retrieval.

  The ChartManager sends a request to the RetrieveCryptoDataManager on the Server passing the list of cryptocurrencies in the user's wallet. The RetrieveCryptoDataManager then calls the Messari API to retrieve historical data for each cryptocurrency, collecting them into a list to return to the application.

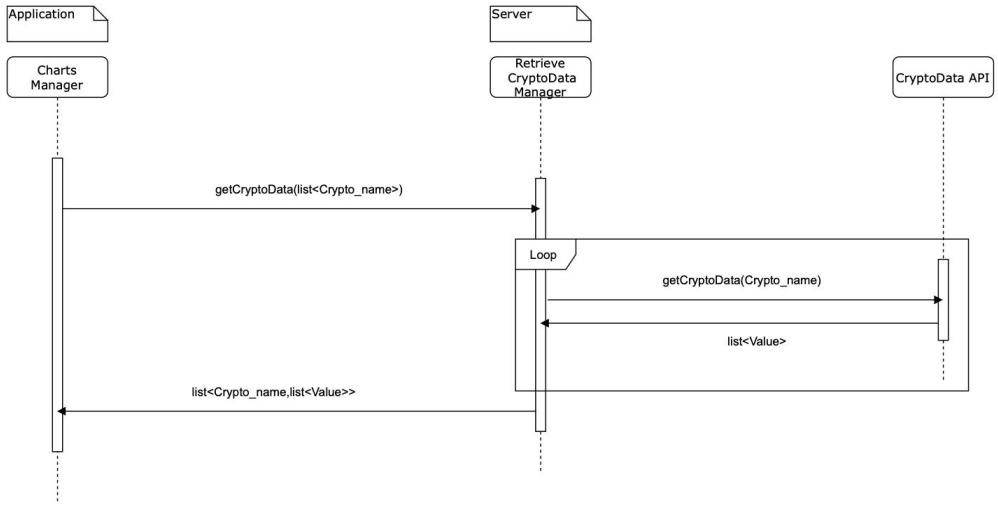


Figure 2.6: Charts retrieval runtime view.

- **Prediction retrieval:** The sequence diagram in the figure below shows the flow of the charts retrieval.

  The PredictionManager sends a request to the ComputePredictionsManager on the Server passing the list of cryptocurrencies of the user's wallet. The RetrieveCryptoDataManager then calls the Messari Api to retrieve historical data foreach cryptocurrency, collecting them into a list to return to the ComputePredictionsManager

which computes the predictions for each one of them and finally returns the list to the Application .

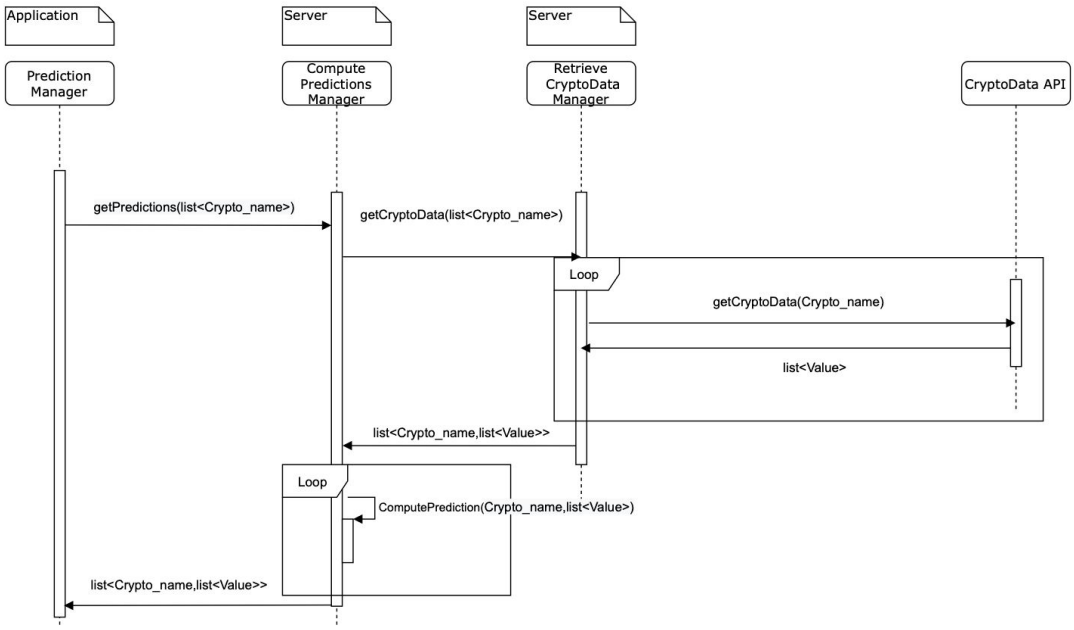


Figure 2.7: Predictions retrieval runtime view.

2.4 Component interfaces

In this paragraph are described all the interfaces of each component of the system. The methods presented are the ones considered the most relevant in order to exploit the main critical aspects of application behavior, components dependencies and internal interactions.

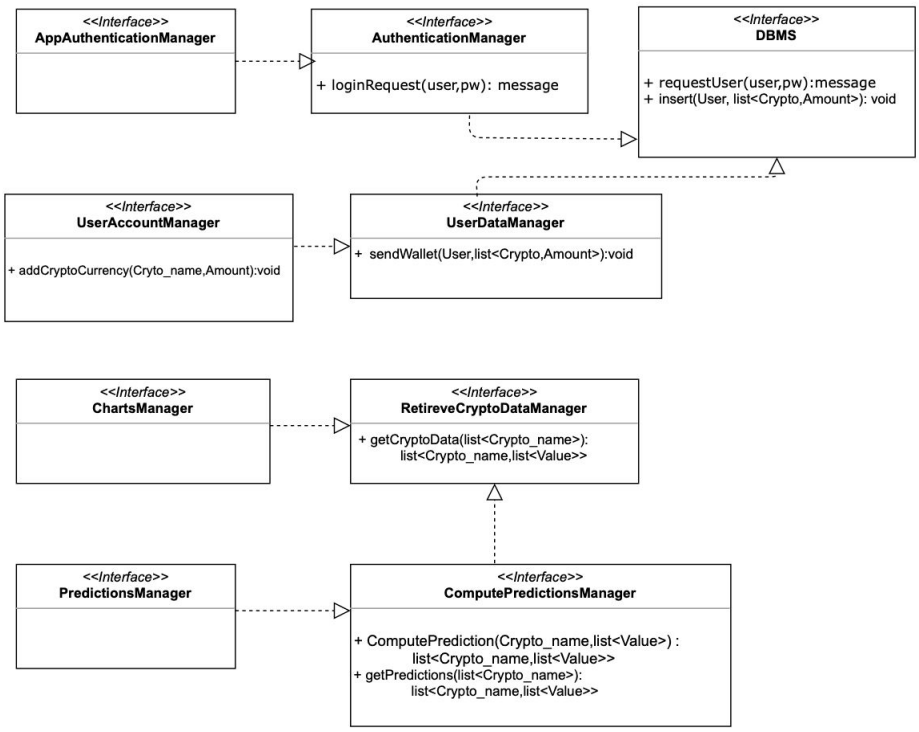


Figure 2.8: interface diagrams of the system.

- **loginRequest(user,pw): bool :** the method in the AppAuthenticationManager calls the loginRequest function of the AuthenticationManager on server side that checks and returns an exception through a bool value to warn user on the incorrect submission.
- **addCryptoCurrency(Cryto_name,Amount): void :** the method in the UserAccountManager adds a pair <Cryto_name, Amount> to the user's wallet (a list containing all his/her cryptocurrencies)
- **sendWallet(User,list<Crypto,Amount>): bool :** the method in the UserAccountManager calls the sendWallet function in the UserDataManager on server side that interacts with the DBMS to update user's wallet and it returns a positive message in case of success, otherwise it raises an exception. This exception is thrown through a bool value to warn the user that the wallet cannot be updated.

- **insert(User, list<Crypto,Amount>): bool :** the method in the UserDataManager calls the *Insert* function in the DBMS to update the database and returns a positive message in case of success, otherwise it raises an exception. This exception is thrown through a bool value to warn the user that the wallet cannot be updated.
- **requestUser(user,pw): bool :** the method in the AuthenticationManager calls the requestUser function in the DBMS to check the user's credentials and returns a positive message in case of success, otherwise it raises an exception. This exception is thrown through a bool value to warn the user of his/her wrong credentials .
- **getCryptoData(list<Crypto_name>): list<Crypto_name,list<Value>> :** the method in the ChartManager and the one in ComputePredictionManager calls the getCryptoData function in the RetrieveCryptoDataManager (passing a list of cryptocurrencies) which returns a list of pair <Crypto_name, list<Value>>.
- **ComputePrediction(Crypto_name,list<Value>) :list<Crypto_name,list<Value>> :** the method in the ComputePredictionsManager runs a recurrent neural network on the server side and returns a list of pair <Crypto_name, list<Value>> (the predicted data).
- **getPredictions(list<Crypto_name>): list<Crypto_name,list<Value>> :** the method in the PredictionManager call the getPredictions function in the ComputePredictionsManager (passing a list of cryptocurrencies) which returns a list of pair <Crypto_name,list<Value>>.

2.5 Other design decision

CryptoSimulator has an artificial intelligence engine, in particular it runs a Recurrent neural network on the Server that takes as input 500 historical values of a cryptocurrency (the daily close value) and returns 96 values which represent the prediction of the future 96 daily close values.

The RNN consists in an LSTM layer followed by a fully-connected layer with 40901 parameters :

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 100)	40800
dropout_2 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 1)	101
activation_2 (Activation)	(None, 1)	0

Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0

Figure 2.9: Summary of the RNN model

The training phase consisted of 100 epochs, the loss function used is the mean square error and the final error of the model is 0.0352.

The following plot shows a comparison between the model's prediction and the ground truth.

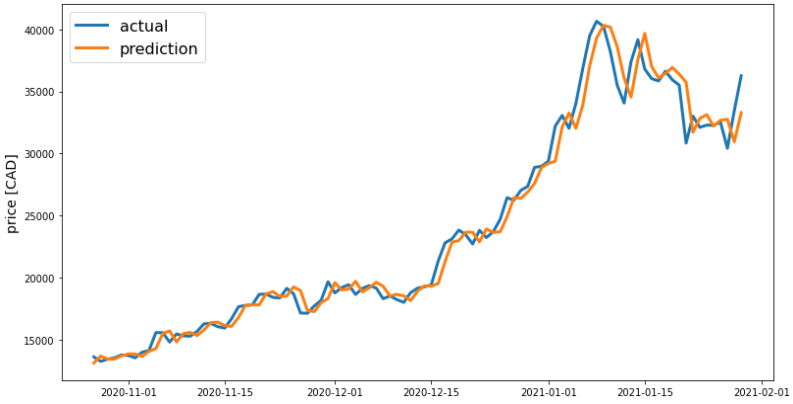


Figure 2.10: Model prediction and ground truth comparison.

3. Requirements traceability

This chapter explains which of the components are used to meet the requirements defined (in order to achieve the prefixed goals). It also contains a general description on how each component interacts with the others.

- **R1:** The System allows users to authenticate into the application.

  Mobile Application: AppAuthenticationManager → Server: AuthenticationManager → DataBase

- **R2:** Users are allowed to retrieve information regarding the preferred cryptocurrencies.

  Mobile Application: ChartsManager → Server: RetrieveCryptoDataManager → CryptoData API

- **R3:** The system builds predictions on future trends of the cryptocurrencies.

  Mobile Application: PredictionManager → Server: ComputePredictionManager

- **R4:** The application presents data to users in the form of interactive graphs and plots.

  Mobile Application: ChartsManager

- **R5:** Users are allowed to interact with graphs and plots by setting different time granularities and by visualizing specific periods' values.

  Mobile Application: ChartsManager

- **R6:** The system allows users to create and manage their own wallet by adding, editing and deleting their cryptocurrencies.

  Mobile Application: UserAccountManager → Server: UserDataManager → DataBase

- **R7:** The system allows users to visualize the current value of their wallet.

  Mobile Application: ChartsManager

- **R8:** The application allows users to visualize future predictions on how their wallet value may evolve.

  Mobile Application: ChartsManager

- **R9:** The system allows users to store information on the cloud in order to recover them from multiple devices.

  Mobile Application: UserAccountManager → Server: UserDataManager → DataBase

4. User interface Design

In this chapter are presented and described the mockups of the main components of the mobile application. For each of them both the dark and light mode are shown.

Figure 4.1: Historical data presentation mockups

In the picture above is shown how the cryptocurrency's historical data is presented to the users. For each cryptocurrency an interactive chart is rendered. Users can modify the data time

granularity and hide/show the price of a specific day by swiping on the chart itself.

Figure 4.2: Predictions page mockups

In the picture above is shown the page relative to the cryptocurrencies in-depth information. In particular, for each asset is shown a section relative to its highest and lowest prices for the selected time granularity. On the bottom part there are charts relative to both the historical and predicted trends of the current asset. By swiping left or right users can navigate between the different cryptocurrencies of their wallet.

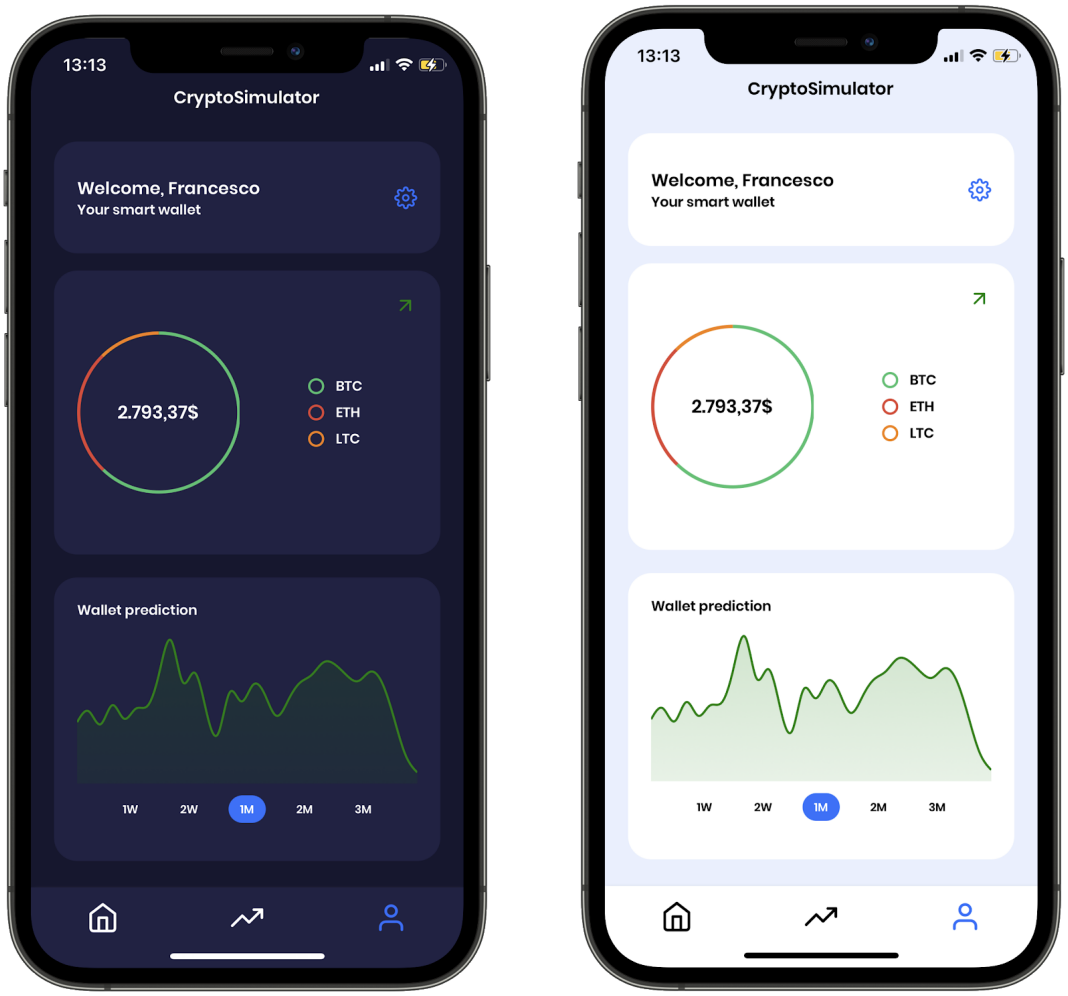


Figure 4.4: Wallet page mockup

In the Account page, users can visualize their own wallet. In particular, all the cryptocurrencies stored are shown as a pie chart and the current total value of the wallet is presented. Moreover, in the bottom part, is shown a chart representing the prediction computed by the AI of the future trend of the wallet's total value. Users can either modify its time granularity or retrieve information concerning specific days by swiping on the chart itself.

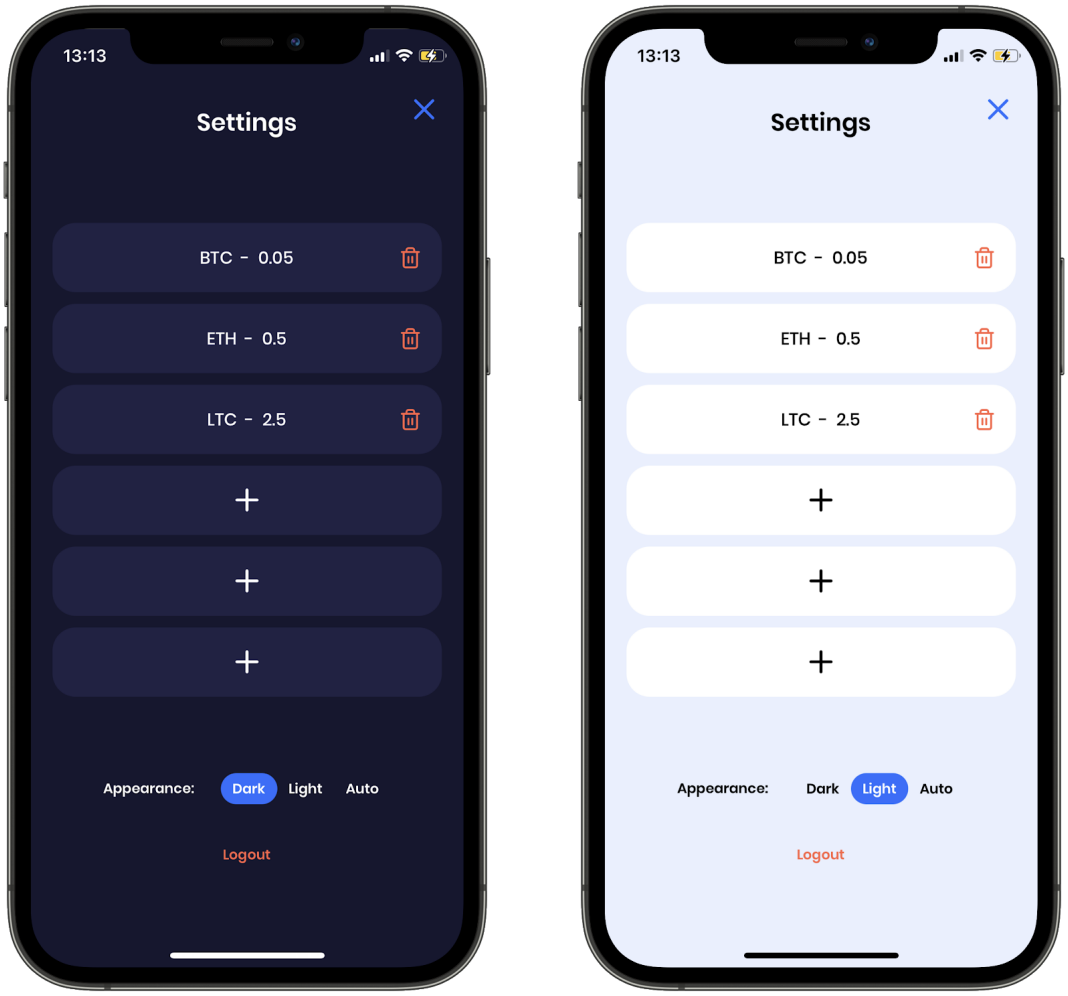


Figure 4.3: Settings page mockups

In the Setting Page Users can edit their own wallet by adding, modifying or deleting the cryptocurrencies. In addition, it is possible to modify the Application's appearance by setting it either to the dark and light mode. Finally, users can logout by clicking on the corresponding button.

5. Flow of Implementation and integration

This chapter aims to explain all the implementation and integration choices made to develop the whole system, such as the main implementation strategies, the technologies involved and the development, deployment and integration of the Application's main components.

5.1 Implementation and integration strategy

The implementation flow mainly follows a bottom-up strategy, which means that all components will be firstly implemented and tested and after integrated and re-tested in their own subsystem.
Thanks to the bottom-up strategy, the development of the Server component and the Mobile Application with each of their subcomponents was made in parallel.
The AuthenticationManager, the UserDataManager, the RetrieveCryptoDataManager and the ComputePredictionsManager were developed and tested independently and then merged together in the Server.

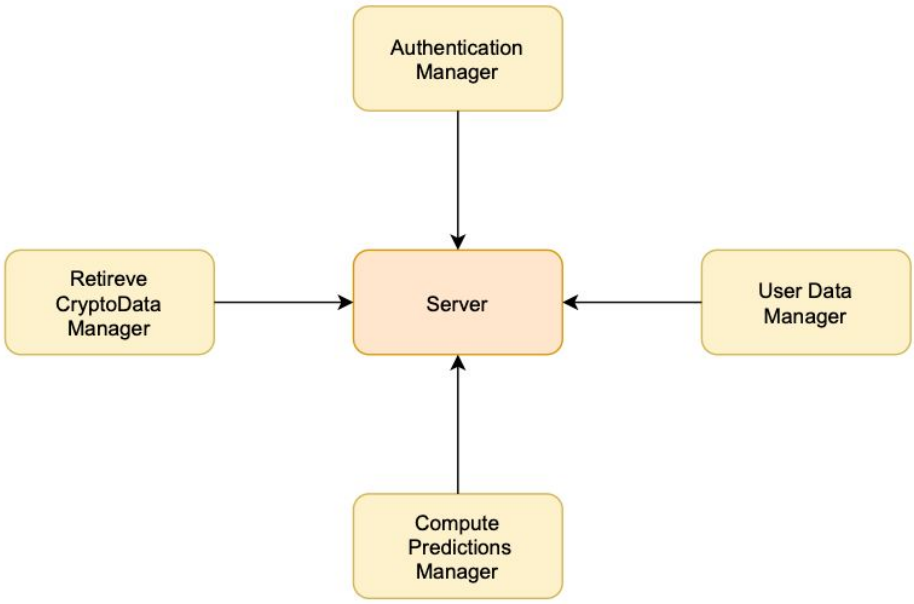


Figure 5.1: Server's implementation flow

Also the AppAuthenticationManager, the UserAccountManager, the ChartsManager and the PredictionsManager were developed and tested on their own and then merged together in the Application.

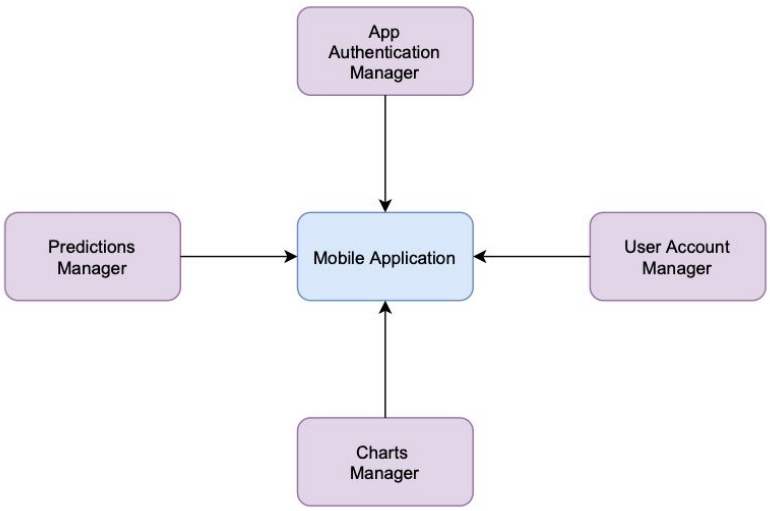


Figure 5.1: Mobile application's implementation flow

Finally all the communications between the Server and the Application were developed and the entire system was tested as a whole.

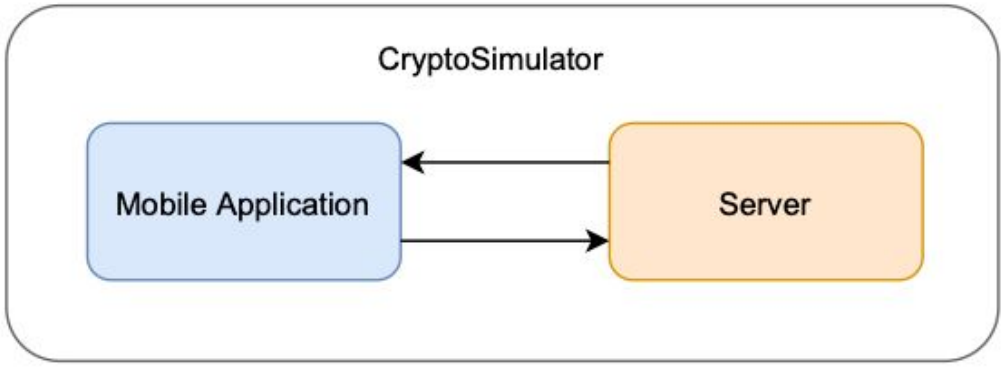
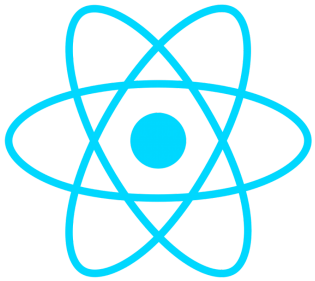


Figure 5.3

5.2 Technologies Involved

In this section are listed and described all the technologies used for the application's development such as the programming languages, the frameworks and the SDKs used.

- **React Native:** The React Native Framework has been used to develop the whole mobile application. The reason why it has been chosen is that it is a cross-platform framework, therefore it provides the tools to develop an application for both the IOS and Android Operating Systems.



- **Expo:** For what concerns the mobile application development, In addition to React Native's components, it has been decided to also use the Expo SDK, in order to have access to an additional variety of libraries and APIs for both the IOS and Android Operating Systems. Moreover, it provides the possibility to easily deploy and test mobile applications online.

- **Firebase:** Firebase has been chosen as the main system development platform to build, deploy and host all the application's main features. In particular, it provides APIs for some of the main functionalities such as Authentication, DataBase creation and management and server's cloud functions.

- **Google Cloud Platform:** As Cloud Computing Service Provider it has been chosen to use the Google Cloud Platform. In particular, it provides reliability, security, automatic scalability and the possibility to integrate in a single project all the components used to develop the application (such as FIrebase, Server hosting, and the AI Platform).

Google Cloud Platform

- **TensorFlow:** TensorFLow has been used as the main framework to build and test the AI engine of the application. In particular it has been used to develop, train and evaluate the Recurrent Neural Network Model used for predicting the cryptocurrencies' trends.

5.3 External APIs

The main APIs used to develop the application are listed and described below:

- **Messari API:** It provides all the information on the cryptocurrencies used in the application such as assets' prices and trends.

- **Crypto Icons API:** It has been used to retrieve the Icons of the cryptocurrencies supported by the mobile application. Its use is limited only to the development of the UI of the mobile application.

5.4 Mobile Application

As mentioned before, the mobile application has been developed using the React Native framework.

The figure below shows the way in which components are nested in the application and a brief description of each one is provided:

- **App:** App is the main component of the application, when mounted calls the Firebase Authentication API to check if any user is logged in the system (the session is set as

persistent, so once entered the user needs to explicitly logout).

If no user is logged, App renders the AuthenticationPage component, otherwise it renders the main core of the application, managed by a TabNavigator with three screens (components) : AccountPage, Charts, Predictions.

As soon as a user enters the application, App calls the Firebase Firestore API to retrieve his/her wallet and performs all the server's function calls to retrieve also every cryptocurrencies' data (historical and predicted) in order to pass them to its children.

Every latency due to the server requests and responses are managed with a loader, communications between subcomponents are managed via props and callbacks, finally shared preferences are also used to save into the application's memory some information such as the color scheme (light/dark).

- **AuthenticationPage:** AuthenticationPage contains a StackNavigator with two screens : SignUp and SignIn, to allow the user either to log or register into the system.

- **SignUp:** SignUp manages the registration phase of a user into the system, in particular it gets all the information needed (username, email, password) performing controls with security rules. When the user submits the sign up form , this component calls the Firebase Authentication API and based on the response, shows an error message or redirects the user to the App.

- **SignIn:** SignUp manages the login phase of a user into the system, in particular it gets all the information needed (email, password), calls the Firebase Authentication API and based on the response, shows an error message or redirects the user to the App.

- **AccountPage:** AccountPage contains a StackNavigator with two screens : Account and Settings, to allow the user to navigate between his/her account and the settings page.

- **Account:** Account is a screen that shows to the user his/her wallet with its total price value and a plot with its predicted trends (sum of predicted trends of each cryptocurrency in the wallet).

  Plots are shown using the react-native-svg-chart library, and in addition, the prediction plot is wrapped with a PanResponder to make it interactive.

- **Settings:** Settings is a screen that allows users to update their wallet adding or removing cryptocurrencies, therefore calling the Firebase Firestore API to update user's wallets in the database. This component also allows users to change the color scheme (light/dark) or logout from the system.

- **Charts:** Charts is a screen that shows to the user a plot for each cryptocurrency in his/her wallet with their historical data.

  Plots are shown using the react-native-svg-chart library, each one is wrapped with a PanResponder to make them

interactive and in addition a panel with some buttons allows the user to choose different time granularities for the charts.

- **Predictions:** Predictions is a screen that shows to the user a plot for each cryptocurrency in his/her wallet with their predicted future trends.

  Plots are shown using the react-native-svg-chart library, each one is wrapped with a PanResponder to make them ineffective and in addition a panel with some buttons allows the user to choose different time granularities for the charts. This screen also shows the highest and lowest predicted price of the cryptocurrencies.

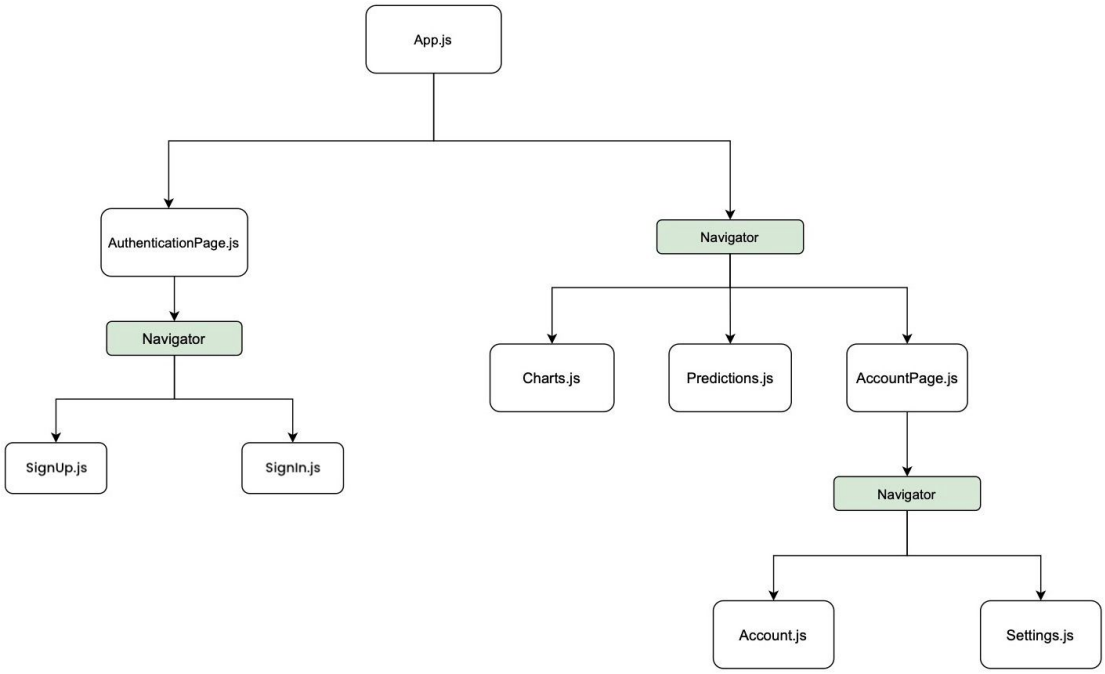


Figure 5.4: Mobile application components' nesting

5.5 Server

For the Development and Hosting of the server has been used the *Firebase Cloud Functions*, *Google Cloud* Functions and *Google AI Platform* APIs with either Node.js and Python programming languages.

In particular, the server is composed of two cloud functions, which respect the interfaces defined in the Chapter 2:

- **getCryptoData():** The function takes as input the name of a cryptocurrency, a start date, an end date and an interval, and launches a request to the *Messari* API to retrieve all the information on the selected cryptocurrency. If the request is fulfilled, the API returns the corresponding data, including the trend for the selected time window of the selected cryptocurrency. Then, the server returns to the client all the requested information. The function has been written using the Node.js programming language.

- **getPredictedCryptoData():** The function's purpose is to take as input a cryptocurrency and perform all the operations necessary to retrieve the data predicted by the Recurrent Neural Network. In particular, it sends a request to the API to retrieve the past values of the cryptocurrency and performs some preprocessing operation on the data such as loading it in the correct data structure or performing data normalization. Then, it sends a request to the *Google AI Platform's* API with the relative data in order to run the deep learning model and compute the predictions. If the request succeeds, it returns to the client the corresponding information. The function has been written using the Python programming language.

For what concerns the authentication of the users and the storing of their information (such as the wallet), *Firebase* and the APIs it provides have been used:

- **Authentication:** The authentication mechanism has been implemented by using the *Firebase authentication* API. Currently, the only method supported is the signup/login with email and password.
- **Data Storage:** The storage of the user's data such as the wallet (which contains the selected cryptocurrencies and the relative quantities) has been implemented by using the *Firebase Cloud Firestore* API.

The Recurrent Neural Network has been Deployed by using the *Google AI Platform*. In particular, after the implementation and training of the model, it has been saved on a *Google Cloud Storage Bucket*. At each request, the API loads the model and its weights and computes the predictions on the specific cryptocurrency.

6. Testing

This chapter's purpose is to explain and describe all the testing procedures that have been performed in order to check the correctness of the source code of the entire system (backend and frontend). Many use cases have been considered in such a way to cover all the main scenarios in which the application is supposed to operate.

6.1 Static Analysis

The first step to improve the code quality has been to start using static analysis tools in order to highlight the presence of any errors in the source code.

In particular, it has been decided to perform two kinds of static analysis, by using two different frameworks, each of them focused on specific aspects such as integrity, correctness and consistency of the source code:

- **ESLint:** It has been used as a Linter Analyzer to catch common errors such as unused code and to help improve code consistency. The framework has been used on all the components of the mobile application and on the backend functions.

- **Flow:** It has been used as a Type Checking to ensure that all the constructs passed to a function match what the function was designed to accept, preventing passing a string to a

counting function that expects a number, for instance. Also in this case it has been used both on the Mobile Application's components and on the server side functions.

6.2 Unit Testing

The purpose of the Unit test is to check the behaviour of small units of the system, therefore functions and classes.

This type of test on CryptoSimulator's system has been made using Mocha, a JavaScript test framework running on Node.js.

The Testing procedure consisted in developing several use cases to test functions both on Server and Mobile application.

The unit test of the mobile application has been made on functions used in the components to process data coming from the APIs and the user's inputs, in particular the behaviour of each method has been checked to be the desired one.

```
parsePrice integrity test 1
  #parsePrice()
    ✓ should return "" when the value is null

parsePrice integrity test 2
  #parsePrice()
    ✓ should return "" when the value is not a number

parsePrice result test
  #parsePrice()
    ✓ should return "1.679,76$" when the value is 1679.76

isEquivalent integrity test 1
  #isEquivalent()
    ✓ should return false when at leat one argument is null

isEquivalent integrity test 1
  #isEquivalent()
    ✓ should return false when at leat one argument is not a dictionary

isEquivalent result test 1
  #isEquivalent()
    ✓ should return true if the arguments are equals

isEquivalent result test 2
  #isEquivalent()
    ✓ should return false if the arguments are not equals
```

Figure 6.1: Mobile application's unit testing

The unit test of the server has been made on the functions that are called by the application through HTTP requests, in particular the use cases consisted in different arguments of the requests made by the application component.

```
getCryptoData integrity test 1
  #getCryptoData()
    ✓ should return [] when the assets list is empty

getCryptoData integrity test 2
  #getCryptoData()
    ✓ should return [] when the start period list is not a valid date

getCryptoData integrity test 3
  #getCryptoData()
    ✓ should return [] when the end period list is not a valid date

getCryptoData integrity test 4
  #getCryptoData()
    ✓ should return [] when the selected interval list is null

getCryptoData result test
  #getCryptoData()
    ✓ should return a 200 statusCode if all the input arguments are corrected
```

Figure 6.2: Server's unit testing

6.3 Integration Testing

The purpose of the integration test is to check the behaviour of components combined together and in particular the interaction between them.

As mentioned in paragraph 5.1, the development strategy has followed a bottom-up approach and therefore, after the unit testing each subcomponent, the communication between had to be tested.

For this type of testing on CryptoSimulator's system Mocha has been used (as for the unit test), in particular the goal was to check if data were passed correctly between components of the Application and the Server and vice versa.

6.4 End-To-End Testing

The purpose of end-to-end (E2E) tests is to check if the system works as expected on a device from the user perspective.

This procedure consists in listing all the possible interactions the users can have with the system and checking if it works as expected.

In particular, given the system requirements, the use-cases used to test the user-system interaction are listed below:

- **R1: The System allows users to authenticate into the application.**

  Use case: User enters the application and is asked to login.

1. **Test case:** User enters the right credentials (email and password) in the application.

   Result: Login successful, the main page of the application is shown.

2. **Test case**: User enters the wrong or incomplete credentials (email and password) in the application.

   Result: An error message is shown, the application requires to repeat the login procedure.

- **R2: Users are allowed to retrieve information regarding the preferred cryptocurrencies.**

  Use case: User opens the page of the application that shows historical data of his/her preferred cryptocurrencies.

1. **Test case**: User has cryptocurrencies in his/her wallet.

   Result: A list of interactive plots is shown in the screen.

2. **Test case**: User does not have cryptocurrencies in his/her wallet.

   Result: A message is shown on the screen pointing up the absence of cryptocurrencies in the wallet, suggesting to add them in the settings page.

- **R4: The application presents data to users in the form of interactive graphs and plots.**

  R5: Users are allowed to interact with graphs and plots by setting different time granularities and by visualizing specific periods' values.

  Use case: User opens the page of the application that shows historical data or the page with predictions of his/her preferred cryptocurrency (already added in the wallet).

1. **Test case**: User clicks the buttons to change the granularities of the plots.

   Result: The application rerenders the plots showing different time granularity charts.

2. **Test case**: User moves his/her finger on the plots.

Result: The application moves the tooltips on the charts and shows the price value of cryptocurrencies corresponding to the time stamps chosen by the user.

- **R6: The system allows users to create and manage their own wallet by adding, editing and deleting their cryptocurrencies.**

  Use case: User opens the settings page and clicks the button to add a new cryptocurrency.

1. **Test case**: User enters correctly the cryptocurrency's name and value.

   Result: The chosen cryptocurrency is added in the user's wallet, the wallet is shown in the account page and charts about historical and predicted data of that cryptocurrency are shown in the other pages.

2. **Test case**: User entered incorrectly or partially the cryptocurrency's name and value.

   Result: The application doesn't allow the cryptocurrency to be added in the user's wallet, each page shows a message pointing up the absence of cryptocurrencies in the user's wallet.

- **R7: The system allows users to visualize the current value of their wallet.**

  R8: The application allows users to visualize future predictions on how their wallet value may evolve.

  Use case: User opens the account page

1. **Test case**: At least one cryptocurrency is in the user's wallet.

   Result: The application shows a pie chart with the user's cryptocurrencies and an interactive plot with the predicted trend of his/her wallet value.

2. **Test case**: No cryptocurrencies are in the user's wallet.

   Result: The application shows a message pointing up the absence of cryptocurrencies in user's wallet

- **R9: The system allows users to store information on the cloud in order to recover them from multiple devices.**

  Use case: User enters the application for the first time and performs the registration procedure.

1. **Test case**: User enters his/her credentials correctly (username, email and password) in the application.

   Result: Login successful, the main page of the application is shown.

2. **Test case**: User enters his/her credentials incorrectly or partially (username, email and password) in the application.

   Result: An error message is shown, the application requires to repeat the registration procedure.

3. **Test case**: User enters his/her credentials already used by other users in the application.

   Result: An error message is shown, the application requires to repeat the registration procedure.

   Use case: User logs out from the system and performs the login in another device.

1. **Test case**: User performs the login with his/her credentials.

   Result: The application shows correctly all the data about that specific user's wallet and cryptocurrencies.

7. References

- **Crypto Icons:** Crypto Icons API
- **Expo:** Expo.io
- **ESLint:** ESLint - Pluggable JavaScript linter
- **Firebase:** Google Firebase
- **Flow:** Flow: A Static Type Checker for JavaScript
- **Google Cloud Platform:** Google Cloud: Cloud Computing Services
- **Google Cloud Functions:** Google Cloud Functions
- **Google AI Platform:** Google Cloud AI Platform
- **Messari:** Crypto Research, Data, and Tools
- **Mocha:** Mocha - the fun, simple, flexible JavaScript test framework
- **React Native:** React Native · A framework for building native apps using React
- **TensorFlow:** TensorFlow