CMPE451

GROUP 11

Milestone 1 Report

1- Executive Summary

As we have given our first Milestone presentation to the customer, we are going to deliver the details of our progress and current situation in the project with this Milestone 1 report.

1.1 Introduction

To start, our project is a social platform for people who consider themselves as traders. People can sign up and log in the platform to take advantage of more advanced features such as buying and selling trading equipment. However, users can use searching functionality without login and signup. Guest users also can view articles, events, trading equipment, and comments.

Fundamentally, there are two types of users: basic and trader. Basic users can create, comment and rank on the articles. In addition, they can create and share portfolios. Not only that, they can add trading equipment to the created portfolios. They can also chase economic events and filter them by country and significance levels. Our platform has more advanced features such as setting an alert, making a prediction about trading equipment and following trading equipment.

Trader users have all the functionality that basic users have and they have extra features such as trading indices, stocks, ETFs, commodities, currencies, funds, bonds, and cryptocurrencies. They have an extra ”My Investments” section that enables them to set buy and sell order. Another cool feature of the platform is people can follow other people. In a nutshell, our platform creates a connection between people who are interested in trading and trading world.

1.2 Project Status

We have created the basis of the project from the starting of our project for 2 months. Backend services are built, initial design of front-end and mobile services is implemented.

Based on our initial meetings to construct a general view on the progress of our project, a big portion of deliverables planned for Milestone 1 are delivered to the customer. **Sign-up and login** functionalities are added. Users can sign up and **verify** their **emails** in the registration process.

After a successful login, users can create **portfolios,** add **trading equipment** to their portfolio and **follow the progress of the equipment** via a handy front-end system. Same functionalities are added to the **Mobile** app.

Users can also check **Events** section to follow **impactful news and events.** Besides that, they can also **write, read and rate** articles both by other users and themselves.

1.3 Changes planned

We are going to implement the rest of our requirements and improve some of the features that we already implemented according to our requirements. We are going to add a preview of the trading equipment's current value in the portfolio. Also now added stocks can be added again, this will be changed.

We will add voting and comment section to articles. Also users will be able to make daily predictions about the trading equipments. People can also click on the author and see their profile. Following users and being private or public user functionalities will be added. Moreover, we are going to get all events regularly from a third party API. Our home page will consist of small parts from each section. We are going to add other trading equipments to portfolio. We are going to implement alert functionality for events.

2- List and Status of Deliverables

Task	Status
Review of project requirements	Done
Project plan update	Done
Register	Done
Login / Logout	Done
Email verification	Done
Google Maps API	Done

Article Page	In progress : Viewing articles is done
Profile Page	In progress : Profile card and portfolio features are done
Portfolio	In progress : Adding stocks is done
Trading Equipments	In progress : Stocks are added
Events	Done in mobile, in progress in frontend

3- Evaluation of the Status of Deliverables

Backend Structure

In backend, we want to have a flexible architecture enabling creating models for abstraction of data and dividing the code into modules via services. All the backend deliverables are successfully completed in this milestone. For the next milestones, the prepared structure will make easier to extend the code.

Frontend

In frontend part we implemented the Register page in a fully functional way. We also checked the validity of the credentials of the user in the frontend and allowed Turkish characters as the TA suggested.

Google Maps API is also used while registering process as requested. In registration page user can choose to be a basic user or not. After submitting the registration form successfully user is asked to visit his/her mail address and click the email verification link and then automatically directed to Login page.

After registering by providing the correct information one can login and he/she is directed automatically to the profile page. In profile user can see his/her profile card with name, location and email address. We will add user type, article number, the points that user obtained by making predictions. In profile page user will be able to see his/her own articles.

We added portfolio feature to the profile page. User can add portfolios to the profile page but for now it is not possible to delete portfolios. Also user can add stocks to the created portfolios as trading equipment.

The events segment of the navigation bar is not working but actually the events and the events page are implemented but it is not pulling data from the backend. We written this part with lorem ipsum as we did before with the articles but we did not have time to connect it with the backend side. For now it randomly assigns stars to randomly created events. The code can be seen in bounswe2019group11/frontend/papel-frontend/src/EconEvent path.

Android

In Android, we successfully completed all the deliverables. We think, in the future, we can focus more on the design of the screens and we should also create a consistent design with frontend. We created utilization functions to prevent code repetitions and we should maintain it.

4- Coding work done by each member

Alkım Ece Toprak	Implemented the backend part for the articles including routes and services. Added some functions' documentation to the Swagger json.
Alperen Değirmenci	Implemented the Login and Sign-up pages of the mobile app. Created the design and added functionalities.
Aysu Sayın	Implemented Events and article pages of the mobile app. On the article page, all articles are shown as a custom list I designed. There is a button to create article which directs user to "Add article" page. Also, I implemented the page where user can see the contents of the article. There is also another page for viewing events.
Burak Enes Çakıcı	Implemented Registration Form fields (including choosing location) checker. Added a pop-up in case of bad entries. Added location attributes to user type. Was part of the Front End team and contributed team mates in creating general design and login, signup and profile pages.
Cumhur Kılıç	Implemented event part of the backend. Added data model, route and service for

	event. Added event functions' documentation to the Swagger json.
Doğa Yüksel	Booted up the frontend project with React, implemented Sign-up form and modals for successful/failed sign-ups/logins, implemented Navbar navigation and Navbar changes according to login state. Implemented global logged in state and integrated changes according to state into other pages (can't access profile if not logged in etc.) Implemented Portfolio lists and Stock lists (view & add items). Implemented Articles page and individual article pages and previews. Helped with overall design choices.
Hasan Yaman	Implemented Portfolio, Portfolio Detail and Trading Equipment Detail pages of the mobile app.I also add creating portfolio functionality and adding trading equipments to portfolio functionality. I created Profile page.I created the design for the pages.
İbrahim Kamacı	Implemented article data model and wrote database initializer for the push mock articles to the database. Implemented 'Stock' data model as a trading equipment. Used third party API to fetch stock data for real time experience and added some stocks to database. Stock service and proper endpoints were implemented. Implemented 'Portfolio' data model. Corresponding services and endpoints were implemented. All corresponding endpoint documentations were constructed by using Swagger API Documentation. Deployed frontend application on AWS.
Metin Dumandağ	Constructed Swagger documentation mechanism for the backend. Prepared the database on Mongo Atlas. Implemented and added Swagger documentation for the login, signup, email verification, user and lost password endpoints. Added related models and services for these endpoints. Added authentication check to the article endpoint. Deployed backend service on AWS.
Semih Akgül	Implemented profile card and profile page in frontend. Implemented events and event/id

	pages. Designed the logo of the app and general view of the website. Added pin feature to the map. Added ionicon open source icon set to the project.

5- Requirements

(Until the end of Requirements section, numbering system discards no. of this section '5')

1. Functional Requirements

● 1.1 User Requirements

o 1.1.1. Common User Requirements

- 1.1.1.1. Sign Up Requirements
  - **1.1.1.1.1.** Users shall be able to sign up to the platform providing necessary information, i.e. name, surname, e-mail address and location.
  - **1.1.1.1.2.** Users shall be able to provide their location information via Google Maps.
  - **1.1.1.1.3.** Users shall be able validate their e-mail addresses.
- 1.1.1.2. Sign In Requirements
  - **1.1.1.2.1** Users shall be able to sign in with their e-mail address and password.
  - **1.1.1.2.2** Users should be able to sign in with their Google account.
- 1.1.1.3. Profile Requirements
  - **1.1.1.3.1.** User profiles shall be either public or private to the other users.
  - **1.1.1.3.2.** User with the private profile shall be followed to see contents in a private user profile.
- 1.1.1.4. Portfolio Requirements
  - **1.1.1.4.1.** Users shall have at least one portfolio.

        - **1.1.1.4.2.** Users shall be able to create, rename and delete portfolios.
        - **1.1.1.4.3.** Users shall be able to share their portfolio in their profile page and follow other shared portfolios.
        - **1.1.1.4.4.** Users shall be able to add trading equipments to their portfolio.

- 1.1.1.5. User Specific Events Requirements

    - **1.1.1.5.1.** Users shall be able to have an **Events** section. In this section, users can chase economic events with different significance levels.
    - **1.1.1.5.2.** Users should be able to filter economic events by considering their significance level and country base.

- 1.1.1.6. Profit/Loss Requirements

    - **1.1.1.6.1.** Users shall have a **Profit/Loss** section which is private to the user.

- 1.1.1.7. Article Requirements

    - **1.1.1.7.1.** - Users shall be able to share their ideas as articles.
    - **1.1.1.7.2.** - Users shall be able to comment on the articles.
    - **1.1.1.7.3.** - Users shall be able to rate the articles.
    - **1.1.1.7.4.** - Articles shall be able to be seen by other users.
    - **1.1.1.7.5.** - Comments shall be able to be seen by other users.
    - **1.1.1.7.6.** - Article scorings shall be able to be seen by other users.

- 1.1.1.8. Alert Requirements

    - **1.1.1.8.1.** Users shall be able to set alerts for trading equipments.
    - **1.1.1.8.2.** Users shall be able to set alerts for events.

- 1.1.1.9. Prediction Requirements

    - **1.1.1.9.1.** Users shall be able to make predictions about the trading equipments.
    - **1.1.1.9.2.** User's prediction success rate shall be available on their profile.

        - 1.1.1.10. Searching Requirements
            - **1.1.1.10.1.** - Users shall be able to search for other users.
            - **1.1.1.10.2.** - Users shall be able to search for other users by location.
            - **1.1.1.10.3.** - Users shall be able to search for trading equipments.
            - **1.1.1.10.4.** - Users shall be able to search for events.
        - 1.1.1.11. Social Requirements
            - **1.1.1.11.1.** Users shall be able send follow requests to other users.
            - **1.1.1.11.2.** Users shall be able accept or decline follow requests.
            - **1.1.1.11.3.** Users shall be able follow trading equipments.
            - **1.1.1.11.4.** Users shall be able comment on trading equipments.
    - *1.1.2. Basic User Requirements*
        - **1.1.2.1.** Basic users shall be able to see their profit/loss amount in terms of the currency they choose by manually entering their investments.
    - *1.1.3. Trader User Requirements*
        - **1.1.3.1.** Traders are required to provide an ID number and an IBAN while signing up.
        - **1.1.3.2.** Traders shall be able 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.
        - **1.1.3.3.** Traders shall be able to trade indices, stocks, ETFs, commodities, currencies, funds, bonds, and cryptocurrencies.
        - **1.1.3.4.** Traders shall have a **My Investments** section.
    - *1.1.4. Guest Requirements*
        - **1.1.4.1.** Guests shall be able to view prices of trading equipments, user comments and articles and events.
        - **1.1.4.1.** Guests shall be able to use searching functionality.

- **1.2. System Requirements**
    - *1.2.1. Searching Requirements*

- **1.2.1.1.** System shall consider all the information available in user profiles, trading equipments and articles while performing the search.
- **1.2.1.1.** System shall allow semantic search. System shall also display the users, trading equipments and articles that have semantic relationship with the search tags in the search results.

o *1.2.2. Events Requirements*

- **1.2.2.1.** Events shall have different significance levels. (e.g., one star, two stars, three stars)
- **1.2.2.2.** Events shall be filtered by significance levels and country base.

o *1.2.3. Investments Requirements*

- **1.2.3.1.** Investments shall be made with any trading equipment.
- **1.2.3.2.** Investments section shall be private to user.
- **1.2.3.3.** Investments should contain some functionality such as buy/sell order for a specified rate, set a stop/loss limit.

o *1.2.4. Trading Equipment Requirements*

- 1.2.4.1 Trading Equipment Examples
  - **1.2.4.1.1.** Indicies
    - **1.2.4.1.1.1.** S&P 500
    - **1.2.4.1.1.2.** Dow 30
    - **1.2.4.1.1.3.** Nasdaq 100
  - **1.2.4.1.2.** Stocks
    - **1.2.4.1.2.1.** Acer Therapeutics Inc (ACER)
    - **1.2.4.1.2.2.** Shares TR/Emerging MKTS INFRA(EMIF)
    - **1.2.4.1.2.3.** Analog Devices (ADI)
  - **1.2.4.1.3.** ETFs
    - **1.2.4.1.3.1.** SPDR S&P 500 ETF (SPY)
    - **1.2.4.1.3.2.** Invesco QQQ Trust (QQQ)
    - **1.2.4.1.3.3.** iShares Gold Trust (IAU)
  - **1.2.4.1.4.** Commodities
    - **1.2.4.1.4.1.** Gold
    - **1.2.4.1.4.2.** Silver
    - **1.2.4.1.4.3.** Crude Oil
  - **1.2.4.1.5.** Currencies
    - **1.2.4.1.5.1.** USD

            - **1.2.4.1.5.2.** EUR
            - **1.2.4.1.5.3.** TRY
        - **1.2.4.1.6.** Funds
            - **1.2.4.1.6.1.** Pimco Total Return (PTTAX)
            - **1.2.4.1.6.2.** Vanguard Total Stock Market Index Fund (VTSMX)
            - **1.2.4.1.6.3.** American Funds Growth Fund of America (AGTHX)
        - **1.2.4.1.7.** Bonds
            - **1.2.4.1.7.1.** Treasury Bonds
            - **1.2.4.1.7.2.** Municipal Bonds
            - **1.2.4.1.7.3.** Corporate Bonds
        - **1.2.4.1.8.** Cryptocurrencies
            - **1.2.4.1.8.1.** Bitcoin
            - **1.2.4.1.8.2.** Ethereum
            - **1.2.4.1.8.3.** Monero

    - 1.2.4.2. Trading Equipment Functionality

        - **1.2.4.2.1.** Trade equipments should have some functionality such as the previous close, the percentage change with the previous close, amount change with the previous close, day's range, and moving averages.

- *1.2.5. Alert Requirements*

    - **1.2.5.1.** System shall be able to notify the users in accordance to their alerts.

- *1.2.6. Recommendation Requirements*

    - **1.2.6.1.** System should recommend articles or trading equipments to the users based on their histories.

2. Non-functional Requirements

2.1. Accessibility and Availability

- **2.1.1.** The system should work on web browsers that supports Javascript ES6.
    - **2.1.1.1.** Chrome 58 and above
    - **2.1.1.2.** Firefox 54 and above

        - **2.1.1.3.** Edge 14 and above
        - **2.1.1.4.** Safari 10 and above
    - **2.1.2.** The system should be available %99 of the time.
    - **2.1.3.** The application shall be provided for both web browsers and Android mobile devices.

- **2.2. Performance and Response Time**

    - **2.2.1.** The system shall work with up to a thousand HTTPS requests without crashing.
    - **2.2.2.** The system shall respond to all search requests under 5 seconds.

- **2.3. Security**

    - **2.3.1.** The system should use https, so that the traffic between browser (or app) and the web server is encrypted.
    - **2.3.2.** System should deny requests if excessive amount of requests has been made.
        - **2.3.2.1.** Rate-Limit protocol will be provided by our hosting company.
        - **2.3.2.2.** System accepts HTTP requests up to 1000.
    - **2.3.3.** All passwords stored in the database must be hashed and encrypted by npm bcryptjs package.
    - **2.3.4.** The system shall check that an authorization bearer token is present and correctly formed

- **2.4. Annotations**

    - **2.4.1.** The platform shall support W3C Web Annotation Data Model
    - **2.4.2.** Web Annotation Protocol shall be followed to store annotations.

- **2.5. Portability**

    - **2.5.1.** The application shall be compatible with various Android devices and web browsers that supports Javascript ES6.

- **2.6. Capacity**

    - **2.6.1.** The system must meet the agreed capacity requirements.

6- API Documentation

URL for the API documentation is as following:

http://ec2-18-197-152-183.eu-central-1.compute.amazonaws.com:3000/api-docs/

Article

GET /article

Article get all endpoint. (**articleGet**)

Return type

array[Article]

Example data

Content-Type: application/json


```
[ {
  "comment" : [ "Nice!", "Thank you." ],
  "title" : "Low-Cost Investing Can’t Get Any Lower Than Free",
  "body" : "The central bank cut rates for the first time since the Great Recession in late July...",
  "authorId" : "5da07611e407903a5d01b265",
  "voterNumber" : 72.0
}, {
  "comment" : [ "Nice!", "Thank you." ],
  "title" : "Low-Cost Investing Can’t Get Any Lower Than Free",
  "body" : "The central bank cut rates for the first time since the Great Recession in late July...",
  "authorId" : "5da07611e407903a5d01b265",
  "voterNumber" : 72.0
} ]
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responss

200

OK. All articles returned succesfully.

GET /article/{id}

An endpoint that returns the article with the given id. (**articleIdGet**)

Path parameters

id (required)

Path Parameter — Id of the article.

Return type

Article

Example

data

Content-Type: application/json


```
{
  "comment" : [ "Nice!", "Thank you." ],
  "title" : "Low-Cost Investing Can’t Get Any Lower Than Free",
  "body" : "The central bank cut rates for the first time since the Great Recession in late July...",
  "authorId" : "5da07611e407903a5d01b265",
  "voterNumber" : 72.0
}
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type

response header.

- `application/json`

Responses

200

OK. The article with the given id is returned. Article

`POST /article`

Endpoint to add an article. (**articlePost**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

article article (optional)

Body Parameter — Article to be added.

Return type

Article

Example data

Content-Type: application/json


```
{
  "comment" : [ "Nice!", "Thank you." ],
  "title" : "Low-Cost Investing Can’t Get Any Lower Than Free",
  "body" : "The central bank cut rates for the first time since the Great Recession in late July...",
  "authorId" : "5da07611e407903a5d01b265",
  "voterNumber" : 72.0
}
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 400 401

OK. Article is successfully added. Article

Validation error. Either the title or the body of the article was missing. Check causes field for diagnostics.

Authorization token was missing, invalid or expired. **500**
Internal error occurred. Check the cause field for the diagnostics.

Auth

POST /auth/login

Login endpoint. (**authLoginPost**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

credentials **credentials** **(optional)**
Body Parameter — The user credentials to login.

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
OK. User successfully logged in. User object and JWT token is returned. **401**
Invalid credentials. Either the user with the given email does not exists or the password does not matches the password for the given email.
500
Internal error occurred. Check the cause field for the diagnostics.

POST /auth/sign-up

Sign up endpoint for the Papel. (**authSignUpPost**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

user **User** **(optional)**
Body Parameter — The user to sign up.

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200 400 500
OK. Successfully signed up. A verification email is sent.
Validation error occurred for one of the parameters. Check the name and message fields for diagnostics.
Internal error occurred. Check the cause field for the diagnostics.

POST /auth/sign-up/verification/resend

Email verification resend endpoint. (**authSignUpVerificationResendPost**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

email email **(optional)**
Body Parameter — Email of the user.

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
OK. Verification email successfully resent.
400 500
Either a user not found with the given email or the user already verified.
Internal error occurred. Check the cause field for the diagnostics.

GET /auth/sign-up/verification/{token}

Email verification endpoint. (**authSignUpVerificationTokenGet**)

Path parameters

token (required)
Path Parameter — Verification token

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/text`
  `application/json`

Responses
200
OK. User successfully
validated. **400**
Validation error occurred for one of the parameters. Check the name and message fields for diagnostics.
500 Internal error occurred. Check the cause field for the
diagnostics.

Event

GET /event

Event get all endpoint. (**eventGet**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[inline_response_200]

Example data
Content-Type: application/json

```
[ {
  "date" : "date",
  "country" : "Turkey",
  "rank" : 3.0,
  "comment" : [ "comment2", "comment2" ],
  "title" : "title",
  "body" : "272.7 example"
}, {
  "date" : "date",
  "country" : "Turkey",
  "rank" : 3.0,
  "comment" : [ "comment2", "comment2" ],
  "title" : "title",
  "body" : "272.7
example" } ]
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200

OK. All Events returned succesfully.

GET /event/{_id}

Returns requested event with historical data (**eventIdGet**)

Path parameters

_id (required)
Path Parameter — Unique event ID

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

inline_response_200

Example data

Content-Type: application/json

```
{
  "date" : "date",
  "country" : "Turkey",
  "rank" : 3.0,
  "comment" : [ "comment2", "comment2" ],
  "title" : "title",
  "body" : "272.7
example" }
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 OK. Event returned succesfully.
inline_response_200

Portfolio

GET /portfolio

Portfolio get all endpoint.(Actually its unnecessary but still it can be useful for development) (**portfolioGet**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Return type
array[Portfolio]

Example data
Content-Type: application/json


```
[ {
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}, {
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
} ]
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200 503
OK. All Stocks returned succesfully.
Internal Server Error

DELETE /portfolio/{_id}/stock

Delete the stock to addressed portfolio /portfolio/:id describes the portfolio and /stock describes that request body should be a stock (**portfolioIdStockDelete**)

Path parameters

_id (required)
Path Parameter — Portfolio ID

Consumes
This API call consumes the following media types via the Content-Type request header:

- application/json

Request body

stock Stock (optional)
Body Parameter — Delete the specific stock from specific portfolio Note: _id field should be provided it is required

Return type
Portfolio

Example data
Content-Type: application/json


```
{
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
OK. The portfolio deleted successfully, Actually it should be No Content but for updating the portfolio immediately the endpoint sends the updated portfolio Portfolio
503
Internal error occurred.

POST /portfolio/{_id}/stock

Add the stock to addressed portfolio /portfolio/:id describes the portfolio and /stock describes that request body should be a stock (**portfolioIdStockPost**)

Path parameters

_id (required)
Path Parameter — Portfolio ID

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

stock Stock (optional)
Body Parameter — Add a stock to specific portfolio

Return type
Portfolio

Example data
Content-Type: application/json


```
{
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
```

```
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 503

OK. The portfolio created successfully Portfolio
Internal error occurred.

`GET /portfolio/{portfolioId}`

Portfolio get all endpoint.(Actually its unnecessary but still it can be useful for development) (**portfolioPortfolioIdGet**)

Path parameters

portfolioId (required)
Path Parameter —
Portfolio ID

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

Portfolio

Example data

Content-Type: application/json


```
{
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 503
OK. All Stocks returned successfully. Portfolio
Internal Server Error

POST /portfolio

Create portfolio endpoint (**portfolioPost**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Request body

portfolio portfolio (optional)
Body Parameter — The portfolio to create

Return type

Portfolio

Example data

Content-Type: application/json


```
{
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200 503
OK. The portfolio created successfully Portfolio
Internal error occurred.

GET /portfolio/user/{userId}

Returns all portfolios of the specific user. (**portfolioUserUserIdGet**)

Path parameters

userId (required)
Path Parameter —
User ID

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Return type

array[Portfolio]

Example data
Content-Type: application/json


```
[ {
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
}, {
  "name" : "My Portfolio",
  "_id" : "_id",
  "userId" : "5da076d955d9b93a7308f3d7",
  "stocks" : [ {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  }, {
    "stockSymbol" : "ADBE",
    "stockName" : "Adobe Inc. - Common Stock",
    "price" : 272.7,
    "name" : "ADBE - Adobe Inc. - Common Stock",
    "_id" : "_id"
  } ]
} ]
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
OK. The portfolios sent successfully
503 Internal
error occurred.

Stock

GET /stock

Stock(Trading Equipment) get all endpoint. (**stockGet**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[Stock]
Example data
Content-Type: application/json


```
[ {
  "stockSymbol" : "ADBE",
  "stockName" : "Adobe Inc. - Common Stock",
  "price" : 272.7,
  "name" : "ADBE - Adobe Inc. - Common Stock",
```

```
  "_id" : "_id"
}, {
  "stockSymbol" : "ADBE",
  "stockName" : "Adobe Inc. - Common Stock",
  "price" : 272.7,
  "name" : "ADBE - Adobe Inc. - Common Stock",
  "_id" : "_id"
} ]
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 503

OK. All Stocks returned succesfully.

Internal Server Error

GET /stock/{_id}

Returns requested stock with historical data (**stockIdGet**)

Path parameters

_id (required)

Path Parameter — Unique Stock ID

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

StockDetail

Example data


```
{
  "dailyPrice" : "\"2019-10-16 16:00:00\": {\n            \"1. open\": \"272.7000\",\n            \"2. high\":
\"272.8300\",\n
  "stockSymbol" : "ADBE",
  "stockName" : "Adobe Inc. - Common Stock",
  "price" : 272.7,
  "name" : "ADBE - Adobe Inc. - Common Stock",
  "_id" : "_id",   "monthlyPrice" : "\"2019-10-16\": {\n            \"1. open\": \"270.2300\",\n
\"2. high\": \"272.8100\",\n
}
```


Content-Type: application/json

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200

OK. All Stocks returned succesfully. StockDetail

503 Internal Server Error

User

GET /user

An endpoint that returns all the users. (**userGet**)

Return type

array[User]

Example data

Content-Type: application/json

```
[ {
  "password" : "strongPa$.$word",
  "surname" : "Doe",
  "iban" : "TR33 0006 1005 1978 6457 8413 26",
  "name" : "John",
  "location" : {
    "latitude" : 41.085987,
"longitude" : 29.044008
  },
  "_id" : "_id",
  "idNumber" : "12345678910",
  "email" : "johndoe@gmail.com"
}, {
  "password" : "strongPa$.$word",
  "surname" : "Doe",
  "iban" : "TR33 0006 1005 1978 6457 8413 26",
  "name" : "John",
  "location" : {
    "latitude" : 41.085987,
    "longitude" : 29.044008
  },
  "_id" : "_id",
  "idNumber" : "12345678910",
  "email" :
"johndoe@gmail.com" } ]
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200 500
OK. A list of users are returned.
Internal error occurred. Check the cause field for the diagnostics.

`GET /user/{id}`

An endpoint that returns the user with the given id. (**userIdGet**)

Path parameters

id (required)
Path Parameter — Id of the requested user.

Return type User
Example data


```
{
  "password" : "strongPa$.$word",
  "surname" : "Doe",
  "iban" : "TR33 0006 1005 1978 6457 8413 26",
  "name" : "John",
  "location" : {
    "latitude" : 41.085987,
"longitude" : 29.044008
  },
  "_id" : "_id",
  "idNumber" : "12345678910",
  "email" :
"johndoe@gmail.com" }
```


Content-Type: application/json

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200

OK. User with the given id is returned.

User **400**

There is no such user with the given id.

500

Internal error occurred. Check the cause field for the diagnostics.

POST /user/lost-password

Lost password endpoint. (**userLostPasswordPost**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

email email_1 (optional)

Body Parameter — Email of the password who lost his password.

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200 400

OK. An email that contains instruction about how the user can reset his password is sent.

User with the given email not found. **500**

Internal error occurred. Check the cause field for the diagnostics.

POST /user/lost-password/reset

Password reset endpoint. (**userLostPasswordResetPost**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

Token and password Token and password (optional)

Body Parameter — Lost password verification token and new password of the user.

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200

OK. Credentials are updated with the new password. **400**

Given lost password token is expired, invalid or the password is too short.

500

Internal error occurred. Check the cause field for the diagnostics

7- Project Plan

		Name	Duration	Start	Finish	Predecessors	Resource Names
4		Create Teams	1 day	9/24/19 8:00 AM	9/24/19 5:00 PM		Everyone
5		Revise Requirements	5 days	9/24/19 8:00 AM	9/30/19 5:00 PM		Everyone
6		**Start Project**	**14 days**	**10/1/19 8:00 AM**	**10/18/19 5:00 PM**		
7		Create the project for Android	4 days	10/1/19 8:00 AM	10/4/19 5:00 PM		Hasan Yaman;Aysu Sayn;Al...
8		Create test server	1 day	10/18/19 8:00 AM	10/18/19 5:00 PM		Alkm Ece Toprak;brahim Ka...
9		Design Backend Application Str...	5 days	10/1/19 8:00 AM	10/7/19 5:00 PM		Alkm Ece Toprak;Cumhur Kl...
10		Start React project for Frontend	1 day	10/3/19 8:00 AM	10/3/19 5:00 PM		Doa Yüksel;Burak Enes Çak...
11		**Login / Registeration**	**10 days**	**10/1/19 8:00 AM**	**10/14/19 5:00 PM**		
12		Create Login/Register page on...	3 days	10/3/19 8:00 AM	10/7/19 5:00 PM		Doa Yüksel;Burak Enes Çak...
13		Create Login/Register page on...	5 days	10/1/19 8:00 AM	10/7/19 5:00 PM		Alperen Deirmenci
14		Implement Login/Register Syst...	5 days	10/1/19 8:00 AM	10/7/19 5:00 PM		Metin Dumanda
15		Create validation page on Fron...	5 days	10/8/19 8:00 AM	10/14/19 5:00 PM		Doa Yüksel;Burak Enes Çak...
16		Create validation endpoint on ...	5 days	10/8/19 8:00 AM	10/14/19 5:00 PM		Metin Dumanda
17		**User Profiles**	**4 days**	**10/10/19 8:00 AM**	**10/15/19 5:00 PM**		
18		Create profile page on Frontend	4 days	10/10/19 8:00 AM	10/15/19 5:00 PM		Burak Enes Çakc;Semih Akgül
19		Create profile page on Android	4 days	10/10/19 8:00 AM	10/15/19 5:00 PM		Hasan Yaman
20		Create endpoints for profile page	4 days	10/10/19 8:00 AM	10/15/19 5:00 PM		Alkm Ece Toprak;Cumhur Kl...
21		**Events Page**	**5 days**	**10/15/19 8:00 AM**	**10/21/19 5:00 PM**		
22		Create events page on Frontend	3 days	10/15/19 8:00 AM	10/17/19 5:00 PM		Burak Enes Çakc;Semih Akgül
23		Create events page on Android	2 days	10/18/19 8:00 AM	10/21/19 5:00 PM		Alperen Deirmenci;Aysu Sa...
24		Create events page endpoints ...	2 days	10/18/19 8:00 AM	10/21/19 5:00 PM		Alkm Ece Toprak;Cumhur Kl...
25		**Articles Page**	**5 days**	**10/15/19 8:00 AM**	**10/21/19 5:00 PM**		
26		Create articles page on Frontend	2 days	10/18/19 8:00 AM	10/21/19 5:00 PM		Doa Yüksel;Burak Enes Çak...
27		Create articles page on Android	3 days	10/15/19 8:00 AM	10/17/19 5:00 PM		Aysu Sayn
28		Create articles page endpoint ...	3 days	10/15/19 8:00 AM	10/17/19 5:00 PM		Alkm Ece Toprak;Cumhur Kl...
29		**Trading Equipment Page**	**5 days**	**10/15/19 8:00 AM**	**10/21/19 5:00 PM**		
30		Create trading equipment pag...	3 days	10/15/19 8:00 AM	10/17/19 5:00 PM		Doa Yüksel
31		Create trading equipment API	2 days	10/18/19 8:00 AM	10/21/19 5:00 PM		Alkm Ece Toprak;Cumhur Kl...
32		Create trading equipment pag...	2 days	10/18/19 8:00 AM	10/21/19 5:00 PM		Alperen Deirmenci;Aysu Sa...
33		Milestone 1 Presentation	1 day	10/22/19 8:00 AM	10/22/19 5:00 PM		Everyone

8- User Scenarios

8.1. Mobile App User

8.1.1

Hasan is a student who wants to invest his money wisely. He heard the Papel mobile application from his best friend and installs the application. First, he tries to sign up as a trader user but he sees invalid IBAN error. He realizes his mistake and writes correct IBAN. Then he validates his user by clicking the link on the validation page. Then he tries to log in. But first, he writes the wrong password. Then he writes the correct password and logs in. After login, he creates a portfolio from the portfolio screen. Then he adds 4 stock to his newly created portfolio. Then he decides to remove 3 of them. He clicks the only remaining stock and looks at the daily and monthly graphs of the stock. And finally, he closes the app.

8.1.2

Jeff is a 55 years old businessman. After a meeting he logins to Papel mobile application. He navigates to the Articles screen and he looks at the article list. He clicks one of the articles and reads it. After reading the article, he decides to give 5 stars to the articles. He also wants to add an article. Therefore, he clicks the add article button but he gives up since writing an article is a serious job. Then, he clicks the events section and looks at the events. Finally, he closes the app.

8.2. Web User

Melih Yiğit is a 26 year old employee who has recently developed an interest in stock market and finances. After hearing about our new platform *Papel.xyz*, he opens our website in a new tab at work. He wants to add some trading equipments to check from time to time so he tries to sign up as basic user. At first, he doesn't want to provide his real information so he enters a fake name with numbers but the system warns him to enter a valid surname. When correcting his surname, he forgets to choose a location. The system, then, forces him to choose a location. He picks the correct location after second try and signs up successfully. In a pop up window, he reads he has to check his email to validate it. He goes to his inbox and clicks on the validation link. Then he is taken to login page. After logging in, he is directed to his profile page. He first wants to check out the articles page. In the opening window, he sees a lot of articles written by other users. He clicks on one of them that sounds interesting. After reading the article, he returns to his profile page. He wants to make an investment but he is not familiar with stocks and trading equipments that much. He first checks the trading equipments list and views status of a few of them. When he clicks on an item, he is directed to its page where he sees its current value and a figure plotting the past value. He ends up creating a portfolio called "My New Portfolio" with 2 trading equipments. At the end, he logs out of the system.

9- Code Structure, Branching

We used fork based branching model. So, each team member forked the main repository under the BounsWE user, created branches on their forks as they developed features and send pull requests. The pull requests are reviewed at least by one sub-team member(backend, frontend or mobile). If there are any changes requested, it is addressed and after the approval, the pull request is merged into the master branch.

10- Evaluation of Tools and Project Management

Last year we were a group but this year we are a team. Bekir is not among us but there is nobody new to the team. This year we are divided into smaller groups, backend, frontend and mobile. Alkım Ece Toprak, Cumhur Kılıç, İbrahim Kamacı and Metin Dumandağ are in backend team. Alperen Değirmenci, Aysu Sayın and Hasan Yaman are in android team.

Burak Enes Çakıcı, Doğa Yüksel and Semih Akgül are in the frontend team. We decided that in each group there would be someone who knew the work more or less. We can say that everybody could join the group that she/he wanted.

In the period up to the Milestone 1 subgroups arranged meetings according to the need of themselves. Meanwhile using the github more actively than we did in the previous year, we continued to use whatsapp for instant messages and sometimes for notifications about github issues. We also arranged meetings including all team members but small ones were a lot frequent because getting together in the class hour each week was meeting our needs and decreasing the workload to arrange a wide meeting. In case of need we continued after the class.

At the very beginning of the term we knew that none of us were very experienced and some were very new to their subject but group members were able to help each other and there was an atmosphere that everybody can ask questions and express thoughts.

We aimed to complete specific tasks each week up to the milestone week and achieved that. After the Milestone 1 we will again determine goals up to the next milestone considering the remaining time until the reveal of the final product.

10.1 Backend

- **Express:** Express is a fast, unopinionated, minimalist web framework for Nodejs. We used Express to power up our backend endpoints. It was straightforward to write endpoints and the ability to use middlewares that are published by the large Express community (such as bodyParser) made our job a lot easier.
- **Swagger:** Swagger facilitated us to write API documentation for our endpoints. We were able to specify the format of data required or returned by endpoints.
- **MongoDB & Mongoose**: MongoDB is a general purpose, document-based, distributed database for modern applications. We used MongoDB Atlas as our main database. To access the data on MongoDB, we used mongoose. With the help of Mongoose, we were able to create, read, update or delete the data on MongoDB using models instead of queries. Also, the well support for Nodejs promises by Mongoose made development easier.

10.2 Frontend

- **ReactJS :** We used ReactJS to build the web part of the project. React JS is an open source JavaScript library. It allows us to create components on a single page which can be updated without reloading the page. React enables us to control the flow of data between the components.

- **Bootstrap :** Bootstrap is an open source CSS framework and a design tool which includes a set of elements that can be useful to build a website such as forms, labels, buttons etc.
- **Ionicons :** Ionicon is an icon package designed for web, Android and iOS applications. It is an open source.
- **Google Maps API :** We used Google Maps API in our registration page to provide location information. Map allows user to select location multiple times.
- **Jquery :** JQuery is JavaScript library that enables writing complex codes in a shorter, faster and simpler manner.

10.3 Mobile

- **Android Studio:** We used Android Studio as an IDE for Android application development since we don't have any other choice. It has lots of built-in features that help development such as code completion, code formatting and creating constructors, getters, and setters automatically for newly created classes.
- **Postman:** We used Postman to test API endpoints, analyze responses from endpoints and find out different error messages.