Group 12

Milestone Report 2

Practice-app

13.06.2021

Refika Kalyoncu
Adalet Veyis Turgut
Ömer Yılmaz
İhsan Gökcül
Gökay Yıldız
Burakcan Kazım Yeşil
Batuhan Tongarlak
Kürşat Talha Berk Yamanoğlu

Table of Contents

Executive Summary

Project Description

Before starting the implementation of our app BeABee, we implemented a practice app which would help us experience the challenges ahead. In this practice app, we planned to implement an app from scratch which has a database, a backend server that is connected to the database, and a frontend UI which makes use of the backend server. All team members are required to make use of at least one external api and provide at least one functionality in our backend server. The server needed to be implemented with respect to RESTful API architecture. Besides the provided functionality, the members are also required to write tests for their code, this is to allow all members experience the challenges of a complete development process on a rather small project. After the implementation the app shall be deployed to a remote server using Docker so that the members can also experience a deployment process and gain insight.

Project Status

Before starting the development process, we had a meeting as a team to understand what needs to be done, and how we should do it most efficiently as a team. We decided that each member should create a new branch for their own work in GitHub where our project code is stored. Then we decided that for each task, one person should initiate the development process first, and the others can inspect the example code to implement their part. This allowed us to have a uniform codebase and accelerated our development process because most of the functionality that each member should implement was similar. We repeated this for each step of the process. The first step was to implement our api in Python using the Flask framework. We decided to use SQLite as the database management system. Every member was required to implement at least one endpoint and make use of at least one external api. And we were also required to write tests for our own code. After implementing our parts, each member created a pull request for their branch and other members reviewed their implementation. This helped us maintain a uniform coding style and architecture. After reviewing each other's implementations we decided separating database connection codes from server codes would be a better design so the members refactored their code accordingly. The second step was to create the frontend that utilizes our api. We implemented it in a similar manner. One of us initialized the frontend using Vue.js and others followed. After everyone's implementations were done and pull requests were approved, we merged all branches into master and started the deployment phase. We used Docker as the container of our application and deployed it to Amazon Web Services EC2. We created two Docker containers for our Frontend and Backend. Later using "docker-compose" we connected these two containers which allowed us to easily deploy our application to the remote server at once.

URI of the Deployed Application

- http://18.184.69.221:5000/api/docs

API URL

- **http://18.184.69.221:5000/books [get], [post]**
- **http://18.184.69.221:5000/download_issues [get]**
- **http://18.184.69.221:5000/issues [get], [post]**
- **http://18.184.69.221:5000/issues/<int:number> [get], [post]**
- **http://18.184.69.221:5000/anime/search [get]**
- **http://18.184.69.221:5000/anime/<int:id> [get], [post]**
- **http://18.184.69.221:5000/anime [post]**
- **http://18.184.69.221:5000/addQuotes [post]**
- **http://18.184.69.221:5000/quotes [get]**
- **http://18.184.69.221:5000/randomQuotes[get]**
- **http://18.184.69.221:5000/movies/<keyword:str> [get]**
- **http://18.184.69.221:5000/movies_addReview [post]**
- **http://18.184.69.221:5000/movies_home [get]**
- **http://18.184.69.221:5000/name_information [post], [get]**
- **http://18.184.69.221:5000/convert [post], [get]**
- **http://18.184.69.221:5000/cocktails/get_cocktails/**
- **18.184.69.221:5000/cocktails/create_cocktail/**

  Here is a documentation of our API
  https://github.com/bounswe/2021SpringGroup12/wiki/Practice-app-Documentation

Basic Functionality of Our Project

Every member of our team implemented a different functionality using various external APIs. Here are basic functionalities of our project that you can use.

- Users can get the information of the books of an author. Information includes review url, publication date, summary, isbn numbers, etc.
- Users can post the information of the books. Information includes title, author, review url, publication date, summary, isbn numbers, etc.

- Users can get the information of the daily currency exchange rate. Information includes date, currencies, exchange rate, calculated amount of given money.
- Users can post an exchange rate value for a specific date for two currencies.

- Users can get the average age for given name and country fields. The resulting information is the combination of data acquired from agify.io and application database.
- Users can save new name information to the database to be included in later queries.

- Users can get quotes of the famous people by random subject
- Users can get quotes of the famous people by subject
- Users can add quotes of anybody or theirselves to the database

- Home page for movies which includes 2 options. Add review or Search for keyword.
- Users can get the reviews of movies. Information includes title of the movie, mpaa rating, critics pick, name of the reviewer, title of the review, short summary of the review and the link of the review.
- Users can post movie reviews. Information includes title of the movie, mpaa rating, critics pick, name of the reviewer, title of the review, short summary of the review and the link of the review.

- Downloads and simplifies the last 100 issues in bounswe/2021SpringGroup12 repo to database.
- Users can also get all issues that reside in our database.

- Get detailed results about one anime. Before responding anime info is posted to the database as MalAnime.
- Post User defined animes to the database. Animes posted by users may lack some detail.

- Users can get the cocktails' ingredients, glass type and the instructions with the keyword
- Users can add cocktails they created.

Challenges We Met As A Group

- Writing code without conflicts is an art and we are just newbies trying to learn. Merging and handling conflicts were big challenges.
- Composing frontend, backend dockers and making them work as a whole process was troublesome.
- Import was also a big problem while we were unittesting and the code itself required different versions of import like” main.db.schemas” and “db.schemas”. Thanks to Suzan Üsküdarlı we handled this problem
- We changed the folder structure while implementing the project. Structure change caused some problems while merging. We should determine the folder structure beforehand when we start the project for Cmpe451.
- Learning Vue.js in a relatively short time was troublesome but worth it. We are proud of the view of our website.

List of Status of Deliverables

Name	Due	Delivery date	Prepared by
Researching requests, postman,flask,api etc.	**23.05.2021**	**23.05.2021**	**Everyone**
Searching for suitable apis	**23.05.2021**	**23.05.2021**	**Everyone**
Post and Get methods for api	**30.05.2021**	**1.06.2021**	**Everyone**
Frontend for the apis	**1.06.2021**	**4.06.2021**	**Everyone**
Unittests	**6.06.2021**	**6.06.2021**	**Everyone**
Dockerizing the application	**7.06.2021**	**7.06.2021**	**Everyone**
Deploying the application to a server using docker	**8.06.2021**	**8.06.2021**	**Everyone**
Milestone 2 report	**13.06.2021**	**13.06.2021**	**Everyone**

Evaluation of Deliverables

1.1. Researches for Request, Postman, Flask, Api, etc...

After the first meeting we decided that every member will research about the topics that will be relevant when implementing the practice API we will be using before the next meeting. This was not a deep down research to learn everything about the subjects but was necessary to get familiar with the concepts and tools before diving into the project.

1.2. Search for Suitable APIs

After the first meeting every member of the team had to search and decide on a third-party API that they will be using on their own endpoints. We had no limitation about the API that we chose, so everyone chose and learned about the details of an API that they liked before they started to work on the implementation.

1.3. POST and GET methods for API

After the second meeting, the team decided that Veyis will initialize the backend part of the project using the Flask framework for Python to ensure that every person will follow a common structure to ease up to the merging process.

After Veyis pushed the first draft of the backend, every member started to work on their POST and GET methods that will use the third-party API that will use. Every member pulled the first draft, created a new branch and after they finished their implementation created a new pull request to prevent confusion and errors.

The merging process was one of the most time consuming parts of the all project. Although every member tried to keep track of the master branch and work on it to prevent more conflicts, this process was not that perfect. Since every member had their own pace with the coding process, some pull requests and branches were out of date within a short amount of time, so we had to keep track of the branches and sometimes manually merged the other branches to the branch, so it was a lot of work for the team. We think this problem could have been majorly solved at the beginning by trying to make project structure more modular, but at this stage it was more work to refactor everything.

Although there was a deadline and such, the backend code for nearly everyone were altered throughout the process to support structure, make testing easier etc…

One of the major problems with these structure changes was the import structure of the Python. Because everyone was working in their own development environment and our lack of deep understanding of the Python import system, when merging the branches or changing the project structure we had very difficult times with running the project. Project failed to run most of the time because import statements of personal branches do not work on master branch or vice versa. However thanks to Suzan Hoca and group's efforts we have solved this issue and managed to run our code without any import errors. It is an important aspect to learn about more and think about when structuring the project.

With all these difficulties the backend code was up and running at the end.

1.4. Frontend for the Practice App

For the Frontend, again we decided one member of our group would initialize the code to maintain a coding structure and style. This time Burakcan was the person to initialize the frontend code. However, since the initial draft of the front-end code was delayed and a brand new framework was used that had been learned from scratch by some members, the delivery date was a bit later than expected.

On frontend, the code group used the Vue.js framework, because of it is relatively easy to use and to learn, and its modularity. With Vue, every member worked on their own separate folders and separate files so conflicts were rare and everyone coding at their own pace was more viable. After each member finished their frontend code, they pushed it to their branch to create a pull request.

When merging the different development branches we had less problems this time then the backend part, due to the more modular structure of the Vue.js. So the only major problem with frontend merging was that the general structure of the project has changed several times during the implementation process. So these changes required some work to even out the all branches' structure.

Another thing we had to deal with the frontend was connecting to the backend API. Since everyone was developing in their own environment, everyone used different connection urls for their API. So before getting done with the frontend we chose to keep the API url as an environment variable to keep the front end regular for everyone.

1.5. Unit Tests

A critical part of software development is unit tests. In this project, after every member finished their part of the backend, they designed, coded and executed their unit tests. Generally each member tested their helper methods which were modular parts of their API methods. Each part of the code was tested with both successful and failed tests.

Each part of the app having their own unit tests made the merging process smooth for this part as well. It again had some small difficulties like the frontend but nothing major was there. But same as the other backend parts we had some issues with the import statements at the unit tests too. Most of our members had difficulties even running their tests in their own development environment. But solving the issues with the backend with the help of Suzan Hoca helped a lot here.

1.6. Dockerizing Individual APIs

At the end of the development process, our project had to be deployed somewhere to be accessible by others. And Docker is a tool that helps greatly with the deployment process. So at the end of the API implementation, we wrote a Dockerfile for the project to let each member try to dockerize and run their container on their own PC. Letting everyone run their own container pushed us to learn more about Docker and containers.

After everyone using docker own their own setup we had a meeting to try to dockerize and run the master branch. After a little effort we managed to run the docker container of the backend part in Veyis' computer.

After that we wrote a Dockerfile for the frontend part too. And then a docker-compose file to run both of these containers together.

Here we ran into our first big problem. Although both containers were running without problems on their own, we could not manage to connect them both to each other. We started to do more research and seek a solution to our problem, and finally solved our problem. Next time having more knowledge about docker and networking will help us a lot when dealing with these issues.

1.7. Deploying the Application to AWS

Last step of this project was deploying our application to an AWS server. AWS was a subject none of us had great knowledge about, so the deployment process has come with a lot of difficulties.

At the first meeting we had to try to deploy, we noticed none of us had an AWS account, so most of this meeting was spent trying to register and get a server.

At the second and last meeting of the implementation part of the project we had an AWS account so we uploaded necessary files and codes to the server and ran our docker-compose script. However we had a problem there. Although our backend and frontend seemed to run without any problems, again the frontend could not connect to the backend. So we tinkered around the Dockerfiles and docker-compose string, scanned the internet to find a solution. Although this problem was very similar to the one we had in our local computers, the same solution did not apply here. After a long searching process we found a solution and deployed our code to the AWS server.

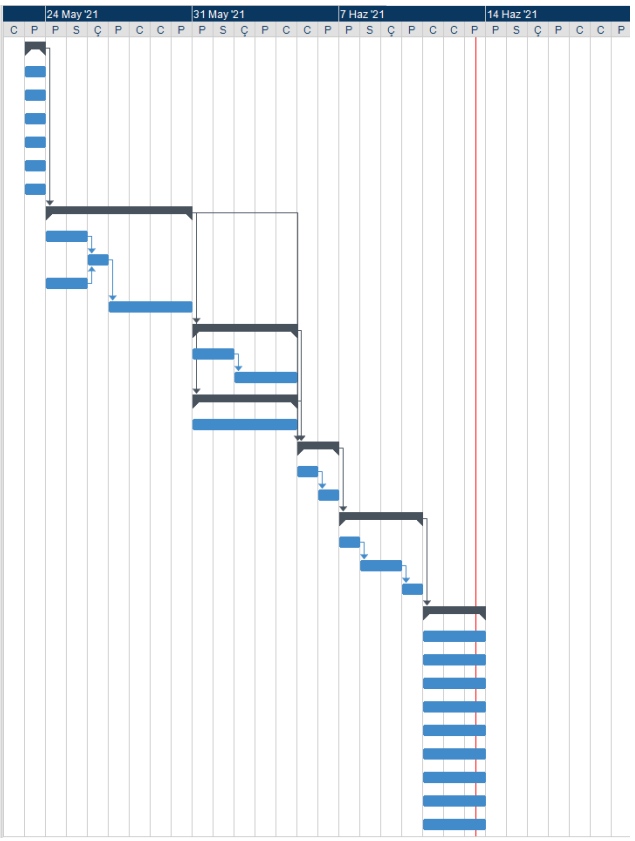
1.8. Milestone 2 Report

After we were done with our implementation process, we had a meeting to split the works about Milestone 2. After this meeting everyone started work on their parts. Since we are familiar with this kind of reporting from Milestone 1 and got some feedback then, we had a more smooth time doing this part compared to the implementation.

Although same as previously we had some problems with ProjectLibre, the program that we use to do planning. After working on it for a while Omer decided to use another application for this part.

Project plan

	Tas...	Task Name	Duration	Start	End	Predecessors	Resources
1		**Research**	**1 day?**	**23.05.2021**	**23.05.2021**		
2		Http Request	1 day?	23.05.2021	23.05.2021		Everyone
3		REST-API	1 day?	23.05.2021	23.05.2021		Everyone
4		Flask	1 day?	23.05.2021	23.05.2021		Everyone
5		Unit Test	1 day?	23.05.2021	23.05.2021		Everyone
6		Docker	1 day?	23.05.2021	23.05.2021		Everyone
7		AWS	1 day?	23.05.2021	23.05.2021		Everyone
8		**Backend**	**7 days?**	**24.05.2021**	**30.05.2021**	**1**	
9		Chose an API	2 days	24.05.2021	25.05.2021		Everyone
10		Initialize API	1 day?	26.05.2021	26.05.2021	9;11	Veyis
11		Chose a DB	2 days	24.05.2021	25.05.2021		Everyone
12		Create End Points	4 days	27.05.2021	30.05.2021	10	Everyone
13		**Frontend**	**5 days**	**31.05.2021**	**4.06.2021**	**8**	
14		Initialize Frontend	2 days	31.05.2021	1.06.2021		Burakcan
15		Create Frontend Pages	3 days	2.06.2021	4.06.2021	14	Everyone
16		**Test**	**5 days**	**31.05.2021**	**4.06.2021**	**8**	
17		Create Tests	5 days	31.05.2021	4.06.2021		Everyone
18		**Merging**	**2 days**	**5.06.2021**	**6.06.2021**	**8;13;16**	
19		Create Pull Requests	1 day	5.06.2021	5.06.2021		Everyone
20		Merge Requests	1 day	6.06.2021	6.06.2021	19	Everyone
21		**Deployment**	**4 days?**	**7.06.2021**	**10.06.2021**	**18**	
22		Dockerize Application	1 day	7.06.2021	7.06.2021		Veyis;Burakcan
23		Deploy to AWS	2 days	8.06.2021	9.06.2021	22	Everyone
24		Release Tag	1 day?	10.06.2021	10.06.2021	23	Veyis;Refika
25		**Report**	**3 days**	**11.06.2021**	**13.06.2021**	**21**	
26		Readme.md	3 days	11.06.2021	13.06.2021		Veyis
27		Document End Points	3 days	11.06.2021	13.06.2021		Everyone
28		Project Description	3 days	11.06.2021	13.06.2021		Berk
29		Project Plan	3 days	11.06.2021	13.06.2021		Ömer
30		Organizing And Evaluation	3 days	11.06.2021	13.06.2021		İhsan;Batuhan
31		Document The Work Done	3 days	11.06.2021	13.06.2021		Everyone
32		List Status Of Deliverables	3 days	11.06.2021	13.06.2021		Refika
33		Evaluation Tools and Pr...	3 days	11.06.2021	13.06.2021		Gökay
34		Document Frontend/Doc...	3 days	11.06.2021	13.06.2021		Burakcan



Evaluation of Tools and Processes

Mindview

Ömer used Mindview for the project plan. We have used ProjectLibre for the first Milestone but Ömer says that Mindview is more user friendly and easier than ProjectLibre. We can use it in the future.

Git

We have used git as a team, each member has successfully managed to use git as a version management system. It makes working on a remote repository on
several local machines are less troublesome. In the learning phase we helped each other as a team for compliance purposes on our master branch and repository.

Github

So far, we have used Github as our organization and to follow the work done by each member on different assignments. After milestone1 and practice-app
homework we have started using the essential functionality of Github, which is developing software collaboratively.
Github is really useful when it comes to reviewing other team members' developments and improvements on base code. Also, when it comes to merging the
separate branches the pull request system makes the merge safer and reliable.

Google Docs

We have used Google docs when there is a need for collaboratively working on a report. It makes it faster to complete the task when there is different
parts that are handled by different members.

VSCode

Some of us used VSCode to develop and inspect our practice-app code base. It is not a complete IDE for a specific language but an effective editor with lots
of extension options with a lot of language support.

Pycharm

Pycharm is an IDE designed for python development. Since we have used Flask which is a micro framework for python for our API development assignment, using

Pycharm as a development environment is useful. Also, we have to write our test cases in python, running the test cases in Pycharm is easier than using VSCode.
Also, the git integration makes it easier to walk around between branches and test other group friends' development when necessary.

Postman

Postman is useful when it comes to testing both the outsource API that we used to gather data and the API that we developed by using the gathered data.
It makes development faster since even though we did not have the frontend at the first place, we tested our backend and continue to do the frontend part without any doubt.

AWS

We have used AWS to deploy our project. At first, it is a bit hard to adjust since we only have a terminal as an interface. Gathering around files, solving bugs
on AWS is a little bit time consuming at the beginning. But in the end we have successfully managed to deploy our product.

Docker

We have successfully used Docker and dockerize our application before the deployment. Docker creates a virtual environment as needed for an application
to successfully run. We specified this requirements and created 2 different containers for backend and frontend since we have implemented frontend and
backend separately. To connect them we have also used docker decompose.

Beekeeper Studio

Veyis used Beekeeper Studio to see the contents of our database. This application is highly rated in Ubuntu and can connect to different database services like Postgres, Sqlite, MySQL.. It shows tables, triggers and contents. Users can also filter out and delete/add the data without writing queries. It has a built-in query executer too.

Grip

Veyis used this for writing documentations. It converts markdown files to HTML. It can be downloaded to Ubuntu with a single command: "**sudo apt**-get **install** -y **grip**". It supports hot reload. Veyis loves to work with markdown rather than Word and he tried pandoc and online converters. grip became the first choice. Linking some sections in the same file, adding external links and making tables were pretty easy.

Table of Work Done by Each Member

Adalet Veyis Turgut	- At first, I wrote a simple GET endpoint using Flask and wrote an informal wiki documentation page of this endpoint. issue #122, pull #123 - Written manual of how to set up the environment and running the application to inform other members. - Get endpoint (mentioned above) was very straight-forward and long. I initialized schemas.py and mapper.py files in order to use ORM. pull #128 - Wrote POST /books/ endpoint. Added database support: created database_initializer.py, create-tables.sql files. issue #129, pull #128 - Reviewed pull requests. - Attended all of the meetings, split tasks, assigned deadlines, commented on others' progress. - Uploaded some of the meeting notes to wiki page: meeting #12, meeting #13 and revised other notes. - Realized that my codes were simple. Considering Ömer's helper functions, I divided my code into helper functions. pull #138 - Wrote unittests. There are more than 25 unittests. 8 of them test endpoints, remainings test helper functions. At first couldn't run tests due to import problems. Tried to generate HTML reports too, but couldn't succeed again. Suzan Hoca helped us to solve these problems. issue #141, pull #142 - Implemented frontend of books page. issue #140, pull #142. Realized that I can improve this desing, made necessary changes pull #158. - Merged most of the pull requests to master - Dockerized backend. This was my first experience with Docker. I created a lot of images and containers. Creating, removing, running, configuring them repetitively made me learn. - Deployed final application as a whole on AWS. Connected frontend to backend, embedded secret keys and ip of the AWS to docker-compose file. issue #150 - Tested the application on my local - local docker - AWS, found bugs and reported them. issue #181 - Written manual to run the application using Docker. - Created URI documentation of our API. Written necessary codes. issue #195
İhsan Gökcül	-Researched about Flask, Restful API, Postman, http requests (meeting#11) -Implemented GET and POST functions for /convert/ endpoint of our API.(issue#127, pull#131) -Improved GET and POST functions using helper functions. Created helper functions in “currency_helper.py” file.(pull#131) -Contributed in “mapper.py” and “schemas.py” files for “/convert” endpoint

	database connection.(pull#131) . -Created both database tables in “create_tables.sql” file, and database connection functions in helper functions for related data that used for /”convert” endpoint.(pull#131) -Prepared a documentation for my endpoint “/convert” GET and POST functions.(https://github.com/bounswe/2021SpringGroup12/wiki/Practice-app-Documentation#2-convert) -Researched about unittest and testing a Restful API on youtube and the internet. -Implemented tests in “test_currency.py” file for testing api functionality for “/convert” endpoint.(issue#169, pull#173) -There were some problems with folder structure. Updated my code according to new folder structure.(issue #135) -Researched about vue.js framework.(issue#171) -Implemented frontend pages for “Currency Conversion” page which uses the "/convert" endpoint in “ConvertHome.vue”, “ConvertPost.vue”, “Index.js”, “App.vue” files.(issue#171, pull#173) -Attended meetings #11, #12, #13, #14, #15. -Reviewed pull requests of other team members. -Contributed to merge operations. -Opened pull requests #131 and #173. -Requested reviews for my pull requests #131 and #173 from my group members.(pull#131, pull#173) -Researched about docker and aws on internet. -Attended some of the meetings for docker and aws.(meeting#18) -Corrected some deleted and problematic parts that occured in merging.(pull#173)
Batuhan Tongarlak	- Research of API, requests and the tools - Cocktail Finder GET and POST endpoint using Flask - Wiki documentation for Cocktail Finder - Initializing 2 mapper for the Cocktail Finder - Two schema for Cocktail and CocktailResponse objects - Frontend pages for cockail finder home page, searching for cocktails page and creating own cocktail recipe page. - Reviewing pull requests of Gökay, Refika, Berk and Burakcan. - Arranging and collecting data from each member's codes for the submission of Practice-app Deliverables on June 10 - Unit tests for post and get methods of Cocktail Finder - Helper functions for Cocktail Finder - Downloading and collecting data about docker

	- Attending the meetings where we dockerized the practice app. - Attending the meetings where we deployed our project. - Helping others in zoom meetings for errors. - Creating, merging and collecting code segments from everyone at M2 report - Zoom meetings with İhsan for the M2 - Reviewing evaluation of tools and giving feedback - Reviewing Project plan and giving feedback - Creating issues after meetings for the tasks - Creating my issues
Refika Kalyoncu	- Movies Review GET and POST endpoint using Flask - Wiki documentation for movie reviews - Initializing 2 mapper for the movie reviews - Table for the movie object - Frontend pages for movie review home page, adding movie review page and searching for a movie review according to a keyword page. - Convert my .html files into .vue and helped Batuhan during conversion. - Reviewing pull requests of Batuhan Tongarlak and Veyis Turgut. - Gave comments to others' pull requests - Orginizing meeting dates and times using when2meet - Taking meeting notes for each meeting and uploading as as a wiki documentation - Unit tests for post and get methods of movie review - Downloading and collecting data about docker - Dockerizing movie review's backend in my local - Attending to the meetings where we dockerized the practice app. - Attending the meetings where we deployed our project. - Helping other in zoom meetings for errors. - Creating list of deliverables - Reviewing the final structure of Milestone 2 report - Submiting milestone 2 report - Reviewing evaluation of tools and giving feedback - Reviewing Project plan and giving feedback - Contacting with asistants and Suzan Üsküdarlı for organizing meetings and for asking the questions of the team as a communicator - Creating issues after meetings for the tasks - Merged my pull request with Veyis - Be a part of creating Release Tag - Creating issues for myself - Creating pull requests of my own.

Ömer Yılmaz	- Issues GET and POST endpoint using Flask - Wiki documentation for issues API - Creating tables for issues API - Implemented helper methods for Issues API -Implemented unit tests for issues API. - Implemented frontend pages for issues api using Vue. - Reviewing pull requests of Kürşat Talha Berk Yamanoğlu and Veyis Turgut. - Initialized dockerfile for backend. - Attending to the meetings where we dockerized the practice app. - Attending the meetings where we deployed our project. - Helped dockerization and deployment of the application. - I have created project plan using MindView application. - Creating issues for my jobs.
Gökay Yıldız	- Research on RESTful APIs, http requests - Research on public APIs to practice requests and use Postman to practice. - Research on Flask and try sample codes to get used to it. - Choose quote garden api to continue development. - Develop endpoint /quotes/, GET: Users can get quotes of the famous people by providing genre - Develop endpoint /randomQuotes/, GET: Users can get quotes of the famous people by random genre - Develop endpoint /addQuotes/, POST: Users can add quotes of anybody - I write helper methods to reduce repetition and provide modularity - Database connection is made by using sqlite3 - created a table that is necessary for quotes - Made necessary changes in folder structure as offered by Ömer and Veyis -Written mappers and schemas for the response I get and data I offer and for database - Test both the data coming from outsource api and the api that is created by me. - Try to learn vue.js to implement frontend. Burakcan gives us a quick tutorial and we can easily start. Also, Veyis and Ömer provide sample codes in vue. Reviewing and producing a working code is easier by those samples. - Implemented frontend in vue.js for quotes and make necessary router connections to the files that I created and add it onto the main page to redirect my page. - Checked dockerfile on local - follow deployment process on live broadcast of veyis, trying to help solving bugs while deploying. - merging some of the pull requests by me, ihsan and veyis - reviewing a lot of pull requests and codes. details explained in the personal report - created 3 pull requests 2 of them successfully merged, the other one canceled by me because of the high number of conflicts - Prepared usage manual documentation for all my endpoints both

	uploaded into github wiki, by personal report, and the doc we have submitted at Thursday. - Writen the evaluation of tools and processes used at this project development, ask other group members for specific usage - Review executive summary, project plan
Burakcan Kazım Yeşil	- Researched about Flask since I had no previous experience with it. - Searched for a - public API to use on my endpoints. - Tested the - JikanAPI to get familiar with it to use it on my endpoints. - Implemented GET - method for the /anime/search/ endpoint. This endpoint lets user to search the - Anime database of MAL. - Implemented GET - method for the /anime/ endpoint. This endpoint lets user to get detailed - information about a certain anime. - Implemented POST - method for the /anime/ endpoint. This endpoint lets user to post new animes to - the project's database. - Refactored the codebase - according to suggestions made by Omer and Veyis. These changes modulated the - methods that I've created and seperated into helper methods. This change eases - the testing process. - Made connection - to the database using sqlite3 package for python. - Created Tables - necessary for the AnimeAPI. - Created the initial frontend code for other members to use using Vue.js framework using minimal CSS and tried to keep it simple. - Written a manual to run the frontend code. - Tried to teach other about the Vue.js and help them understand the basics and structure of it to help them implement their own frontends. - Implemented Unittests for the AnimeAPI. These includes both successful and failed tests for each helper method and each endpoint method for the AnimeAPI. - Helped to write Dockerfile of the backend, since I had some experience with Docker before. - Wrote Dockerfile for the frontend. And created environment variable for it to connect to the backend without problems. - Wrote docker-compose.yml to run both of our containers together with Compose. - Dockerized and tested both frontend and backend locally. - Attended the meetings for Dockerizing and helped to create working images for both frontend and backend. - Helped to deploy our dockerized app to the AWS with the group. - Prepared documentation for all my endpoints and methods on the GithubWIKI, used Postman to create example request and returns. - Evaluated the deliverables for the Milestone 2 Report. - Created pull requests and issues along with my work.

Berk Yamanoğlu	- Implemented GET and POST methods for endpoint “/name_information” using Flask - Wiki documentation for my API endpoints. - Writing sql for creating table of name info model - Implemented helper methods for my API that connects to db. - Implemented unit tests for my endpoints. - Implemented frontend pages for Name/Age guess using Vue that makes use of the api functions I have provided. - Created Pull Request #133 for my implemented parts and made necessary changes in the code afterwards according to the feedbacks from my teammates. - Reviewing pull requests #175, #174, #130, and #126. - Helped Veyis with the merge of the different pull requests into the master. - Attending to the meetings where we dockerized the practice app. - Attending the meetings where we deployed our project. - Prepared Project Summary and Status for M2 report. - Creating issues for my jobs.

Codes of Each Member

BOOKS [‘GET]

Gets an author name from users and returns reviews of the books of that author using NYTimes API. All the work done in helper functions. Detailed information can be found in Veyis' individual report.

```
@app.route('/books/', methods=['GET'])
def get_books():
    # validate parameters
    x = books_helper.validate_input(request.args)
    if x is not None:
        return x
    # fetch results from 3rd party api
    books = books_helper.call_nytimes(request.args.get("name"))
    if type(books) != list:
        return books
    # now we have books of this author in books variable.
    # Let's store this book in database for further references.
    books_helper.add_books_from_nytimes(books)
    # don't return all books, return just as much as user wants
    books = books_helper.get_n(books, request.args)
    return books if type(books) != list else
schemas.BookResponse(num_results=len(books), books=books).__dict__
```

BOOKS [‘POST’]

Users can add their books to our database. Detailed information can be found in Veyis' individual report.

```
@app.route('/books/', methods=['POST'])
def create_book():
    if request.get_json() is None:
        return Response("Body is empty!", status=400)
    book = books_helper.validate_body(request.get_json())
    if type(book) is not schemas.Book:
        return book
    # try to insert book to DB, return forbidden upon failure
    db_response = books_helper.add_book_from_user(book)
    return db_response if db_response is not None else Response("Book added
succesfully!", status=200)
```

ISSUES [‘GET’]

```
@app.route('/download_issues', methods=['GET'])
def download_issues():
    param = {**request.args,
             'per_page': 100}
    if 'state' not in param:
        param['state'] = 'all'
    r =
requests.get("https://api.github.com/repos/bounswe/2021SpringGroup12/issues",
                     params=param).json()
    issue_list = [mapper.git_issue_mapper(element) for element in r]
    issue_helper.insert_multiple_issue(issue_list)
    total_issue = issue_helper.get_issue_count()
    return Response(f'{len(r)} issues are downloaded. There are total {total_issue}
issues in the system', 200)
```

Downloads issues of 2021SpringGroup12 from github api, simplifies them, puts into db.

ISSUES [‘POST’]

```
@app.route('/issues', methods=['POST'])
def post_issue():
    try:
        issue = mapper.issue_mapper(request.get_json())
    except:
        return Response(f'An error occurred while posting json. Json =
{request.get_json()}', status=400)
    issue_helper.insert_multiple_issue([issue])
    total_issue = issue_helper.get_issue_count()
    return Response(f'There are total {total_issue} issues in the system', 200)
```

Puts an issue to db

ISSUES [‘GET’]

```
@app.route('/issues/<int:number>', methods=['GET'])
def get_issue(number: int):
    issue = issue_helper.get_issue(number)
    if issue is None:
        print("Buradayım")
        return Response(f"Issue number {number} not found!", status=400)
    return jsonify(issue.__dict__)
```

Returns an issue from db.

ISSUES [‘GET’]

```
@app.route('/issues', methods=['GET'])
def get_all_issues():
    max_results = 30
    if request.args.get("max_results") is not None:
        max_results = request.args.get("max_results")
    issue_list = issue_helper.get_all_issues(max_results)
    return jsonify([issue.__dict__ for issue in issue_list])
```

Returns all issues in db.

ANIME [‘GET’]

```
@app.route('/anime/search/', methods=['GET'])
def search_anime():
    params = request.args
    # Validation
    validation = anime_helper.validate_search_params(params)
    if validation is not None:
        return validation
    # API Connection
    search_result = anime_helper.jikan_api_search(params)
    if type(search_result) != list:
        return search_result
    # Map result
    searched_animes = [mapper.searched_anime_mapper(
        anime).dict() for anime in search_result]
    # Return results
    return jsonify(searched_animes)
```

This method searchs the MyAnimeList and returns the results as a list of JSON objects.

ANIME [‘GET’]

```
@app.route('/anime/<int:id>', methods=['GET'])
def get_anime(id: int):
    anime_helper.jikan_api_get(id)
    # API Connection
    result = anime_helper.jikan_api_get(id)
    if type(result) == Response:
        return result
    # Map Result
    anime = mapper.anime_mapper(result).dict()
    # Add to DB
    db_response = anime_helper.add_mal_anime_to_db(anime)
    if type(db_response) == Response:
        return db_response
    # Return results
    return anime
```

This method gets and returns detailed information about a certain anime. It also Posts that Anime to our database.

ANIME [‘POST’]

```
@app.route('/anime/', methods=['POST'])
def post_anime():
    # Get the request body
    requestBody = request.json
    # Map to object
    post_anime = mapper.create_anime_mapper(requestBody).dict()
    # Add to DB
    database_response = anime_helper.add_user_anime_to_db(post_anime)
    return Response("Anime added successfully", status=200) if database_response is
None else database_response
```

This method accepts a JSON object which includes information about a new anime and posts it to our database.

QUOTES [‘POST’]

add_quote(): "/addQuotes/" - POST: Users can add quotes of anybody or themselves to the database

```
@app.route('/addQuotes/', methods=['POST'])
def add_quote():
```

```
    if request.get_json() is None:
        return Response("Body is empty!",status=400)
    quote = quote_helper.quote_validate(request.get_json())
    if type(quote) is not schemas.Quote:
        return quote

    db_response = quote_helper.add_quote_from_user(quote)

    return db_response if db_response is not None else Response("Quote added
succesfully!", status=200)
```

QUOTES [‘GET’]

get_quote_opt(): "/quotes/" - GET: Users can get quotes of the famous people by providing subject

```
@app.route('/quotes/', methods=['GET'])
def get_quote_opt():
    x = quote_helper.validate_input(request.args)
    if x is not None:
        return x

    quoted = quote_helper.call_quote_api(request.args.get("genre"))
    if type(quoted) != list:
        return quoted

    quotes = [mapper.quote_mapper(s) for s in quoted]

    quote_helper.add_quotes_quote_garden(quotes)

    return quotes if type(quotes) != list else schemas.QuoteResponse(data =
quotes).__dict__
```

QUOTES [‘GET’]

get_quotes(): "/randomQuotes/" - GET: Users can get quotes of the famous people by random subject

```
@app.route('/randomQuotes/', methods=['GET'])
def get_quotes():
    t = quote_helper.get_genres()
    # we get all genres and select one randomly to call
```

```
    rnd = int(random.uniform(0, len(t)))

    quoted = quote_helper.call_quote_api(t[rnd])
    if type(quoted) != list:
        return quoted

    quotes = [mapper.quote_mapper(s) for s in quoted]
    quote_helper.add_quotes_quote_garden(quotes)
    return quotes if type(quotes) != list else
schemas.QuoteResponse(data=quotes).__dict__
```

MOVIES [‘GET’]

```
@app.route('/movies_home/', methods=['GET', 'POST'])
def movies_home():
    return
```

Below method returns the home page endpoint since I started with html I needed an url for each page so that return render.template will return correctly after I turned my files into frontend I decided not the change the endpoint so that the keyword will have a completely different page which does not include 2 buttons.

MOVIES [‘GET’]

```
@app.route('/movies/', methods=['GET'])
def get_movies():
    if "keyword" not in request.args:
        return Response("Please provide a keyword!", status=400)
    # now we are sure keyword parameter is supplied, request rewiew the moves with
the keywords from NYTimes API.
    keyword = request.args.get("keyword").title().replace(" ", "+")
    r = requests.get(

"https://api.nytimes.com/svc/movies/v2/reviews/search.json?query={}&api-key={}".for
mat(keyword,NYTIMESKEY))
    r = r.json()

    # now we have the movies.
    # storing movies in database for further references.
```

```
    dict = {}
    if r["results"] != None:
        movies = [mapper.movie_mapper(s) for s in r["results"]]
        con = sqlite3.connect(DB_PATH)
        cur = con.cursor()
        for movie in movies:
            # -----------------------------------
            try:
                cur.execute("INSERT INTO Movie(display_title, mpaa_rating,
critics_pick, byline, headline,summary_short, link) VALUES (?,?,?,?,?,?,?)",
                            (movie.display_title, movie.mpaa_rating,
movie.critics_pick, movie.byline, movie.headline, movie.summary_short,
movie.link.url))
            except:
                # do nothing upon failure, this is not a critical process
                continue

        for movie in movies:
            display_title = movie.display_title
            movie_info = {"display_title": movie.display_title, "mpaa_rating":
movie.mpaa_rating, "critics_pick": movie.critics_pick,
                          "byline": movie.byline, "headline": movie.headline,
"summary_short":  movie.summary_short, "link": movie.link}
            dict[display_title] = movie_info
    return jsonify(dict)
```

It handles incoming get requests to "/movies/" endpoint. First it checks whether keyword is given as an argument or not if it didn't it gives a bad request error. Then it gets the keyword and and sends a request to the api as a "query" argument to the corresponding api. Then turns it into a json object. Then I initialized a dictionary which collects and stores the results which are movie reviews and turns them into a Movie object using a mapper. Then connects to the database and inserts the movies into the Movie table. Then adds these movies into dictionary, the key for each movie is the title of the movie and the value is all information about the review. Then returns this dictionary as a json object so that we can use in frontend side. This code also sends a request to NYTIMES API. It requires an api key to authorize which is embedded to `docker-compose.yml`.

MOVIES [‘POST‘]

```
@ app.route('/movies_addReview/', methods=['GET', 'POST'])
def create_movie_review():
    #movie_review = request.form.to_dict(flat=True)
    movie_review = request.get_json()
```

```
    # make sure that necessary information are given
    # title byline and url should be provided
    if movie_review == {}:
        return Response("Please provide the required information!", status=400)
    if "display_title" not in movie_review.keys():
        return Response("Please provide the title of the movie!", status=400)
    if "byline" not in movie_review.keys():
        return Response("Please provide the reviewer of the movie!", status=400)
    if "link" not in movie_review.keys():
        return Response("Please provide the link!", status=400)
    if "critics_pick" in movie_review.keys():
        if movie_review['critics_pick'] != "1" and movie_review['critics_pick'] !=
"0":
            return Response("Criticks pick must be 1 or 0")
    try:
        movie = mapper.movie_mapper2(movie_review)
        print(movie)
    except Exception as err:
        return Response(str(err), status=400)
        # connect to Database
    con = sqlite3.connect(DB_PATH)
    cur = con.cursor()

    # try to insert movie to DB, return forbidden upon failure
    try:
        cur.execute("INSERT INTO Movie(display_title, mpaa_rating, critics_pick,
byline, headline,summary_short, link) VALUES (?,?,?,?,?,?,?)",
                    (movie.display_title, movie.mpaa_rating, movie.critics_pick,
movie.byline, movie.headline, movie.summary_short, movie.link))
    except Exception as err:
        return Response(str(err), status=403)
    con.commit()
    con.close()
    return Response("Movie Review added succesfully!", status=200)
```

The upper method is for the post method. Instead of separating method as helpers and the function itself I chose to control the post method arguments inside the function itself. The method checks whether the user entered any information about movies if not it gives a bad request error and asks for the required information "Please provide the required information!". Also checks whether the user entered both title, byline and the link of the movie and checks if the user entered the critics' pick is it in the 0 1 format. After these checks if the json object can be mapped(using try-except) it inserts the json into the movie table.

NAME-AGE [‘GET’]

```
@app.route('/name_information', methods=['GET'])
def get_name_information():
    name = request.args.get("name")
    country = request.args.get("country")
    onApi = False
    onDB = False
    countryBased = False

    if name == None or name.strip() == "":
        return Response("Please provide the name!", status=400)

    api_url = f'https://api.agify.io/?name={name}'

    if country != None and country.strip()!="":
        countryBased = True
        api_url+=f'&country_id={country}'

    try:
        r = requests.get(api_url)
        response = r.json()

        if response["age"] != None:
            onApi = True
            countOnApi = response["count"]
            ageOnApi = response["age"]
    except Exception:
        pass


    #Check db for matching entries to add to the results coming from Agify api
    dbData = name_info_helper.get_name_info(name, country, countryBased)


    onDB = dbData!=None and dbData[1] != 0
    if onDB:
        ageOnDB = dbData[0]
        countOnDB = dbData[1]

    #Combine results coming from api and db
    if onApi and onDB:
        resultCount = countOnApi + countOnDB
```

```
        resultAge = ((countOnApi*ageOnApi)+(countOnDB*ageOnDB))/resultCount
    elif onApi:
        resultCount = countOnApi
        resultAge = ageOnApi
    elif onDB:
        resultCount = countOnDB
        resultAge = ageOnDB
    else:
        return Response("No registered data for this querry", status=200)
    result = schemas.NameInfoResponse()
    result.name=name
    result.age=int(resultAge)
    result.count=resultCount
    if countryBased:
        result.country = country
    return jsonify(result.__dict__)
```

NAME-AGE [‘POST’]

```
@app.route('/name_information', methods=['POST'])
def save_new_name_info():
    body = request.get_json()

    if body==None:
        return Response("Please provide correct information in body as json",
status=400)
    # make sure that necessary information are given
    if "name" not in body:
        return Response("Please provide your name to save to the database!",
status=400)
    if "age" not in body:
        return Response("Please provide your age to save to the database!",
status=400)
    if "country" not in body:
        return Response("Please provide your country to save to the database!",
status=400)

    try:
        nameInfo = schemas.NameInfo(
            name=body["name"],
            age=body["age"],
            country=body["country"])
    except Exception:
        return Response("Please provide correct information in body as json",
status=400)
```

```
    success = name_info_helper.insert_name_info(nameInfo)

    if not success:
        return Response("Your information couldnt be saved! Please try again
later!", status=403)

    return Response("Your name information has been added to database!", status=200)
```

CURRENCY CONVERSION [‘GET’]

Users can get the information of the daily currency exchange rate. Information includes date, currencies, exchange rate, calculated amount of given money. exchangerate.host API is used to get data. Then the data coming from exchange.rate API stored in our database for further reference.
-ihsan-

```
@app.route('/convert/', methods=['GET'])
def convert_currency():
    # necesssary info check
    x = currency_helper.validate_get_input(request.args)
    if x is not None:
        return x

    from_curr = request.args.get("from").upper()
    to_curr = request.args.get("to").upper()
    # get data from exchangerate api
    r = requests.get(
        'https://api.exchangerate.host/convert?from={}&to={}'.format(from_curr,
to_curr))
    r = r.json()
    # check "to" rate exitance
    check = currency_helper.non_existing_curr_rate_check(r)
    if check is not None:
        return check

    cr = mapper.currency_rate_mapper(r)

    # save the currency rate record to db if it is not in db
    currency_helper.add_db_from_exchangerate(cr)

    # calculate money with amount value if wanted
    r = currency_helper.calculate_amount(r, request.args)

    r.pop("historical")
```

```
    r.pop("motd")

    return r
```

CURRENCY CONVERSION [‘POST’]

POST function allows you to post a currency rate data for a specific date to database.-ihsan-

```
@app.route('/convert/', methods=['POST'])
def create_currency_hist():
    # check if the necessary information for the record is given. Otherwise 400
error.
    cr = currency_helper.validate_post_input(request.get_json())
    if type(cr) is not schemas.CurrencyRate:
        return cr
    # try to insert record to db.
    db_response = currency_helper.add_db_from_user(cr)
    return db_response if db_response is not None else Response("You have
successfully inserted your currency rate to db", status=200)
```

COCKTAILS [‘GET’]

```
@app.route('/cocktails/get_cocktails/', methods=['GET'])
def get_cocktails():
    x = cocktail_helper.validate_get_input(request.args)
    if x is not None:
        return x
        #cocktail_name has captured
    cocktail_name = request.args.get("cocktail_name")
        #taking data in json formats
    r =
requests.get("http://www.thecocktaildb.com/api/json/v1/1/search.php?s={}".format(co
cktail_name))
    r = r.json()

    check = cocktail_helper.non_existing_cocktail_name_check(r)
    if check is not None:
        return check

    cocktails = [mapper.cocktail_mapper(s) for s in r["drinks"]]

            # We can have more than one cocktail including the cocktail name
            # (margarita, blue margarita) or the keyword can be any place in the
name of the cocktail.
    cocktail_helper.add_cocktails_from_user(cocktails)
```

```
    return schemas.CocktailResponse(cocktails=cocktails).__dict__
```

Users can view the cocktails and their features with the keyword, this method will return all the cocktails including the keyword in its name.

COCKTAILS [‘POST’]

```
@app.route('/cocktails/create_cocktail/', methods=['GET','POST'])
def create_cocktail():
    cocktail_fields=request.get_json()
    if cocktail_fields is None:
        return Response("Body is empty!",status=400)
    cocktail = cocktail_helper.validate_post_cocktail(cocktail_fields)
    if type(cocktail) is not schemas.Cocktail:
        return cocktail
        # connect to Database
    db_response=cocktail_helper.add_cocktail_from_user(cocktail)

    return db_response if db_response is not None else Response("Cocktail added
succesfully!", status=200)
```

Users can post their cocktail recipes to the database