CMPE 451

Milestone-1 Report
Fall 2019

Sercan Ersoy

Alper Çakan

Barış Zöngür

Ali Özden

İsmet Dağlı

Volkan Yılmaz

Furkan Aydar

Efe Önal

Ceren Tahtasız

Table of contents

1. Executive Summary
2. List and Status of Deliverables
3. Evaluation of Deliverables and Impacts on Plan
4. Summary of Coding Work Done
5. Project Requirements
    - 1. Functional Requirements
        - 1.1. User Requirements
        - 1.2. System Requirements
    - 2. Non-Functional Requirements
6. API Documentation
    - 6.1. users
        - 6.1.1. get users
        - 6.1.2. create user
        - 6.1.3. delete user
        - 6.1.4. get user
        - 6.1.5. update user
    - 6.2. auth_tokens
        - 6.2.3. login
    - 6.3. followings
        - 6.3.1. create following
        - 6.3.2. get followings
        - 6.3.3. delete following
    - 6.4. followers
        - 6.4.1. get followers
7. Project Plan
8. User Scenarios Used in Presentation
    - 8.1. Frontend Scenario
    - 8.2. Mobile Scenario
9. Git Workflow
    - 9.1. Overall Structure
    - 9.2. Details on Branches
10. Evaluation of the Tools and Managing the Project

1. Executive Summary

This is the first milestone report that describes the work, the deliverables and the status of work accomplished by group 8 so far. We are working on a project called “Traders’ Platform” as a social platform to inspect, share, and perform actions for trading purposes.

We planned that we design sign in and sign up functions with show profile options in this milestone because initialization of backend and connection with frontend/mobile take some time. It is important to connect first and test it for future work. However, we decided to add some minor features too after the connection is established. For mobile platform we added edit user profile and for frontend platform we added follow a user.

We describe what we have done detailed in this report. We write overviewed project requirements. Backend team explain API documentation. Frontend and mobile teams explain scenarios that is created for presentation.

2. List and Status of Deliverables

Deliverable Name	Description	Status
Backend Project	The API for our frontend and the mobile client to use. Tech stack: Postgre, Django	In progress
Mobile project	Android client for Mercatus. Tech stack: Kotlin	In progress
Frontend Project	Browser client for Mercatus. Tech stack: React JS	In progress
API Documentation	The document describing the authentication and the usage of the endpoints	In progress
Deployment	Published version of the mentioned projects.	Done. Changes may be needed in the future.
Group Meetings	Meeting notes and agendas are in Wiki	In progress

Project Requirements	Description of what the final product should be able to serve	Done. No requirement will be removed, new requirements might be specified in the future.
Diagrams	Use case diagrams, class diagrams and sequence diagrams of the project.	Completed
Personas and User Stories	Various use cases with imaginary users of the app.	Completed
Project Plan		Completed
Issues	Issues can be viewed to realize how the three project was distributed into smaller problems and how the assignments were given.	In progress
Pull requests	Milestones of the commits.	In progress

3. Evaluation of Deliverables and Impacts on Plan

So far, we have completed all of what we had planned to complete before the first milestone. After several discussions with our customer Alper, we had planned to have the tech stacks initialized and the basic functionalities (such as user profiles, authentications and permissions, follow feature etc.) finished by the first milestone date. We have managed to follow this plan strictly and indeed have done all these. All of our implementations are working well and as expected, except for a few minor bugs.

We are planning to finish, again just like we have discussed and concluded together with our customer, almost all of the features before the second milestone. Since we have succeeded in keeping up with our first milestone plan; our plan for the second milestone has not been delayed or adversely affected in anyway. So, we believe we will be able to keep up with the plan again.

4. Summary of Coding Work Done

Member	Work Done
Sercan Ersoy	1. Added endpoint delete user endpoint to the backend. 2. Added endpoints follow/unfollow user, get followers/followings to the backend. 3. There was situations for example a user with token tries to modify some other user's profile using PUT method. Corrected these situations with checking if the user with token is the target user. 4. When password is not sent to the PUT request, password was changing unintentionally. Corrected this situation. 5. When create only fields (like first_name and last_name) were not given to the PUT request, there was an error. Corrected this situation. 6. Removed trailing slashes from url endpoints to stick with the REST conventions. 7. Updated .gitignore file when needed, fixed a typo in the setup_db_creds.py. 8. For testing, we were using SQLite. Changed this back to the Postgres.
Alper Çakan	**Backend:** 1. Initialized the Python Django backend app. 2. Implemented a script that sets up the credentials for connecting the backend app to the database 3. Implemented the extra Django settings (switching to postgres database, iso date format; CORS settings; email based auth) 4. Wrote up a technical readme about the backend app that has instructions about setting up the app, the database, implementing a new endpoint, implementing a new model, a new serializer 5. Implemented the data initializer ("fixture") that sets up the authentication groups in the database 6. Implemented the model "User" 7. Implemented the "details" serializer for the model "User", which also includes encrypted password storage in DB and defaulting the auth group to "basic user" when unspecified 8. Implemented the "simple" user serializer (which I designed to eliminate the circular serialization problem with the user follow relationships) 9. Implemented a new base serializer class that allows you to specify "create_only_fields" which are only written to DB while being created and never updated again (example usage: name surname) 10. Implemented User listing, create, update, read (view profile) 11. Implemented the authentication mechanism in the backend (auth tokens database, Django auth settings, fields etc.) 12. Implemented the login (i.e., auth token create) endpoint 13. Implemented the permissions mechanism and a couple of frequently

	needed permissions classes (is post request or is the user authenticated, is the user trader, is the user admin,. …) **Frontend:** 14. Implemented all-users listing functionality 15. Implemented followers and followed lists feature for the user profile 16. Fixed the “http unauthorized” bug of the user profile update feature 17. Implemented follow user feature 18. Implemented unfollow user feature
Barış Zöngür	1. Implemented sign-up and log-in pages stated based primary functionalities using hooks and state registers. 2. Implemented various state based render functions 3. Implemented different types of rendering of navbar for different types of user types. 4. Implemented sign-up pages connections with back-ends endpoints. 5. Correlated work with other frontend members to help with various implementations. 6. Main component design(app.jsx) and props passing to different components. 7. General organization of the code and developing visuals for pages.
Ali Özden	1. Made research about tools. 2. Initialized frontend application with reactJS. 3. Implemented Sign Up and Login components 4. Implemented api connections for Login and Profile pages. 5. Customized Profile Area for better visuals. 6. Implemented Followers component as modal. 7. Implemented Update Credentials section. 8. Attended frontend meeting which we decided our coding standards. 9. Attended 3 online workshops with frontend team for hot-fix’s. 10. Deployed frontend application. Made research about deployment methods and decided to serve application with nginx. 11. Attended meeting with backend team about api usage.
İsmet Dağlı	1. Revised the requirement, mock-up, project-plan, design 2. Made research about testing methods 3. Designed the database model with the back-end team 4. Documentation on the design database model 5. Made a research about how to implement on sign-up and login with google auth in Django 6. Testing Django implementation with SQLite, since I face unsolvable problems with PostgreSQL. (Due to PostgreSQL connection problems, I have not been able to many contributions on implementation of the back-end.) 7. Due to inefficiency until milestone1 because of me, I moved the

	mobile team.
Volkan Yılmaz	1. I revise the requirement, mock-up, project-plan, design and request some changes. 2. I initialize mobile branch and create first project structure. 3. I implement sign in and sign up layouts. We determined 1 login activity for both users and 2 separate activities for trader and basic users since we request different credentials. 4. I initialize and develop API connection with backend and mobile. 5. I create models for register, login and user info with data. 6. I implement sign in functionality and test it whether it works. 7. I implement sign up functionality for both trader and basic user. Afterwards I test it whether it works. I added date picker to layout. 8. I help to design and create edit profile screen. I generate shared preferences to transfer a user token when a user logs in.
Furkan Aydar	1. Initialized the Spring Boot project. 2. Set up the MongoDb database 3. Connected database with the Spring Project. 4. Created the User model. 5. Created the User repository. 6. Created Security configuration for the app. (JWT Authentication) 7. Modified the security configuration to separate the basic user, trader, guest and the admin users. 8. Created register, login and get users endpoints. 9. Created the CORS configuration. 10. Wrote the documentation for the initial version of the backend API. 11. Migrated the database to the PostGreSQL. After that, team decided to change the backend framework and we did not use the commits I made. Yet they are still in our repository.
Efe Önal	1. I created the first navbar and differentiated it based on the user types, a different navbar was rendered for each user. I customized it with a dropdown menu and a search box, as well as our logo. 2. I created the profile page component which is the main structure of a profile page. 3. I created the profile area component which is rendered in Profile Page component. I customized it's design. 4. I added update feature via an update button in the profile area. 5. After UI design phase was complete, I made the successful login redirect the user into her profile page with her credentials gotten with a get method. 6. I made research on Google Login and Google Maps features. 7. We made three online meetings as the frontend team in order to be able to work in harmony.
Ceren Tahtasız	1. Updated requirements document and mock-up according to feedback from new members.

	2. Updated communication plan and ReadMe file. 3. Made research on tools for mobile, git workflows and testing types for whole team. 4. Planned the design of layouts and flow of activities for mobile. 5. Implemented register layout: this page gives information to users on trader and basic user and when you click one, you go to corresponding registration form. 6. Implemented profile page layout for users which shows profile image, birthday, user name and mail. There's a button to edit profile. 7. Implemented profile page connection with back-end endpoints to retrieve data to show on user's profile. (layout and backend connection) 8. Implemented edit profile page (layout and backend connections) 9. Tested sign-in, sign-up, edit profile functionalities using black box testing. 10. Created milestone presentation scenario and made the presentation.

5. Project Requirements

1. Functional Requirements

1.1. User Requirements

1.1.1. Guests

- 1.1.1.1. The system shall allow the guests to view the price of any trading equipment.
- 1.1.1.2. The system shall allow the guests to read user comments about trading equipment.
- 1.1.1.3. The system shall allow the guests to view articles and annotations in articles.

1.1.2. User Authentication and Account Management

- 1.1.2.1. Sign Up
- 1.1.2.1.1. The system shall allow the user accounts to be either "basic" type or "trader" type.
- 1.1.2.1.2. The system shall require the user to provide their name, surname, email address and location to be able to create a **basic** account.
- 1.1.2.1.3. The system shall require the user to provide their name, surname, email address, location, id number and IBAN to be able to create a **trader** account.
- 1.1.2.1.4. The system, when it requires the user to provide their location, shall require the user to provide it using Google Maps.

- 1.1.2.1.5. The system should allow the user to use their Google account to provide the necessary sign up information which are available in their Google account and require the rest of the necessary information to be input using text.
- 1.1.2.1.6. The system shall obtain the necessary sign up information using text inputs, unless otherwise specified.
- 1.1.2.1.7. The system shall allow users to change their account type when necessary information is provided. Trader user can downgrade to Basic user and Basic user can upgrade to Trader user.
- 1.1.2.2. Sign In
- 1.1.2.2.1. The system shall allow users to sign in only using their Google accounts or by providing the necessary sign in information, username and password.

1.1.3. Trading

- 1.1.3.1. The system shall, for the trading users, have a "My Investments" section which shall have the functionalities specified here (1.1.3).
- 1.1.3.2. The system shall allow the trading users to invest on any trading equipment.
- 1.1.3.3. The system shall allow the trading users to make a buy order for a specified rate.
- 1.1.3.4. The system shall allow the trading users to set stop/loss limits.

1.1.4. Tracking

- 1.1.4.1. The system shall provide each user with a "Profit/Loss" section that is private to the respective users, and this section shall have the functionalities specified here (1.1.4).
- 1.1.4.2. The system shall show both basic and trading users their profit/loss amount, in terms of the currency they choose, by the user manually entering their investments.
- 1.1.4.3. The system shall show trading users their profit/loss amount, in terms of the currency they choose, by the user's trading history on the system.
- 1.1.5. Comments
- 1.1.5.1. The system shall allow users to comment and annotate on articles.
- 1.1.5.2. The system shall allow users to comment on trading equipment.

1.2. System Requirements

1.2.1. Search

- 1.2.1.1. System shall support searching for users and trading equipments considering all the information available and filtering the search results.
- 1.2.1.2. System shall support semantic search.
- 1.2.1.3. System shall support location based search.

1.2.2. Recommendation

- System shall have a recommendation system that recommends articles or trading equipment to users based on their history (what they already follow).

1.2.3. Ranking

- 1.2.3.1. System shall allow traders to rank others idea.

1.2.4. Trading Equipment

- 1.2.4.1. System shall allow traders to trade indices, stocks, ETFs, commodities, currencies, funds, bonds, and cryptocurrencies.
- 1.2.4.2. Each trading equipment shall include many functionalities, including but not limited to: the previous close, percentage change with the previous close, amount change with the previous close, day's range, and moving averages.

1.2.5. Communication

- 1.2.5.1. System shall allow users to follow other users and trading equipment.
- 1.2.5.2. System shall allow users to share their ideas as articles.
- 1.2.5.3. System shall allow users to comment and rate ideas of other users.
- 1.2.5.4. System shall allow users to comment about trading equipment.

1.2.6. Profit

- 1.2.6.1. System shall have a Profit/Loss section for each user.
- 1.2.6.2. This section should be private for each user.
- 1.2.6.3. System shall allow basic users to see their profit/loss amount in terms of the currency they
choose by manually entering their investments.
- 1.2.6.4. System shall allow trading users to see their profit/loss amount in terms of the currency they
choose by both manually entering their investments and using the investments they made in the Traders Platform.

1.2.7. Portfolio

- 1.2.7.1. System shall ensure each user has at least one portfolio.
- 1.2.7.2. Portfolios shall be able to renamed, edited, shared and followed (if already shared).
- 1.2.7.3. System shall allow users to add any trading equipment to their portfolios.

1.2.8. Notifications

- 1.2.8.1. System shall allow users to set alerts for certain levels of trading equipment.

- 1.2.8.2. System shall notify users in accordance with their alerts.

1.2.9. Account

- 1.2.9.1. System shall be provided necessary information for signing up. (Basic users are expected to provide their name, surname, email address, and location.)
- 1.2.9.2. System shall have specific location of users on Google Maps.
- 1.2.9.3. System should use Google account to retrieve necessary information for signing up/in.
- 1.2.9.4. System shall offer basic functionality to users after signing in and validating their email address.

1.2.10. Event

- 1.2.10.1. System shall have events section for users which contains economic events.
- 1.2.10.2. Events shall have significance level and country base properties.
- 1.2.10.3. Events shall be able to searched, filtered and chased.

1.2.11. Deployment

- 1.2.11.1. System shall be able to handle native web and native mobile (Android) clients. (Hybrid applications are not allowed.)
- 1.2.11.2. System shall be deployable on a remote and manually configurable server. (Amazon EC2 or Digital Ocean are strongly recommended.)

1.2.12. Standards

- 1.2.12.1. System shall support W3C Web Annotation Data Model.
- 1.2.12.2. System shall follow W3C Web Annotation Protocol.
- 1.2.12.3. System shall follow the standards introduced by the World Wide Web Consortium (W3C).

1.2.13. Ethics and Legal

- 1.2.13.1. System shall handle personal information, contact information, copyrighted contents and license issues according to legal requirements.

2. Non-Functional Requirements

2.1. Accessibility

- 2.1.1. The system shall be accessible on Android and Web.
- 2.1.2. The system shall be compatible with at least Chrome 70, Firefox 64, IE 11 or higher version.
- 2.1.3. The system shall be compatible with Android 5.1 or higher version.

- 2.1.4. The system shall have responsive UI for different screen resolutions for both Android devices and web browsers.

2.2. Availability

- 2.2.1. The system shall have at least 97% uptime. (Except during maintenance or unexpected errors)
- 2.2.2. The system should be under maintenance every Tuesday between 2.00 am and 2.15 am regularly.
- 2.2.3. If there exists an ongoing maintenance, the system should inform users about the estimated termination of the construction at least one day before maintenance.

2.3. Reliability

- 2.3.1. The system should generate a proper log file which includes server reports, error logs and user activities when an error occurs.
- 2.3.2. The system should create backup file everyday.

2.4. Security

- 2.4.1. The system should remind users to change their password after using it for 1 year.
- 2.4.2. The system should require passwords to contain number of characters between 8 and 20; must contain at least one lowercase letter, one uppercase letter, one numeric digit, but cannot contain whitespace.
- 2.4.3. The system should not allow user to login for 10 minutes after 5 unsuccessful login attempts.
- 2.4.4. The system should session expiration if a user is inactive for 5 minutes.
- 2.4.5. The system should have human verification tool (reCAPTCHA) to prevent bots.

2.5. Privacy

- 2.5.1. The system shall secure the private data of users according to Information Privacy Law.

2.6. Database

- 2.6.1. The system shall encrypt personal data of users with RSA before storing them in the database.
- 2.6.2. The database architecture should have trigger functionality in the case of unexpected network issues in the middle of value exchanges between users.
- 2.6.3. The database architecture should handle 97% of the queries less than 0.1 second.

6. API Documentation

Our API documentation can also be viewed at
https://documenter.getpostman.com/view/8736871/SVzuZgBL?version=latest

6.1. users

6.1.1. get users

Description: Endpoint for retrieving all the users in the system.
Request Method: GET
Request URL: {baseUrl}/users
Headers: Authorization: Token {token}
Data: -
Successful Status Code: 200
Example Response:

```
[
    {
        "email": "ahmet.kazan@gmail.com",
        "lat": 0.0,
        "long": 0.0,
        "first_name": "Ahmet",
        "last_name": "Kazan",
        "date_of_birth": "1990-01-01",
        "profile_image": null,
        "pk": 13,
        "groups": [
            1
        ],
        "followers": [
            {
                "pk": 14,
                "first_name": "Taylan",
                "last_name": "Aksoy"
            }
        ],
        "followings": [
            {
                "pk": 15,
                "first_name": "Ali",
                "last_name": "Özden"
            }
        ]
    },
    {
        "email": "taylanaksoy@gmail.com",
        "lat": 0.0,
        "long": 0.0,
        "first_name": "Taylan",
        "last_name": "Aksoy",
        "date_of_birth": "1997-09-22",
        "profile_image": null,
        "pk": 14,
        "groups": [
            1
        ],
        "followers": [],
        "followings": [
            {
                "pk": 13,
                "first_name": "Ahmet",
                "last_name": "Kazan"
            }
        ]
    }
]
```

6.1.2. create user

Description: Endpoint for creating a user in the system.
Request Method: POST
Request URL: {baseUrl}/users
Headers: -
Data:

```
{
    "email": "taylanaksoy@gmail.com",
    "first_name": "Taylan",
    "last_name": "Aksoy",
    "date_of_birth": "1997-09-22",
    "groups": [1]
}
```

Successful Status Code: 201
Example Response:

```
{
   "email": "taylanaksoy@gmail.com",
   "lat": 0.0,
   "long": 0.0,
   "first_name": "Taylan",
   "last_name": "Aksoy",
   "date_of_birth": "1997-09-22",
   "profile_image": null,
   "pk": 24,
   "groups": [
       1
   ],
   "followers": [],
   "followings": []
}
```

6.1.3. delete user

Description: Endpoint for deleting a user in the system.
Request Method: DELETE
Request URL: {baseUrl}/users/{id}
Headers: Authorization: Token {token}
Data: -
Successful Status Code: 204
Example Response: -

6.1.4. get user

Description: Endpoint for retrieving a user in the system.
Request Method: GET
Request URL: {baseUrl}/users/{id}
Headers: Authorization: Token {token}
Data: -
Successful Status Code: 200
Example Response:

```
{
    "email": "taylanaksoy@gmail.com",
    "lat": 0.0,
    "long": 0.0,
    "first_name": "Taylan",
    "last_name": "Aksoy",
    "date_of_birth": "1997-09-22",
    "profile_image": null,
    "pk": 24,
    "groups": [
        3
    ],
    "followers": [],
    "followings": []
}
```

6.1.5. update user

Description: Endpoint for updating a user in the system. You can use this method partially e.g. it is possible to change only a specific number of fields.
Request Method: PUT
Request URL: {baseUrl}/users/{id}
Headers: Authorization: Token {token}
Data:

```
{
    "date_of_birth": "1997-09-22"
}
```

Successful Status Code: 200
Example Response:

```
{
    "email": "taylanaksoy@gmail.com",
    "lat": 0.0,
    "long": 0.0,
    "first_name": "Taylan",
    "last_name": "Aksoy",
    "date_of_birth": "1996-08-19",
    "profile_image": null,
    "pk": 14,
    "groups": [
        1
    ],
    "followers": [],
    "followings": [
        {
            "pk": 13,
            "first_name": "Ahmet",
            "last_name": "Kazan"
        }
    ]
}
```

6.2. auth_tokens

6.2.3. login

Description: Endpoint for logging in to the system. Gives an authentication token to be used later.
Request Method: POST
Request URL: {baseUrl}/auth_tokens
Headers: -
Data:

```
{
     "email": "taylanaksoy@gmail.com",
     "password": "1234"
}
```

Successful Status Code: 200
Example Response:

```
{
    "token": "81974cc2e6d2027f6f381aff24cf8e641280c3b8",
    "user_id": 24
}
```

6.3. followings

6.3.1. create following

Description: Endpoint for following another user in the system.
Request Method: POST
Request URL: {baseUrl}/users/{user_pk}/followings
Headers: Authorization: Token {token}
Data:

```
{
    "following_pk": 13
}
```

Successful Status Code: 200
Example Response:

```
{
    "email": "ahmet.kazan@gmail.com",
    "lat": 0.0,
    "long": 0.0,
    "first_name": "Ahmet",
    "last_name": "Kazan",
    "date_of_birth": "1996-08-19",
    "profile_image": null,
    "pk": 13,
    "groups": [
        1
    ],
    "followers": [
        {
            "pk": 24,
            "first_name": "Sercan",
            "last_name": "Ersoy"
        },
        {
            "pk": 14,
            "first_name": "Taylan",
            "last_name": "Aksoy"
        }
    ],
    "followings": [
        {
            "pk": 15,
            "first_name": "Ali",
            "last_name": "Özden"
        }
    ]
}
```

6.3.2. get followings

Description: Endpoint for retrieving followings of a user in the system.
Request Method: GET
Request URL: {baseUrl}/users/{user_pk}/followings
Headers: Authorization: Token {token}
Data: -
Successful Status Code: 200
Example Response:

```
[
    {
        "email": "ahmet.kazan@gmail.com",
        "lat": 0.0,
        "long": 0.0,
        "first_name": "Ahmet",
        "last_name": "Kazan",
        "date_of_birth": "1996-08-19",
        "profile_image": null,
        "pk": 13,
        "groups": [
            1
        ],
        "followers": [
            {
                "pk": 24,
                "first_name": "Sercan",
                "last_name": "Ersoy"
            },
            {
                "pk": 14,
                "first_name": "Taylan",
                "last_name": "Aksoy"
            }
        ],
        "followings": [
            {
                "pk": 15,
                "first_name": "Ali",
                "last_name": "Özden"
            }
        ]
    }
]
```

6.3.3. delete following

Description: Endpoint for unfollowing a user in the system.
Request Method: DELETE
Request URL: {baseUrl}/users/{user_pk}/followings/{following_pk}

Headers: Authorization: Token {token}
Data: -
Successful Status Code: 204
Example Response: -

6.4. followers

6.4.1. get followers

Description: Endpoint for retrieving the followers of a user in the system.
Request Method: GET
Request URL: {baseUrl}/users/{user_pk}/followers
Headers: Authorization: Token {token}
Data: -
Successful Status Code: 200
Example Response:

```
[
   {
       "email": "sercan@gmail.com",
       "lat": 0.0,
       "long": 0.0,
       "first_name": "Sercan",
       "last_name": "Ersoy",
       "date_of_birth": "1990-01-01",
       "profile_image": null,
       "pk": 24,
       "groups": [
           3
       ],
       "followers": [],
       "followings": [
           {
               "pk": 13,
               "first_name": "Ahmet",
               "last_name": "Kazan"
           }
       ]
   },
   {
       "email": "taylanaksoy@gmail.com",
       "lat": 0.0,
       "long": 0.0,
       "first_name": "Taylan",
       "last_name": "Aksoy",
       "date_of_birth": "1996-08-19",
       "profile_image": null,
       "pk": 14,
       "groups": [
           1
```

```
        ],
        "followers": [],
        "followings": [
            {
                "pk": 13,
                "first_name": "Ahmet",
                "last_name": "Kazan"
            }
        ]
    }
]
```

7. Project Plan

	30.9.2019-6.10.2019	7.10.2019-13.10.2019	14.10.2019-20.10.2019	20.10.2019-22.10.2019
Frontend				
1.General React Study	✓			
2.Initialize Frontend		✓		
3.Create Main Application Layout for Different User Types		✓		
4.Develop Login and SignUp Pages			✓	
5.Develop Profile Page			✓	
6.Establish communication with the API				✓
7.Develop Following Functionality				✓

Backend				
1.Initializing the app	✓			
2.User model and serializer		✓		
3.User CRUD endpoints		✓		
4.Authentication			✓	
5.Permissions classes			✓	
6.User follow-unfollow features				✓
Mobile				
1.General Kotlin and Android Study	✓			
2.Initialize Mobile	✓			
3.Develop Login and SignUp Layouts		✓		
4.Develop basic user and main activities		✓		
5.Establish communication with the API			✓	

6.Sign in and sign up functionality			✓	
7.Develop show profile page activity and functionality				✓
8.Develop edit profile page activity and functionality				✓

8. User Scenarios Used in Presentation

8.1. Frontend Scenario

Can Ozture is a rich man who lives in Istanbul. Lately he has been interested in trading. A friend of his, Ali, is an active user of Mercatus Traders Platform and he encourages Can to use the application, or at least, check the application out as a basic user.

Ali opens the web application and signs-up as a basic user. After that, he logs-in and gets directed to his profile page. He clicks the users button which renders all users of Mercatus. He finds Ali and follows him. Accidentally he follows another person but then he unfollows the user right back.

8.2. Mobile Scenario

Ali Doğan is a white collar man who is working at a corporate firm. He is 30 years old and he follows economy closely. He hears Mercatus app from a friend and he downloads it.

Ali clicks on sign up and sees 2 user types in the app. He chooses to be a trader because he is looking for an app to invest through. He fills all necessary info to sign up as a trader (including IBAN and TCKN). He logins using his credentials. He sees his profile then edits it: he changes his birthday, mail and password. Then he sees his updated profile page.

9. Git Workflow

9.1. Overall Structure

We used the following Git Workflow in our development process:

Branch Name	What For	Description	Example
master	Development	Used for development, accepts merges from feature or hotfix branches.	-
stable	Stable	Is stable, accepts merges from master or hotfix branches.	-
feature-{f|b|m}{no}	Feature	For new features, must be checked out from master.	feature-b121
hotfix-{f|b|m}{no}	Hotfix	For fixing release bugs, must be checked out from stable.	hotfix-m163

Note that we do not have a stable branch yet because our project is not stable enough right now. As soon as we decide that it is stable, we will create the stable branch from master.

9.2. Details on Branches

1. **master** branch is used for development. It accepts merges only from **feature** or **hotfix** branches (except real urgent situations).
2. **stable** branch is composed of commits that are merged from **master** and are tested/stable. It can also accept merges from **hotfix** branches to fix bugs.
3. **feature** branches stem from **master**. These branches correspond to a new feature issue (e.g. branch **feature-f123** is created for a frontend feature issue with id 123). These branches must be deleted after merging.
4. **hotfix** branches stem from **stable**. These branches correspond to a bug issue (e.g. branch **hotfix-m142** is created for a mobile bug issue with id 142). These branches must be deleted after merging.

10. Evaluation of the Tools and Managing the Project

For the backend application, we had initially decided on using Java Spring along with MongoDB as the database engine. However, later we noticed that the conceptual model structure and the needs of our application was better suited to using PostgreSQL as the database engine. After implementing some features and endpoints using this stack, we observed that Spring was introducing a significant amount of boilerplate code and was more complex, which made it unnecessarily slow-to-develop for a project of this scale. As a result, we migrated the application to Python Django. Now, we are quite satisfied with our choice of technologies and frameworks. Django, along with the REST extension and a couple of more extensions has made the development really easy and fast for us.

For mobile, we decide to use Kotlin and for IDE we use Android Studio. For milestone 1, we were a group of 2 people for mobile application since initialization of backend and work relatively a bit more than other platforms. Even though we don't have any experience with Kotlin, we deliver our product as planned.
Kotlin is rapidly gaining popularity among android developers. It is native mobile language and it has lots of advantages over Java. Java is 20 years old and since each new version should compatible with old versions, it prevents Java to progress more. Kotlin is however very young language and interoperable with Java. Same tasks can be done with less effort and with short code blocks. Kotlin compiler is more advanced and safer. Moreover, a lot of IDEs support Kotlin language now.
Android studio performs well in terms of testing, debugging and run the application. You can connect with github and you can achieve a lot of things from IDE.

For frontend, we used reactJS since it is easier to create a web application with HTML and JS functionalities together. reactJS's component functionality is helped us to manage our project and work cooperatively and also saved us from code reuse. After project initialization we set a meeting to decide our coding standards and make project management plan. Since we set 4 meetings, it was easy to manage project. While deciding which parts we were going to implement before milestone 1, we focused on functionality and skipped static components for being clear to our customer about application's current state. We had a little hard time sending requests to our api since none of frontend team members have experience on this concept. In our meeting about api with backend team we solved that issue. We used axios for sending requests to our api and react-bootstrap (which is a js library) for our styling.