

# *PROJECT REPORT*

## *PT04*

A crypto-currency exchange application: CryptoJoint

*Laurens Kubat (CP) & Ferran van der Have & Reinier Sanders &  
Wout van den Berg & Wouter van Battum*

*Radboud University | Comeniuslaan 4, 6525 HP Nijmegen*

## Table of Contents

<b>INTRODUCTION .....</b>	<b>3</b>
GENERAL DESCRIPTION .....	3
GOALS OF THE SYSTEM .....	3
USERS OF THE SYSTEM .....	3
<b>DESCRIPTION .....</b>	<b>4</b>
FOCUS ON PROPERTIES .....	4
PRODUCT JUSTIFICATION.....	4
SPECIFICATIONS.....	5
<b>DESIGN .....</b>	<b>6</b>
GLOBAL DESIGN .....	6
DETAILED DESIGN .....	6
<i>Classes</i> .....	6
DESIGN JUSTIFICATION .....	11
<b>PROJECT MANAGEMENT .....</b>	<b>12</b>
DIVISION OF TASKS.....	12
PROGRESSION .....	12
<i>Week 1:</i> .....	12
<i>Week 2:</i> .....	12
<i>Week 3:</i> .....	12
<i>Week 4:</i> .....	12
<i>Week 5:</i> .....	12
<i>Week 6:</i> .....	12
ORGANIZATION .....	12
<b>EVALUATION .....</b>	<b>13</b>
LAURENS KUBAT.....	13
REINIER SANDERS .....	13
FERRAN VAN DER HAVE .....	14
WOUT VAN DEN BERG.....	15
WOUTER VAN BATTUM .....	15

## Introduction

### General description

Our application let the users practice with the trading of cryptocurrency. Users are able to trade on different exchanges, with crypto coins. When downloaded, the users get \$10,000 fake dollars to trade with on the exchanges.

### Goals of the system

The goal of our application is to let the customers get to know the ins and outs of cryptocurrency trading and practice with the trading.

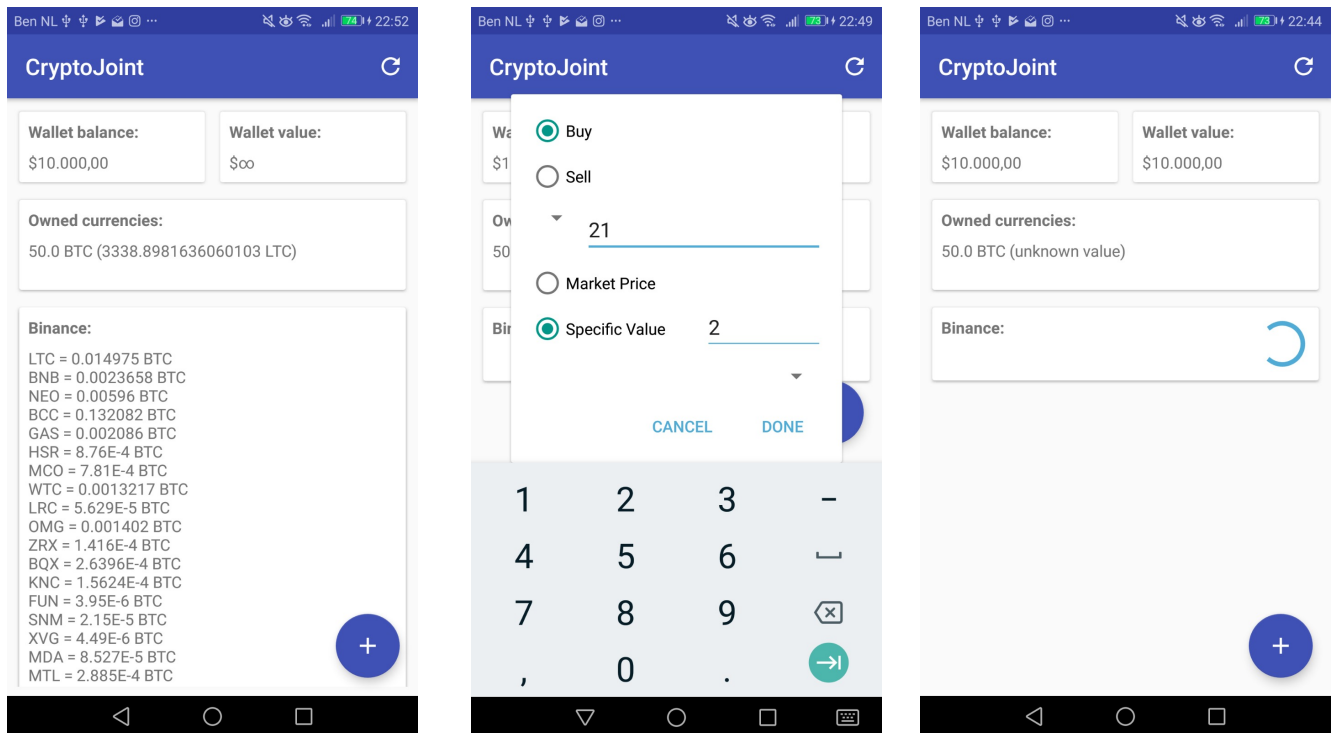
### Users of the system

We assume that the users of our application are familiar with cryptocurrency but not quite familiar with trading.

The users of our application are interested in our application, so they can practice trading with fake money and if they think they have mastered the skill of trading with cryptocurrency they can use real money.

## Description

### Focus on properties



The box named 'Wallet balance:' is the current balance from the wallet, the box called 'Wallet value:' is the current value from the wallet. State the user has bought \$1,000 of a currency, then the wallet balance says \$9,000 dollar but the wallet value can differ on wherever the value of the currency is increasing or decreasing. In the box named 'Owned currencies:' are the currencies that the user owns shown. Also if the user wants to only watch the price and value of a currency, the currency is shown there. The box named 'Binance:' shows all the currencies that Binance offers.

In the most left screenshot one can see the screen from a user who bought 50.0BTC. The user has chosen to represent this value as another currency. Therefore, the value of LTC is stated next to the BTC value. The user is able to buy or sell a currency to click on the button on the bottom right with the plus-sign in it and it unfolds to the screenshot in the middle.

In the middle screenshot one can see the screen the user gets to see if they want to buy a currency. First, they have the option to buy or sell the currency they selected. Secondly, they choose the amount of currency they want to buy or sell. Thirdly, the user can choose to buy or sell the currency at the current market value or at a specific market value that the user can itself can choose.

In the rightest screenshot one can see the screen where the chosen currency is added to the 'Owned currencies:' box.

### Product justification

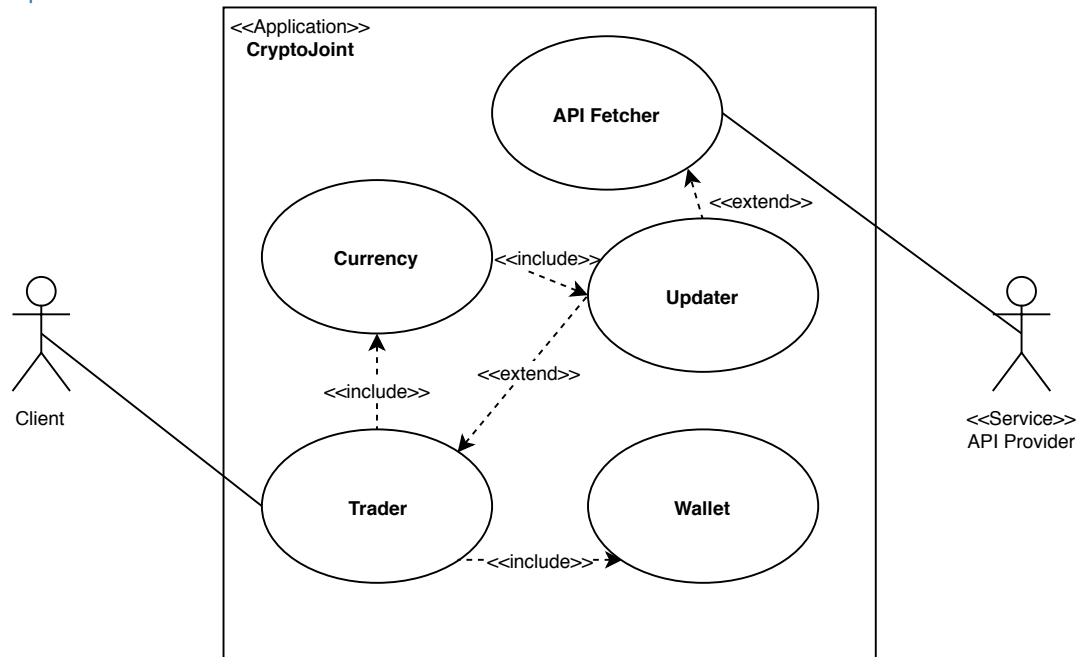
Our application is worth downloading because it gives someone the opportunity to learn how to trade with cryptocurrency. Also the usage and differences of exchanges.

A product that is similar to ours, is BUX. BUX is an application for the 'normal' stock market. However, BUX is only one example, but there are more of these apps. But our application is only one for crypto exchanges that we know of.

The new and innovative contributions of our application are:

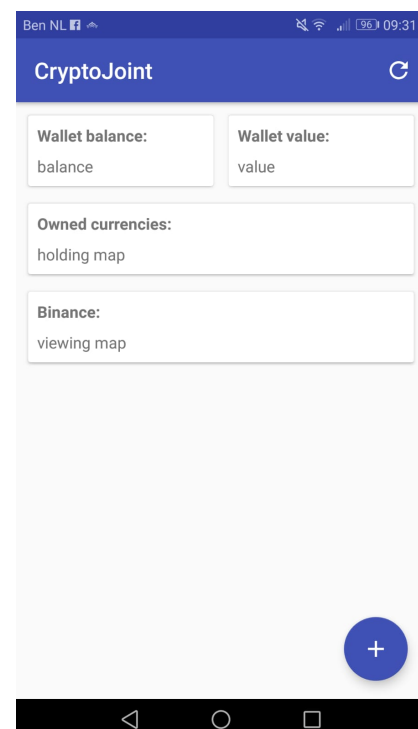
- Cryptocurrency trading
- The opportunity to watch a currency in a quick view

## Specifications



In this Use Case the underlying model from the front end of the application is shown. The user is able to trade currencies in our application, that happens in the Trade case. In the application the user is able to buy and sell currencies, that happens in the Trader component of the Use Case.

As shown in this screenshot to the right from the application in state of developing the Wallet balance box is a Holding Map, also the Wallet value box is a Holding Map, such as the Owned currencies box. And as shown the Binance box is a Viewing Map



## Design

### Global design

The five main components of our application are:

- Client
- Trader
- API Fetcher
- Updater
- Wallet

The Client component uses the `updateViews()` function from the MainActivity class. This component gets from the Wallet the `getBalance()` and sends to the Wallet the `setBalance(double balance)`. It also gets the `getHolding(String currency)` from the wallet and sets the holding from the Wallet with `setHolding(String currency, double amount)`.

The Trader component adds a holding to the wallet with `addHolding(String currency, double amount)`.

The API fetcher sends to the API Binance component a new Binance API. It also get the symbols from the exchange from the API Binance.

The Updater creates a new API fetcher. And gets the symbols from the exchange from the fetcher.

The Wallet component only sets the Client in the constructor, the Wallet does not need any other component except for the Client.

### Detailed design

#### Classes

##### *MainActivity*

This class extends AppCompatActivity and has seven global variables:

- Client client
- Trader trader
- Updater updater
- Private Context context
- Private Activity activity
- Private CoordinatorLayout layout
- Private PopupWindow popupwindow

The MainActivity class starts with a protected void `onCreate(Bundle savedInstanceState)`. At first it sets the variables context, activity and layout, also a Toolbar is created and a FloatingActionButton. Subsequently with an `onClick` void the popupwindow is set. After that this function, it creates a new client, also it initializes a new Updater updater. Then it checks if there is any data to load, if so it loads it immediately. After that it calls the function `updateViews()`.

The function `public void updateViews()` gets called whenever new trade information is received,. If so it updates the views.

The protected void `onPause()` pauses the application and saves the data.

The public boolean `loadData()` returns true if the data that is saved on the device is loaded successfully to the application.

The public void `saveData()` saves the data from the wallet and the trader to the device from the user.

### *Currency*

The Currency class implements Serializable. The class has two global variables:

- Private String name
- Private double value

The constructor needs a name and a value. The name is a currency's name and the value is the current value from the currency.

The getter public String getName() returns the name of the currency.

The getter public double getValue() returns the value of the currency.

The setter public void setValue(double value) sets the value of the currency.

### *APIFetcher*

The APIFetcher class has only one global variable:

- Private APIBinance binance

Nothing happens in the constructor of the APIFetcher class.

The getter public API GetFormat(Currency toBuy, Currency toSell, String Exchange) gets the format of an exchange API. Where the parameter toBuy is the currency which is to be bought, the parameter toSell is the currency which is to be sold, the parameter Exchange is the exchange on which to buy or sell. This function returns the object which inherits the API.

The getter public ArrayList<CurrencyTuple> getSymbols(String Exchange) returns the symbol of a CurrencyTuple.

The public APIBinance MakeBinanceAPI(String Endpoint) returns a new APIBinance with the String Endpoint.

### *APIBinance*

This class implements the interface API and has two global variables:

- Private String pair
- Private String url

The constructor needs a String pair and a String url, where pair is the pair from the currency and the symbol from that currency.

The getter public ArrayList<CurrencyTuple> getSymbols() returns the symbol from the currency.

The function public Double makeCall() makes a get-request for the price of a currency.

### *Updater*

The Updater class has two global variables:

- Private Map<PairTradeType,API> CurrentPairs
- Private APIFetcher fetcher

The constructor creates an Updater and does not need any parameters.

The function public void AddPair (String Endpoint, String Pair, String Exchange) adds a currency pair to the currency pairs list. It also adds support for different endpoints.

The getter public HashMap getUpdate() returns an updated version of the HashMap from currency pairs and their values.

## Client

The Client class has four global variables. These variables are a:

- Private ArrayList<CurrencyTuple> currencies
- Private Wallet wallet
- Private MainActivity mainActivity
- Private Trader trader

The constructor only needs a MainActivity.

The setter public void setTrader(Trader trader) sets the value of trader from the client.

The setter public void setWallet(Wallet wallet) sets the value of the wallet from the client.

The getter public Wallet getWallet() returns the value of the wallet from the client.

The setter public void setCurrencies(ArrayList<CurrencyTuple> currencies) sets all the currencies from the client and updates the Views from the MainActivity.

The getter public ArrayList<CurrencyTuple> getCurrencies() returns all currencies the client has.

The public void printCurrencies(Textview t) prints all the currencies from the exchanges.

The public void printCurrenciesInDollar(Textview t) is the Textfield that the values of the currencies, compared to dollars, need to be printed to.

The function public ArrayList<Currency> currencyValues(String currencyname) creates an ArrayList with all the CurrencyTuples with USDT as not owned. This means a list of all the dollar values of all the currencies. The String currencyname is the name of the currency to which the other currencies are compared to. This function returns an ArrayList of all the currencies with their dollar values.

The getter public Currency getCurrency(String currencyName) returns the object of currency with the given currencyName.

The public void buy(Currency currency, double amount) creates a buy trade with the double amount and the currency that needs to be bought and does this using the Trader class with the line trader.Trademaker(true, currency, amount, usdt, 0);. The currencies are bought with the money from the balance from the user.

The public void sell(Currency currency, double amount) creates a sell trade, where it sells the value of the double amount of the currency. This happens using the Trader class with the line trader.Trademaker(false, currency, amount, usdt, 0);.

The last option is to sell a currency and get another currency in return. This happens in the public void trade(Currency toSell, double amountSell, Currency toBuy, double amountBuy) function where toSell is the currency that needs to be sold, amountSell is the amount of that currency that needs to be sold, toBuy is the currency that needs to be bought and amountBuy is the value at which the client wants to conduct the trade, if this is equal to zero, then the trade is conducted at the next update. The currencies are traded using the Trader class with the line: trader.Trademaker(true, toSell, amountSell, toBuy, amountBuy);.



## Trader

The Trader class extends the java import TimerTask and has six global variables, these are a:

- Private Updater updater
- Private String exchange
- Private static int id
- Private ArrayList<Trade> trades
- Private Wallet wallet
- Private Client client

The constructor needs an Updater updater, this is the updater of the current trade. The constructor also needs a Client client, which is the user of the application. Finally, the constructor needs a String exchange, which is the exchange of the current trade. In the constructor of this class a new ArrayList is created and named trades, also is the public void start() is called. Also is the int id set to one.

The public void start() creates a new Timer called timer and the timer is scheduled for a specified task for repeated fixed-rate execution, the timer executes the function run() every two seconds.

With the public void removeTrade(Trade trade) a trade from the list of trades can be removed, just by a simple line "trades.remove(trade);" where trade is the trade that has to be removed.

The public void addWallet(Currency currency, double amount) adds an amount of a currency to the wallet of the user. Where currency is the currency which is to be added to the wallet and amount is the amount of currency that is to be added to the wallet. It does that by one line: wallet.addHolding(currency.getName(), amount);.

The public void updateTrader() changes the list of currencies that the trader has at the moment with the new list of currencies the trader has. It also updates all values using a for-loop and trade.updateValues(); where the function updateValues() is from the Trade class.

The public void doTrades() tries to conduct all trades in the list "trades". Using a for-loop.

The getter public ArrayList<Trade> getTrades() returns a list of all trades.

The getter public Wallet getWallet() returns the wallet of the client.

The getter public ArrayList<CurrencyTuple> getCurrencies() returns the currency list of the client.

The public void Trademaker(boolean buy, Currency currency, double amount, Currency target, double value) makes a new trade and adds it to the list of trades, this can both be a buy trade or a sell trade.

The public void run() runs every two seconds the functions updateTrader() and doTrades().

## Trade

The Trade class has seven global variables. They are all public. The variables are a:

- Public boolean buy
- Public Trader trade
- Public CurrencyTuple currencytuple
- Public double amount
- Public double targetvalue
- Public double amounttarget
- Public int id

The constructor of this class needs a boolean buy to indicate whether it is a buy or sell trade, it is true if it is a buy trade and false if it is a sell trade. The constructor also needs a Trader trader, the trader which conducts the current trade. It also needs a Currency start, it is the currency which you are going to trade. A double amount is also needed, the amount of currency the user is going to trade. It also needs a Currency target, this is the currency the user wants to get after this trade. The 'target' and 'start' are used to create a new CurrencyTuple. The constructor also needs a double targetvalue, this is the value which the user wants to conduct the trade, if this is equal to zero, then the trade conducts at the next update. The last double the constructor needs is an int id, this is the trade id and is unique for every trade.

The public void updateValues() updates the values of the two currencies in the current trade.

The public int getId() returns an int id.

The private double currencyAmount() returns the current value of a trade the user owns. It gets the price of currencytuple and multiplies it with the double amount.

The public boolean doTrade() conducts a trade if all circumstances allow it. The function returns a true if the trade has been conducted successfully and false if the trade has not been conducted successfully.

The public String toString() returns the current trade in string format.

## Wallet

The Wallet class implements Serializable and has three global variables:

- Private Map<String, Double> wallet
- Private double balance
- Private Client client

The constructor needs a wallet, which is a map that contains the holdings of the wallet. A balance, which is the balance of the wallet and the constructor needs a client, which is the Client of the wallet.

With the setter public void setBalance(double balance) the balance from the wallet is set.

The getter public double getHolding(String currency) returns the amount of currency that the wallet contains.

The getter public double getBalance() returns the balance from the wallet.

The setter public void setHolding(String currency, double amount) adds a currency to the wallet and sets the amount of the currency at the value of the double amount, using the line: wallet.put(currency, amount);.

With the function `addHolding(String currency, double amount)` a currency is added to the wallet with amount of the value from the double amount. The line `setHolding(currency, getHolding(currency) + amount);` makes that happen.

The getter `public Double getValue(String currencyName, String valueAs)` returns the value of a currency expressed as another currency.

With the function `public void printWallet(Textview t, String compareTo)` the wallet is printed to a textview. The parameter `compareTo` is the currency that the wallet holdings should be compared to.

### CurrencyTuple

The `CurrencyTuple` class has three global variables:

- Private `Currency owned`
- Private `Currency notOwned`
- Private `double price`

The constructor needs a `Currency owned`, where `owned` is a currency that the user owns. It also needs a `Currency notOwned`, where `notOwned` a currency is that the user own. At last, the constructor the constructor sets the double price to 0.00.

The getter `public Currency getNowOwned()` returns the currencies that the user does not own.

The getter `public Currency getOwned()` returns the currencies that the user owns.

The getter `public double getPrice()` returns the price from the current currency that the user is working with.

The setter `public double setPrice(double price)` sets the price of the double price.

### Design justification

Our design is a good design because the components cohere well. For the API class we had to choose between two different communication protocols that are provided by the exchange. We had to choose between a REST(ful) API or a websocket connection. The big difference between these two connection types, applied to our use case, is that with the REST API the client sends a GET request and gets a response. Using a websocket connection the client subscribes on a channel of data. Thereafter the client gets multiple responses from this channel, in this case a change of price. Because with our application the client chooses when to refresh it was an easy choice. So therefore the REST API is the best fit for our application. Of course every now and then a refresh has to be done, but still the REST API is the better fit. This is because an automatic update only needs to happen every 30 seconds to be operating well, but a websocket connection passes several hundred price changes per second.

## Project management

### Division of tasks

- Front-end: Reinier Sanders (chief)
- Back-end: Laurens Kubat (chief), Wout van den Berg, Ferran van der Have
- Project report: Wouter van Battum (chief), Ferran van der Have

### Progression

#### Week 1:

In the first week we just planned the project and everyone suggested some idea's. Also the meeting with the student assistant was in the first week.

#### Week 2:

in week two we started on the report and made appointments for meetings about the stages of the application.

#### Week 3:

In week three we started implementing the code and interface.

#### Week 4:

In week four we working on implementing the code and the interface, and during the meeting discuss the interface and make sure the same methods and attributes are used.

#### Week 5:

This week we were finishing up the code.

#### Week 6:

Handed in the code and finished the report.

### Organization

In the second week Willem Coster has dropped out from this Bachelor program. Therefore, he did not longer collaborate with us on this project and we worked without him. Willem was assigned to work on the project report and the front-end with Reinier.

In the project meetings we all collaborated well and listened to everyone's opinion. If any uncertainties occurred, for example if Map needed to be used or a List, no one hesitated to ask. Also if someone struggled with a part someone who knew how to solve it or something would help always helped. The members that worked on components that cohered, communicated well to attune the classes and components to each other.

During the meetings we did not actually said at that time this has to be ready, it was more like who is going to do what and when that was decided everyone just started working on his part and occasionally someone in the WhatsApp group asked how everyone was doing.

## Evaluation

### Laurens Kubat

In the end, I am decently satisfied with the end product. As to the updater and API fetcher, which I was mostly responsible for, there certainly some changes I would make, if I were to do a second version. For example, I figured that it would be easiest if I make a 'parser' for the responses of the exchanges myself, instead of using some open source library. Although my parses works just fine, it would be difficult to implement more features to the trader and the client with the use of my parser. This is mostly because I discard a lot of values in comparison to an open source parses, which doesn't discard all this data.

Looking back at the development process for the backend (which I was the chief of), I think we divided the work at the right parts, so that our parts were about the same size and effort. In total, we had three meetings for all the backend engineers, outside of the other meetings, which I felt was enough. Since Willem would work on the front end, his leaving of the project didn't matter for out part.

Looking forward, I would make the most changes in our use of GitHub/version control. I got the feeling that most backend engineers didn't know how GitHub works and weren't willing to find out how it works. At which point it becomes very hard to do regular code reviews or make sure that communication between our parts matched. If we would use this correctly, then we might be able to limit the amount of backend meetings to two or at least three very short meetings. In a future project, I would make sure that there is some form of agreement on which tools to use. Such as GitHub, or meet in person to do code reviews, instead of leaving it open like now and doing both half.

### Reinier Sanders

My task throughout development was being chief of the front-end of our app. I much more affiliate myself with the design of software instead of the actual coding. Therefore, I almost immediately prompted that I wanted to do this part of the project. Too much rejoice of the rest of the group, since they were all more interested in the back-end development. Willem Coster agreed to help me with the design. Unfortunately, he quit the project after about a week, due to him not being convinced he was able to obtain his BSA. This left the design of the app entirely in my lap, which I was actually quite relieved of. I am quite fickle when it comes to design decisions, so I was pretty happy to do it all by myself.

I feel like our group worked decently. The work was pretty evenly divided between everyone. The back-end team met every week to evaluate their code and to get on the same page. I had finished the global design of the app quite quickly, so I had to wait for them to finish the back-end. Due to some bugs and errors, it took a while before that was done. This caused me to only have about two or three days in order to link the entire front-end to all the various functions. I hit a pretty big roadblock at that point as our app kept crashing on start-up every single time. I was stumped as during testing on the Android emulators, everything seemed to work fine. If I was given more time than only two or three days at that point, I'm sure it could have been fixed.

We presented our app to Ward in its most finished state. That was, the back- and front-end were still completely separated. Luckily we could show that the underlying functions all worked and did their part. We explained the problem we encountered and that we honestly had no idea how to fix it. When asked if we could complete the project when given a bit more time. I told Ward that probably a day or two or three more would have been enough. So that is what we got. I am extremely grateful for this, because I was feeling a bit ashamed almost, to hand in the project in the state it was in.

At time of handing in this report we have found the problem and fixed it. I am now in the process of linking all front-end Views and Widgets to the back-end and haven't encountered any more problems. I am very relieved that this very small bit of extra time allows us to finish it and deliver a whole, nicely packed, working product.

In general, this was quite a fun learning experience. Personally, I already had some experience working with Android Studio. This allowed me to do my part pretty quickly and help the others when they had questions or got stuck thanks to Android Studio's sometimes quirky behaviour. Most of the problems could be solved by just Googling the issue, but I was glad to help nonetheless, because it moved the project forwards.

If we had to do another project I'd say the group would benefit the most from organizational improvements, instead of technical ones. I'd say the biggest point of possible improvement would be communication. It was pretty minimal in my opinion. Also the preparation phase wasn't as smooth as it should be. Making sure everyone knew exactly how the tools we used (Android Studio and GitHub) worked could have been done a bit better. Although that isn't entirely to blame on the process itself. Some of us were simply more eager than others to actually learn and understand how they worked.

In the end it all worked out though and I am glad with the result.

#### Ferran van der Have

The cooperation went very well with this group. After the first meeting we had everyone told what they were good at and we could divide the work evenly amongst everyone. After a while we lost one group member, he was supposed to work on the front end of the app, so it didn't matter much for me, as I was doing the back end.

I am satisfied with the developing process, everyone did what he was supposed to do. I learned that even if you divide a lot of the work, it is still fairly difficult to put everything together.

I had never worked with GitHub before, and other members of our group wanted me to use it. However, no one really explained it very well to me how to use it with the programs we use. Eventually I had to find things out for myself. GitHub didn't work how I wanted it to work, so I just made the best of it. I think it all worked fine the way we worked. Meeting in person is better in my opinion, then reading everything on the internet.

Altogether I am satisfied with the group and how everyone worked.

### Wout van den Berg

At the beginning of the project we came up with the idea to make a cryptocurrency trader application. Although I don't understand that much of cryptocurrencies I liked this idea. It is not possible to say if I'm satisfied with the final product yet because we got a little extra time due some complications and thus the project is not finished yet. However, I believe that I will be satisfied with the final product of the application.

When working on the project my task was to develop the client and the wallet. The process started quite good, everybody could work on their own part which all went quite well and that promised a good final product. However, when merging all individual made parts of the project we had some complications (for which we received some extra time). The main reason for these problems at merging the end of the project was that we gave ourselves too little time to merge the project. Merging gave some errors that we didn't see beforehand.

In future work I would probably try to merge the separate parts of the project earlier so that some problems can be discovered sooner and thus fixed within the time span of the project. In a future project I would also make more use of GitHub, which I didn't use that much this time because at the beginning it was very new for me.

### Wouter van Battum

In the beginning we had two different ideas for the application but they were rejected, because they were too difficult to implement or too easy. Therefore, Laurens came up with the idea to create a cryptocurrency trading application and I liked that idea. I do not know cryptocurrencies very well, but I had always been interested in cryptocurrencies so I was excited to develop this application.

I think the corporation within our group went well. From the beginning it was clear who was going to do what. We made a clear division of tasks, and I think it was divided well. Everyone got the task they wanted and fitted the best with their expertise. The only thing is that after Willem Coster quitted, Reinier had to do the front end. Although the division of tasks stated Ferran and I had to write the report I did it alone. I have no problem with that because Ferran was really busy with the back end and I managed to write the report alone.

The communication between the group members could be better I think, because the meetings took longer than expected. I think that had to do with the fact that the most of us were not familiar with GitHub, including me. Therefore, maybe some miscommunication about the code occurred.

I am satisfied with our application, because it works well and lived up to my view of the application that we wanted to create, minus one or two features, but I take that for granted.