CmpE 352 - Milestone 2
Group Report

GROUP 1

Kadir Gökhan Sezer
Osman Fehmi Albayrak
Ece Sarkın
Ege Onur Tağa
Efekan Kavalcı
Mustafa Atay
Harun Erkurt
Kamil Korkut
Ahmet Yazıcı
Ömer Özdemir

20.05.2022

CONTENTS

.

1. Executive Summary

1.1 Summary of project and overall status

We as a group, implemented a learning platform that uses django template as a boilerplate. The project consists of several user implemented functions as well as default django files. We initially made a requirements list to be able to use the helpful packages of python built in virtual environment. We then concluded on building a learning platform that has students and teachers as users. Users can sign up and login to the site. Students can view their courses, enroll in courses, give rate to courses, and specify their preference just like a semantic search. Teachers on the other hand can add new courses, view course statistics, view their courses and delete the existing courses. We have use a modular method to check edge cases and possible errors. We used several html files and a little bit of styling via css. Html files are for the user interface. We link the html and py paths with a urls.py file that each of us have edited along the way. We made use of mysql, we made insertions, deletions and queries over it via python. We have made several unit tests to test every possible case that can occur. Our project is running without errors, dockerized, and deployed. Necessary running instructions are added to the readme file elaborately.

We as team members created API endpoints satisfying GET and POST methods and tested them with our unit tests. Moreover we performed tests using POSTMAN and also our shell curl commands.

Now, we believe, as a team we are ready to implement a fully functional learning platform, not a practice version.

1.2 URL of the Deployed Application

Deployed Application URL: 18.234.216.208:8000/

1.3 API Documentation

Link to the API Documentation File: API Documentation

Authentication

POST /doLogin/ doLogin

Description

This api simulates a login attempt. If the given username and password matches, the corresponding user's last login time is updated in the database.

Parameters

Name	Description
username *required varchar(200) (path)	Username of the user that wants to login.
password *required varchar(200) (path)	Password of the given user

Responses

Code	Description

200	OK
	
	Example Response bodies:
	`{'loggedIn': True}`
	`{'loggedIn': False, 'error': 'Wrong username or password.'}`

250	Bad Request
403	Forbidden
404	Not Found

GET /canLogin/ canLogin

Description

This api returns whether a user can login using the given credentials (username and password)

Parameters

Name	Description
username *required varchar(200) (path)	Username of the user that wants to login.

password *required varchar(200) (path)	Password of the given user

Responses

Code	Description
200	OK Example Response bodies: {'canLogin': True} {'canLogin': False, 'error': 'Wrong username or password.'}
250	Bad Request
403	Forbidden
404	Not Found

Course Statistics

POST /teacher/teacher_save_course_statistics/

teacher_save_course_statistics

Description

This api saves the statistics of a desired course instantly in the table named "Course Statistics History". In this way, you keep the data of the time-based statistics of the courses.

Parameters

Name	Description
course_name *required varchar(200) (path)	The course you want to save the statistics instantly

Responses

Code	Description
200	OK

201	Created
401	Unauthorized
403	Forbidden
404	Not Found

GET /teacher/teacher_get_course_statistics/ teacher_get_course_statistics

Description

This api returns the current statistics of a desired course as Json.
Model: "Course Name", "Teacher Name", “Total Students”, “Ratings”, “Rate Count”, “Last Statistics View Time”, “Last Course View Time”

Parameters

Name	Description
course_name *required varchar(200) (path)	The course you want to see statistics for

Responses

Code	Description
200	OK Example Value | Model {"data": [["CMPE150", "Mustafa Atay", 2, 10.0, 2, null, null]]} {"data": [["courseName", "TeacherName", TotalStudents, Ratings, RateCount, lastStatistics ViewTime, lastCourseViewTime]]}
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

Give Rate

POST /student/student_give_rate/post/ student_post_rate

Description
This api updates the rating of a course in the database. The new rate and new rate count are returned in the JSON response.

Parameters

Name	Description
course_name *required varchar(200) (path)	Name of the course the user wants to rate.
rate *required int (path)	Desired rating for the course that is being rated. Integer from 0 to 5.

Responses

Code	Description
200	OK
400	Bad Request

GET /student/student_give_rate/get student_get_rate

Description
This api retrieves the rating of a course in the database. The rate of the course is returned in the JSON response.

Parameters

Name	Description
course_name *required varchar(200) (path)	Name of the course the user wants to rate.

Responses

Code	Description
200	OK
400	Bad Request

Add Course

POST /teacher/do_add_course/ do_add_course

Description

This method adds the course to the system.

Parameters

Name	Description
course_name *required varchar(200) (path)	Course you want to add.

Responses

Code	Description
200	OK
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

GET /teacher/can_add_course/ can_add_course

Description

This function checks if the course is a valid (non existing in the system) course

Parameters

Name	Description
course_name *required varchar(200) (path)	The course you want to check.

Responses

Code	Description
200	OK
201	Created

401	Unauthorized
403	Forbidden
404	Not Found

Student enroll

POST /student/student_enroll_entered/ student_enroll_entered

Description:
This API makes the student enroll to the desired course.

Parameters

Name	Description
course_name `varchar(200)` `(path)`	The course you want the logged in user to enroll

GET /student/student_enroll/ student_enroll

Description:
This API returns a list of all the courses a student is enrolled to.
Model: {"course_name"}

Name	Description
course_name `varchar(200)` `(path)`	The course name the student is enrolled to.

Student-my-courses

POST /student/student_my_courses/seen/ student_my_courses_seen_update

Description

This api updates a user’s last seen time of his/her own courses.

Parameters

Name	Description
username *required **varchar(200) (path)**	Username of the user that wants to login.

Responses

Code	Description
200	OK Example Response bodies: {“message”: “Updated!”}
250	Bad Request
403	Forbidden
404	Not Found Example Response bodies: {“message”: “Student not found!”}

GET /student/student_my_courses/get/ getAllCourses

Description

This api returns enrolled courses for the given username. If the given username is found in the database, the user's courses will be returned, otherwise it sends an error message in String format.

Parameters

Name	Description
username *required **varchar(200) (path)**	Username of the user whose courses will be seen.

Responses

Code	Description
200	OK Example Response bodies: {'data': [“CMPE150”, “CMPE160”]} {'data': []}
250	Bad Request
403	Forbidden
404	Not Found Example Response bodies: {“error”: “Student not found!”}

Specify Preferences,

host: **18.234.216.208:8000** **- send the requests there.**

POST /student/preferences/post/ **student_preferences_post**

Description

This api takes student_username, password, topic and level and adds the topic and level, together constituting the preference of that user to the Preference table in the database and returns the added entry with status=True if the call is successful. Note that (student_username, topic) pair is unique. Author:Ege Onur Taga

Parameters

Name	Description
student_username *required varchar(200) (path)	Username of the student user that wants to add a preference.
password *required varchar(200) (path)	Password of the given student user
topic *required varchar(200) (path)	Topic that the student is interested in
level *required varchar(200) (path)	Level that the student wants to learn the topic. Must be one of: “Beginner”, “Intermediate” or “Advanced”, case sensitive.

Responses

Code	Description
200	OK Example Response bodies: {"status": "true", "student_username": "quanex7", "topic": "Data Science", "level": "Advanced"}

250	Bad Request
403	Forbidden
404	Not Found

Example call: curl --location --request POST '18.234.216.208:8000/student/preferences/post/' \
--form 'student_username="quanex7"' \
--form 'password="123123g"' \
--form 'topic="Data Science"' \
--form 'level="Advanced"'

GET /student/preferences/get/ student_preferences_get

Description

Given the student username, this API returns the preferences of the student user as a JSON file consisting of topic and level.

Parameters

Name	Description
student_username *required varchar(200) (path)	Username of a student that you want to learn the preferences about.

Responses

Code	Description
200	OK Example Response bodies: `{"preference_1": {"topic": "Mathematics", "level": "Intermediate"}, "preference_2": {"topic": "Physics", "level": "Advanced"}, "preference_3": {"topic": "Topology", "level": "Beginner"}}`
250	Bad Request
403	Forbidden
404	Not Found

Example call: curl --location --request GET
'18.234.216.208:8000/student/preferences/get/?student_username=quanex7'

Teacher My Courses

POST **/teacher/teacher_update_course/** teacher_update_course/

Description

This api updates the name of the course that is given by the logged teacher. Logged teacher cannot update the name of the course which is not added to the database before.

So, you can update the course name after adding it to the database.

Parameters

Name	Description
course_name `varchar(200) (path)`	The course name you want to change.
new_name `varchar(200) (path)`	New name of the course.

Responses

Code	Description
200	OK Example response bodies {"quanex1": “Success”}
404	Cannot added Example response bodies {"quanex1": “Error”}

GET /teacher/teacher_get_courses/ teacher_get_courses

Description

This api returns the current list of courses that are given by the teacher whose username is given.

Parameters

Name	Description
username `varchar(200) (path)`	The username of the teacher whose course list will be displayed

Responses

Code	Description
200	OK Example response bodies {"quanex1": [[“CMPE150”, “CMPE160”]]}
404	Not Found Example response bodies {“quanex12”: “User is not registered or not a teacher"}

Teacher-Delete-Course

GET /teacher/teacher_delete_course

Name: Kadir Gokhan Sezer

Database:

Courses Table

course_name: text
teacher_username: text
rating:fload
rate_count: integer

Description

This endpoint is called with the button on the teacher page. The purpose in it shows the active course_name's of the teacher who has logged in. This endpoint/page is used by the teacher when he wants to delete his own course. When he enters via a browser, he sees a textbox. He writes the name of the course he wants to delete here. When you press the button, it deletes the lesson. And it can additionally display a cat photo. Apart from these, there is a button to go to the homepage. Finally, there is the button to undo the last action.

Parameters

Name	Description
***1:username** *required* `varchar(200) (path)`	Username of the user that wants to login.
***1:password** *required* `varchar(200) (path)`	Password of the given user

*1:This parameters are taken from the session, you do not have to send them.

Responses

Code	Description

200	OK
	
	Example Response bodies:
	`{'my_teacher_courses_list':`
	`[course_name,course_name,..],`
	`'photo':(str) some_url,`
	`'enter_text': str text`
	`}`

400	Bad Request
401	Unauthorized
403	Forbidden

404	Not Found
50x	Bad Gateway

Teacher-Delete-Course-Entered

POST /teacher/teacher_delete_course_entered

Name: Kadir Gokhan Sezer

Database:

course_name: text
teacher_username: text
rating:fload
rate_count: integer

Description

This page is triggered by the submit button on the /teacher-delete-course page. It deletes the parameter (course_name) it received from the database. There are some rules for this deletion process. Here are the rules:
- The course you want to delete must belong to you.
- The course you want to delete must be in the database.

Parameters

Name	Description
***1:username** *required varchar(200) (path)	Username of the user that wants to login.
***1:password** *required varchar(200) (path)	Password of the given user
coursename* required varchar(200)	Name of the coursename(course_name in db)

*1:This parameters are taken from the session, you do not have to send them.

Responses

URL	Description
/?status=X	X can be Success or Fail.

Teacher-Delete-Course-Undo

POST /teacher/teacher_delete_course_undo

Name: Kadir Gokhan Sezer

Database:

course_name: text
teacher_username: text
rating:fload
rate_count: integer

Description

This endpoint is triggered by the Undo Process button on the /teacher_delete_course page. There are some rules for this endpoint to work. Here are the rules:

- When you want to undo your transaction, it is based on your last attempt (not the last successful transaction). If not, it throws an error.

Parameters

Name	Description
***1:username** $*^{required}$ `varchar(200) (path)`	Username of the user that wants to login.
***1:password** $*^{required}$ `varchar(200) (path)`	Password of the given user
lastdeleted $*^{required}$ `varchar(200)`	The course_name of the course that was deleted on the last attempt.

*1:This parameters are taken from the session, you do not have to send them.

Responses

Code	Description
204	OK Example Response bodies: {'my_teacher_courses_list': [..], 'photo':(str) some_url, 'enter_text': "Success.You saved the course you deleted before." }
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
408	Example Response bodies: {'my_teacher_courses_list': [..], 'photo':(str) some_url, 'enter_text': "Failed.You never deleted a course or the last attempt was not valid." }
409	Example Response bodies: {'my_teacher_courses_list': [..], 'photo':(str) some_url, 'enter_text': "Failed.The course is already in the db." }
50x	Bad Gateway

Get-Courses

GET /teacher/getcourses

Name: Kadir Gokhan Sezer

Database:

course_name: text
teacher_username: text
rating:fload
rate_count: integer

Description

This endpoint returns the user's course_names.

Parameters

Name	Description
username *required varchar(200) (path)	Username of the user that wants to login.
password *required varchar(200) (path)	Password of the given user

*This parameters are taken from the session, you do not have to send them.

Responses

Code	Description
200	OK Example Response bodies: {'courses':[course_name,course_name,..] }
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
50x	Bad Gateway

Sign UP

POST /sign_up_entered/

Description

This api saves the given user credentials into the database to the Users Table.

Parameters

Name	Description
post_name_surname *required varchar(200) (path)	The name and surname of the user you want to register into the system.
post_username *required varchar(200) (path)	The username of the user you want to register into the system.
post_password *required varchar(200) (path)	The password of the user you want to register into the system.
post_is_teacher boolean (path)	If the parameter is True, the user will be a teacher. If it is False, then the user will be a student.

Responses

Code	Description
200	OK
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

GET /see_all_registered_users/

Description

This api returns the current registered users of a desired database as Json.

Parameters

Name	Description
None	No parameter is needed for this api

Responses

Code	Description
200	**OK** **Example Value | Model** **{"RegisteredUsers": [["faith.hancock"], ["faith.hancock111111111"], ["faith.hancock414141"], ["faith.hancock414141414"], ["faith.hancock456"], ["faith.hancock555"], ["faith.hancock666666"], ["quanex"], ["quanex1"], ["quanex2"], ["quanex3"], ["quanex4"], ["quanex5"], ["quanex6"], ["quanex7"], ["quanex8"], ["quanex9"]]}** **{ RegisteredUsers:[["username"],["username"],… }**
201	**Created**
401	**Unauthorized**
403	**Forbidden**

404	**Not Found**

GET /sign_up_check_username/

Description

This api searches the database whether there is a user in the database with the given username.

Parameters

Name	Description
post_check_username	**The username which is wanted to search in the database whether it exist or not. If post_check_username already exists in database, this api will return "Sorry! This username is not available.".** **Otherwise, it will return "Yes! This username is available."**

Responses

Code	Description
200	**OK** **If check_username exists à"Sorry! This username is not available.".** **Otherwise à "Yes! This username is available."**
201	**Created**

401	**Unauthorized**
403	**Forbidden**
404	**Not Found**

1.4 Basic functionality of the project

As explained in the introduction, this is a practice application designed to give us hands-on experience on implementing real-world software. Our application has authentication and sign up mechanism just like the real world application does. It provides some functionalities in a pretty good manner like sign-in or sign-up or specify preferences. Everyone worked in different lines of developments and at the end, we collected all the development lines in a single branch, main. Now, the application is also dockerized and runs at the URL given at the beginning of this milestone report.

1.5 Challenges

One of the most difficult parts of the project was to merge many branches of development into a single branch. It was pretty challenging, since everyone goes at a different pace and trying to synchronize everyone is pretty tiresome.

Moreover, we used the databases explicitly and it was problematic as well, since then all parts of the project, let it be dockerization or deployment gets complicated as well. Not only dockerization, we also experienced problems in the unit testing, it took a great amount of time to solve these problems. One thing we all learned is to let the use of Database to Django's management.

Writing explicit SQL statements did nothing but harm us in our journey of development. Luckily we solved these problems but it exhausted us more than it should have done.

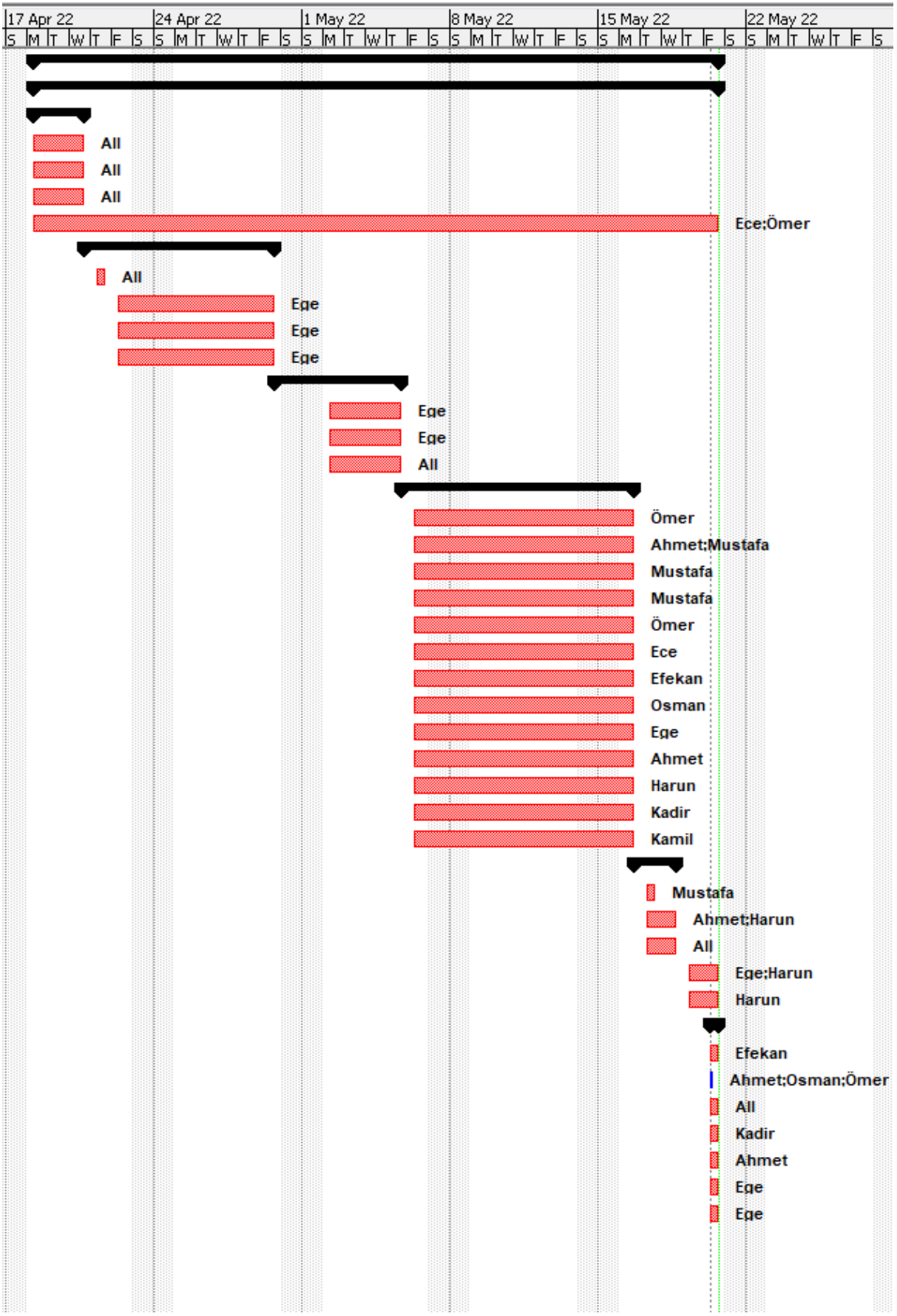
1.6 List and Status of deliverables

The table below shows the deliverables and their status.

Deliverables	Status
Group Report - Milestone 2	**Finished on 20.05.2022**
Practice App(Source Code)	**Finished on 20.05.2022**
Test Codes of Practice App	**Finished on 20.05.2022**
Requirements	**Finished on 19.05.2022**
Design	**Finished on 19.05.2022**

2. Project Plan

	Name	Duration	Start	Finish	Resource Names
1	**Prepairing Deliverables for Milestone 2**	**25 days?**	**4/18/22 8:00 AM**	**5/20/22 5:00 PM**	
2	**Practiceapp**	**25 days?**	**4/18/22 8:00 AM**	**5/20/22 5:00 PM**	
3	**Research**	**3 days?**	**4/18/22 8:00 AM**	**4/20/22 5:00 PM**	
4	Restful APIs	3 days?	4/18/22 8:00 AM	4/20/22 5:00 PM	All
5	Git usage	3 days?	4/18/22 8:00 AM	4/20/22 5:00 PM	All
6	Evaluation of Tools	3 days?	4/18/22 8:00 AM	4/20/22 5:00 PM	All
7	Meeting Notes	25 days?	4/18/22 8:00 AM	5/20/22 5:00 PM	Ece;Ömer
8	**Requirements**	**7 days?**	**4/20/22 5:00 PM**	**4/29/22 5:00 PM**	
9	Determining Fundemental Features	1 day?	4/20/22 5:00 PM	4/21/22 5:00 PM	All
10	Prepairing Template for Requirements	6 days?	4/21/22 5:00 PM	4/29/22 5:00 PM	Ege
11	Specifying Project Requirements	6 days?	4/21/22 5:00 PM	4/29/22 5:00 PM	Ege
12	Prepairing Glossary	6 days?	4/21/22 5:00 PM	4/29/22 5:00 PM	Ege
13	**UML Designs**	**4 days?**	**4/29/22 5:00 PM**	**5/5/22 5:00 PM**	
14	Use Case Diagram	4 days?	4/29/22 5:00 PM	5/5/22 5:00 PM	Ege
15	Class Diagram	4 days?	4/29/22 5:00 PM	5/5/22 5:00 PM	Ege
16	Sequence Diagrams	4 days?	4/29/22 5:00 PM	5/5/22 5:00 PM	All
17	**Implementation (API, HTML, Unit test)**	**7 days?**	**5/5/22 5:00 PM**	**5/16/22 5:00 PM**	
18	Sample data	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ömer
19	Initial pages	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ahmet;Mustafa
20	User guards	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Mustafa
21	Login	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Mustafa
22	Sign up	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ömer
23	Student enroll	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ece
24	Give rate	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Efekan
25	Student my Courses	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Osman
26	Specify preferences	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ege
27	Teacher add Course Page	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Ahmet
28	Course statistics	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Harun
29	Course delete	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Kadir
30	Teacher my courses	7 days?	5/5/22 5:00 PM	5/16/22 5:00 PM	Kamil
31	**Documentation**	**2 days?**	**5/16/22 5:00 PM**	**5/18/22 5:00 PM**	
32	Read.me	1 day?	5/16/22 5:00 PM	5/17/22 5:00 PM	Mustafa
33	API documentation template	2 days?	5/16/22 5:00 PM	5/18/22 5:00 PM	Ahmet;Harun
34	API documentation	2 days?	5/16/22 5:00 PM	5/18/22 5:00 PM	All
35	Deployment	2 days?	5/18/22 5:00 PM	5/20/22 5:00 PM	Ege;Harun
36	Dockerization	2 days?	5/18/22 5:00 PM	5/20/22 5:00 PM	Harun
37	**Milestone 2 Report**	**1 day?**	**5/20/22 8:00 AM**	**5/20/22 5:00 PM**	**All**
38	Completeness of Milestone 2 Report	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	Efekan
39	Project Plan	0.333 days?	5/20/22 8:00 AM	5/20/22 10:40 AM	Ahmet;Osman;Ömer
40	Work done by each member	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	All
41	Evaluation process of tools	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	Kadir
42	Summary of the project	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	Ahmet
43	Basic functionality	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	Ege
44	Challenges we met	1 day?	5/20/22 8:00 AM	5/20/22 5:00 PM	Ege


17 Apr 22
24 Apr 22
1 May 22
8 May 22
15 May 22
22 May 22
All
All
All
Ece;Ömer
All
Ege
Ege
Ege
Ege
Ege
All
Ömer
Ahmet;Mustafa
Mustafa
Mustafa
Ömer
Ece
Efekan
Osman
Ege
Ahmet
Harun
Kadir
Kamil
Mustafa
Ahmet;Harun
All
Ege;Harun
Harun
Efekan
Ahmet;Osman;Ömer
All
Kadir
Ahmet
Ege
Ege

3. Summary of Work Performed

Name	Work	Issue & Pull Request links	Wiki and external links
Ece Sarkın			
	Overall Summary: I have kept the record of the meeting notes, and uploaded them to the wiki page. I have written API functions for the practice applications but couldn't test my code.		
	Meeting #8 **Meeting #9** **Individual report** **Practice-app: student enroll API functions**	**#210** **#223** **#230** **#229**	**Meeting notes #8** **Meeting notes #9** **on Moodle** **PR#213 and PR#224**
Efekan Kavalcı			
	Overall Summary: I have implemented the give rate function of the practice app. I searched for an external api to use and decided on an api which provides famous quotes with different categories, I chose 'learning' category for the quotes since it is related to the context. I have done my unit tests for the GET and POST API in my functionality. I reviewed some of the pull requests and left comments. I gave some contributions to the use case diagram and created sequence diagrams for my functionality. For the report, I created the layout by examining the submission description and I have reviewed and then put the API Documentation.		
	Implementation of student_give_rate functionality	**#197 , #212**	
	Adding external api key variable to README.md	**#192**	
	Unit tests for give_rate APIs	**#233 , #222**	
	Use case diagram review and minor additions	#211	Use Case Diagram
	Sequence diagrams for give_rate APIs	**#218**	Sequence Diagrams
	Milestone 2 report layout	**#232**	
	API Documentation	**#263**	
Ege Onur Tağa			
	Overall Summary: I took an active role in all phases of the practice application assignment. I implemented Specify Preferences functionality of the assignment and wrote REST API endpoints satisfying the get and post methods for specifying preferences. Moreover, I reviewed many pull requests sent by other team members and tried to give satisfying reviews as much as I could. I detected bugs and notified team members or solved the bugs by myself. In addition, I have written three unit tests, which test the running of the API's I have created. On top of that, I designed and created requirements and glossary pages of the practice app and added contributions to class and use case diagrams by modifying and adding classes or use cases. For the deployment and docker image creation, I worked with Harun Erkurt and tried to solve the problems in dockerization and deployment. I tried to contribute to the dockerization and deployment as much as I could and was present during the dockerization of the application and runned the docker image on my local as wel and tested the codel. In addition, I prepared user accaptence tests. I contributed to the preperation One of the greatest contributions of this project was to learn how to work with many people on a single project.		
	PracticeApp-Bugfix: Removed imports from asyncio.windows events and types	**#187**	**Pull#186**

	PracticeApp-Implementation: Functionality Change and Database Changes	**#191**	**Pull#190**
	PracticeApp-Implementation: Specify Preferences Functionality is implemented and API tests are added for specify preferences endpoints		**Pull#199**
	PracticeApp-Testing: Added one more unit test		**Pull#203**
	PracticeApp-Bugfix: Fixed the broken Sign Up functionality	**#205**	**Pull#208**
	PracticeApp: Preparing Requirements and Glossary	**#207**	**Glossary and Requirements for Practice App**
	PracticeApp: Additions to the Class Diagram	**#209**	
	PracticeApp: Changes in the Use Case Diagram and review of it	**#211**	
	PracticeApp- Sequence Diagram: Specify Preferences	**#214**	
	PracticeApp: Adding Class and Use Case Diagrams to wiki pages and the navigator	**#215**	
	PracticeApp: I have helped Harun Erkurt in dockerizing and deploying the practice application.		
	Milestone2: Group Report - API documentation and preparing the deliverables: requirements, and diagrams.		
Harun Erkurt			
	Overall Summary: I took an active role in all phases of the practice application assignment. I developed teacher_course_statistics.py python code. teacher_course_statistics.html code. I tested all my code and developed a Unit test. I wrote REST API endpoints satisfying the get and post methods for specifying preferences. Moreover, I reviewed many pull requests. I have simply run the code in my local after I pull the branch, and merged with a "reviewed.". I detected bugs and solved all of them. I made the code be ready to run in the docker environment. I wrote Docker and docker-compose.yml files. I made Dockerization, and fixed errors. I set the database environment. I created an Account on AWS. I created MySql Database instance. All necessary operations will be done on AWS and the code will be deployed and run over docker by me.		
	Developing Teacher Course Statistics Code	**195**	
	Unit Test For Teacher Course Statistics	**220**	
	Group meeting for report	**226**	
	Dockerization	**244**	

	Deployment	**259**	
	Preparing Docker Document	**261**	
	Preparing Template For API DOC	**267**	
Kadir Gökhan Sezer			
	Overall Summary: First of all, this project was the most important for me among them, I can say that it was the most valuable, and in addition, it was the most time-consuming. I can say that I am really struggling with time. Constant meetings, constant problems, 100+ messages in the permanent whatsapp group... The most important of these is that we only worked with backend projects before, but we had not worked on such a complex and interconnected project. We had to code both HTML, sql, and python at the same time. Sometimes it took days just to understand, let alone solve the problems. I can say that I spent my days on the django application written in the lesson. Which function calls which, which functions which, was really very complicated and challenging. I guess you can learn something like that. As for the importance and value, prior to this lesson I was very keen to learn to use Git. When I researched, I can say that I was very happy when I heard that this course will teach you to use git. In addition, I undertake the task of api specialist in the company I work for. I can say that I am very curious about API. You can understand the importance of learning these two topics in this course. For this reason, I see everything I learn as a tool that I can use for life, not for homework. First of all, I carefully examined the api (https://apiblueprint.org) page. I already use it every day at the company. However, I started to learn the system I use every day, step by step, from the very beginning. I can't help saying that it is a great feeling to use the knowledge I learned at school in the evening at work. Thanks to this practice app, I can say that I have added a lot to my own professional project. Then, after a few request-response attempts, I started to feel competent about api. After this api system, it was time to learn how to use git nicely. For Git, I watched the PS first and then used a good resource (https://learngitbranching.js.org/) This site is amazing. After taking care of these, all that was left was to learn enough about unit-test and django. I managed this with help from my group. I was amazed by the beauty of Django, it's a really good framework. I learned what the MVC system is, which problem is the solution, but it was not easy. Learning MVC helped me understand this confusion which file comes from where, who calls which function. And as a feedback to myself, I don't think I've done the MVC distinction very well in this project. I don't see this as very abnormal either because it's our first project after all. I think I am good at entry level in both subjects. While I was initially trying to use it for Unittest, I found out that it has a sub-library for django. I can say that I used it and showed it to my friends. Even though another friend gave the start of our own project on Practice App, I also made a django project on my own. Thank you for this quality project. I can say that I learned information that I will never forget for the rest of my life.		
	Milestone 2 - Tools and Processes	**#235**	**#235**
	Milestone 2 - Summary of Work Performed	**#236**	**#236**
	practiceapp - path:teacher_delete_course	**#239**	**#239**
	practiceapp - path:teacher_delete_course_entered	**#240**	**#240**
	practiceapp - path:teacher_delete_course_undo	**#241**	**#241**
	practiceapp - path:getCourses	**#242**	**#242**
	practiceapp- API documentation	**#248**	**#248**
	practiceapp : Find a external API	**#251**	**#251**
	Milestone 2 : Uri of the tag of the project	**#255**	**#255**
	Research on Unit-test		
	Milestone 2 : List and status of deliverables	**#268**	**#268**
Kamil Korkut			

	Overall Summary: I have written the teacher_my_courses part of the project. I have written the features of showing the course list of the logged teacher, current price of bitcoin, showing the course list of the teacher whose username is given by get method, updating the name of the course whose owner is the logged teacher with given course name and new course name by post method. I have written a html page so that the functionalities can be used. Then, I have written the unit test of my part. I have made pull requests, reviewed and approved pull requests, created issues, attended some of the group meetings. I have written my personal report of the project. I have studied git and Django.		
	practice-app: teacher_my_courses **practice-app: teacher_my_courses unit test**	**#237** **#238**	**PR#189** **PR#204**
Mustafa Atay			
	Overall Summary: I was responsible for creating the login page. However, I took further responsibility by creating a codebase. I bootstrapped the django project. I created templates for HTML files for teamuse. I also wrote CSS code for a good website theme. In addition to these, I wrote guard functions which made my team's work easier. After all of this, I worked on my part. I implemented one POST and one GET endpoint. They were canLogin() and doLogin(). I used an external API called "Go Quotes API". I wrote unit tests to test my endpoints. I made 5 pull requests and reviewed 5 of my teammates' pull requests. I created class and use-case diagrams and my individual report.		
	initilization pr	**#175**	**#175**
	initial pages pr	**#182**	**#182**
	login logout pr	**#200**	**#200**
	creating individual report	**#227**	**#227**
	creating class and use-case diagrams	**#231**	**#231**
	code reviews	**#176, #190, #202, #203, #222**	**#176, #190, #202, #203, #222**
	More pull requests	**#177, #188, #258**	**#177, #188, #258**
Osman Fehmi Albayrak			
	Overall Summary: I have implemented the "My Courses" page for students. In this page I have displayed courses that logged in students attend. Also, I have made an external API call to "JokeAPI" and displayed the joke on my page. I have prepared API documentation for my APIs. I have performed unit tests on my unit. I prepared the "Project Plan" for the Milestone 2 with Ahmet Yazıcı and Ömer Özdemir. I made 2 pull requests, one of them for guiding others and the other is the final of my code. I attended meetings and discussed the practiceapp. I have reviewed Ahmet Yazıcı's pull request. I prepared a summary of work done by me for the Milestone 2.		
	External API and Returning JSON object	**#185**	
	Unit tests for my unit	**#249**	
	HTML and API file implementation	**#250**	
	API Documentation	**#252**	
	Project Plan for Milestone 2	**#254**	
	Pull request on external APIs and JSON object return	**#188**	
	Pull request for final	**#201**	
	Summary of work done by me	**#256**	

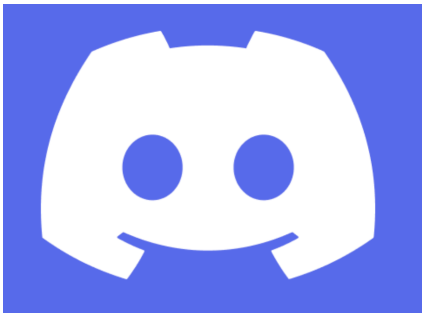
	Bug fixes for deployment	**#257**	
	Code reviews	**#258**, **#198**	
	Sequence diagrams	**#265**	**Sequence diagrams**
Ömer Özdemir			
	Overall Summary: I was responsible for creating the sign up page. However, I took further responsibility by creating create_db.py file and creating sign_up.py, sign_up.html, student_enroll.py, student_enroll.html student_give_rate.py, student_give_rate.html, student_my_courses.py , student_my_courses.html teacher_add_course.py, teacher_add_course.html teacher_course_statistics.py, teacher_course_statistics.html teacher_delete_course.py, teacher_delete_course.html teacher_my_courses.py, teacher_my_courses.html where I filled .html files with corresponding person's name and filled .py files with necessary imports and functions definitions and left body of those functions ##add your code## to make others job easier and prevent merge conflict. Then, I linked those API request via urls.py. After all of this, I worked on my part. I implemented one POST and one GET endpoint. They were can_sign_up() and do_sign_up(). I used an external API called "Random username". I wrote unit tests to test my endpoints. I made 8 pull requests and reviewed 13 of my teammates' pull requests. I created my individual report and prepared a project plan for milestone 2 with Ahmet and Osman..		
	Create_db.py is created	**#176**	
	drop.py is created	**#178**	
	creating sign_up.py, sign_up.html, student_enroll.py, student_enroll.html student_give_rate.py, student_give_rate.html, student_my_courses.py , student_my_courses.html teacher_add_course.py, teacher_add_course.html teacher_course_statistics.py, teacher_course_statistics.html teacher_delete_course.py, teacher_delete_course.html teacher_my_courses.py, teacher_my_courses.html where I filled .html files with corresponding person's name and filled .py files with necessary imports and functions definitions and left body of those functions ##add your code## to make others job easier and prevent merge conflict. Then, I linked those API request via urls.py.	**#180**	
	External api (random username) is implemented in sign_up is done	**#202**	
	Unit test for sign_up is done	**#221**	
	Sign up a minor bug-fix	**#225**	
	External API and Returning JSON object	**#185**	
	Creating individual report for milestone2	**#262**	
	Code reviews	**#175**, **#186**, **#190**	

		#194 #198 #206 #213 #217 #219 #234 #243 #246 #257	
	Preparing project plan	#254	
Ahmet Cemil YAZICI			
	Overall Summary: I initially made a research over django template and how to use it correctly, then me and my friends took an initiative and prepared the initial pages. For that part i took care of teacher.html, student.html, teacher.py, student.py. I have choose to do teacher add course to the system part of the project. I then made a research over which packages to use to be helpful to the project. We listed a requirements text and started the project, I first implemented a simple adding course function and a simple html. After learning about unit tests and needing an external api, methods to support GET, POST; i changed my code a little bit and added unit tests. My functions consists of a checker (can_add_course), a function to call the external random sentence generator API, a function to add to database (do_add_course) and two rendering functions to handle calling external api, redirecting to main page and giving success or fail messages. I have opened several Issues namely issues number 174,185 and 226. I reviewed a bunch of my coworkers pull requests, and made a bunch of pull requests till deployment. At the end, I have wrote an individual milestone report for the project explaining my code, tests and mindset elaborately. I have followed instructions from the pdf that professor gave. I have wrote the project plan part of our group report. I then submitted my individual milestone report over moodle.		
	Milestone 2	Issue#254 Issue#226 Issue#174 Issue#185 Issue#264	PR#179 PR#184 PR#198 PR#217 PR#234 PR#247

4. Evaluation of tools and processes

First of all, we gathered as a team and talked about all frameworks. We looked at which ones would be appropriate and what we could do on them. We didn't have many choices anyway, because we didn't have anyone close to any framework. For this reason, we decided to do it on Django, which is shown in the CMPE352 tutorial. We did some good research for Django. https://www.djangoproject.com this site covers everything for django. Anyone who doesn't understand any django can start the process with just these docs. We didn't choose django just because it is covered in this lesson, it provides such a nice ease of use that it almost takes care of everything itself. It tells you what to do as necessary html files and folders as well as a comment line. Django was a great choice. Of course, we had to choose python as the programming language. The beauty of this language was that it was a language that we all had an opinion about. In addition, the issue we need to specify for python is that it offers us a suitable library option in every field. For python, https://www.python.org can be used as a source.Thanks to these two choices, we were at the same point as a team, and we could look at the project from the same perspective.

We needed an ide after selecting them. In this search process, of course, an idea was needed that would be compatible with github and would provide us with terminal, python facilities, debug features. While we were having this conversation, we thought about VSCode, which we use all the time. The whole group was using it and was able to work in concert with git. After all the discussion we decided to choose VSCode as the "ide".



We used our usual applications for communication. Since this project was a little more intense, we had to talk in a hurry most of the time. For this reason, we used WhatsApp extensively. Whatsapp is like a problem solver here, you can get instant answers to all your problems. It provides facilities for sending photos, screenshots, sound recordings via Whatsapp. We held our general group meetings, planned or unplanned, via zoom. Zoom offers us features such as screen sharing and seeing each other. Since we are not a premium account, we use it for 40 minutes. When our minute is up, we hurt all over again. In addition, we have a discord channel. Group members who want to work together can work actively through that channel.

When looking for a tool for documentation, the first requirement that comes to mind is that it can be online. In such cases, the biggest problem is conflicts that occur during the update. For this reason, it is necessary to work through an online tool. In this search, we realized that most of us use Google's product as drive. In this drive, Google offered us a documentation application. We can even view the history of the file on this application. We can all work on a file at the same time. For this reason, we preferred Google's application called docs.

Finally, we used Project Libre to prepare the project plan. We didn't have much trouble choosing this application because we used the same one in Milestone1 and were extremely satisfied. Therefore, we decided to use this application again.Projectlibre definitely helped us prepare the project plan in a short amount of time and a nice looking project plan outputs.
So, overall, we were able to use our tools easily and we benefited from it to make our practice.app in a short amount of time while satisfying our project requirements.

5. Deliverables

5.1 Glossary & Requirements for Practice Online Learning Platform

GLOSSARY

Online Learning Platform: An online platform that enables student to learn by their own without face-to-face communication.

Teacher: Anyone who knows a topic well and willing to teach it to other people.

Learner: Anyone who is interested in learning.

e-learning: Learning from digital platforms instead of on-site education**.**

User Profile: A virtual entity for the real user, which includes collection of information about a user describing her interests and knowledge.

Recommendations: Suggestions that are offered to the user considering their personal traits, previous usage, likes etc.

Learning Environment: An environment containing all the learning related things or activities. This includes teacher, learner, learning platform, quizzes, assignments and learning place…

Reputation: A score for teachers gathered by the feedbacks of learners that evaluates the teaching performance.

Learning Experience: All the things related to learner's individual learning process. That includes their progress, feedbacks, evaluations etc.

Course Statistics: The statistics related to a course: i.e. its rating, number of attendees etc.

RESTful API: An architectural style for distributed hypermedia systems.

Preferences: The learning preferences of a student consisting of the topics the student wants to learn and the level that the student wants to learn.

Requirements

1. Functional Requirements

1.1 User Requirements

1.1.1 Authentication

1.1.1.1 Sign up

- 1.1.1.1.1 Users shall sign up to enroll the courses on the online learning platform by specifying their username and password in the sign-up process.
- 1.1.1.1.2 Guest user's username should not be taken by another account beforehand.
- 1.1.1.1.3 When guest user signs up, she is redirected to sign-in page.

1.1.1.2 Username, password

- 1.1.1.2.1 A user shall select a unique username and a password containing at least 8 letters.

1.1.1.3 Sign in

- 1.1.1.4.1 A user shall enter her password correctly to log in.

1.1.2 Profile

1.1.2.1 Profile Specifications

- 1.1.2.1.1 Every user shall have a profile page or customized main index page.
- 1.1.2.1.2 A student user shall be able to specify their interests-preferences.

- 1.1.2.1.5 A user shall be able to see her courses (attended or offered).

1.1.3 Guest

- 1.1.3.1 Guest users shall be able to sign-up
- 1.1.3.2 Guest users shall be able to sign-in

1.1.4 Student

- 1.1.4.1 A user who is registered as a student during the sign up process is considered as a student.
- 1.1.4.2 Student shall be able to enroll a course.
- 1.1.4.3 Student shall be able to give rate to courses.
- 1.1.4.4 Students shall be able to see their courses.
- 1.1.4.5 Student shall be able to specify their preferences by providing topic and the degree of level they want to learn.

1.1.5 Lecturer

- 1.1.5.1 A user who is signed up as a teacher shall be considered as a lecturer.
- 1.1.5.2 Lecturer shall be able to add a course.
- 1.1.5.3 Lecturer shall be able to see their course statistics.
- 1.1.5.4 Lecturer shall be able to delete a course.
- 1.1.5.5 Lecturer shall be able to see their courses.

1.2 System Requirements

1.2.1 Pages

1.2.2.1 Main Page

- 1.2.2.1.1 Main page shall include navigations to the the required functionalities of the user.
- 1.2.2.1.2 System shall be able to allow users to log-out from their main pages.

1.2.2.2 Student Pages

- 1.2.2.3.1 Student enroll page shall be able to allow student users to enroll a course that they specify.
- 1.2.2.3.2 Student Give Rate page shall be able to allow users to give rate a course that they specify.
- 1.2.2.3.3 Student My Courses page shall be able to show the courses that the student is enrolled .
- 1.2.2.3.4 Student Specify Preferences page shall be able to show the student preferences and allow the student to add more.

1.2.2.3 Teacher Pages

- 1.2.2.4.1 Teacher Add Course page shall be able to allow teacher users to add a course that they specify.
- 1.2.2.4.2 Teacher Course Statistics page shall be able to allow teacher users to check the statistics of a course they specify.
- 1.2.2.4.3 Teacher Delete Course page shall be able to allow teacher users to delete a course that they specify.
- 1.2.2.4.4 Teacher My Courses page shall be able to allow teacher users to see their courses.

1.2.2 Recommendations

- 1.2.3.1 Students should be able to see the recommended courses.

- 1.2.3.2 The recommeded courses will be decided based on the preferences the students specified.

2. Non-functional Requirements

2.1 Privacy

- 2.1.1 The platform should comply with all rules regarding KVKK and GDPR.

2.2 Security

- 2.2.1 URL endpoints shouldn't be accessible to the guest user.
- 2.2.2 The system should encrypt passwords with SHA-256 algorithm using a randomly generated salt. Passwords hashes and the respective salt shall be stored in the database.
- 2.2.3 Users cannot be able to perform actions without priveleges not defined for them. For example users should not access to data of

2.3 Accessability

- 2.3.1 The platform should be accessed via web interface which should be responsive and support Chrome, Safari and Mozilla.
- 2.3.2 The platform language should be English and support UTF-8 character encoding.

2.4 Performance & Realiability

- 2.4.1 At least 150 registered users with their actions should be satisfied simultaneosly.
- 2.4.2 At least 1500 guest users should be able to inspect the system.

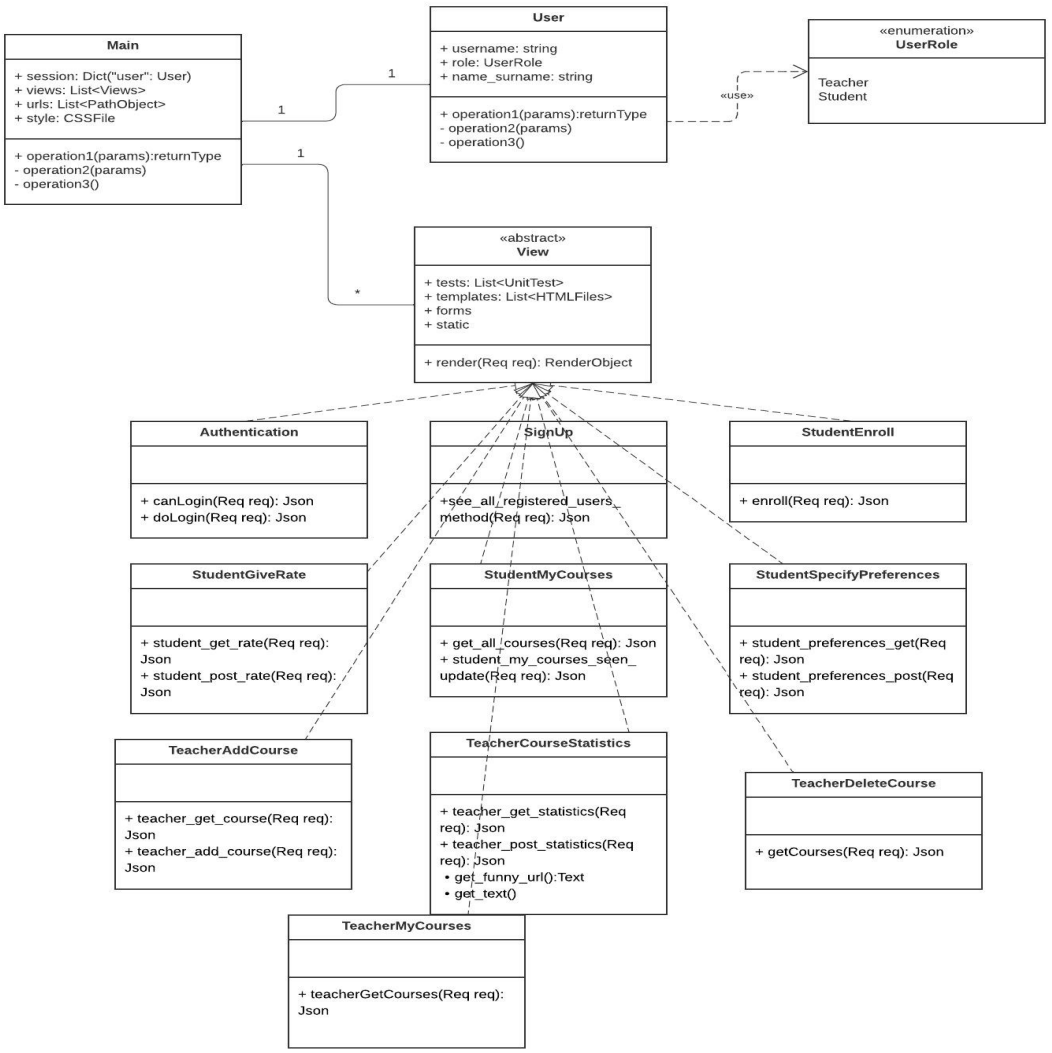
- 2.4.3 A user should have a response in at most 3 seconds excluding the delay based on the machine that the user uses.

2.5 Standards

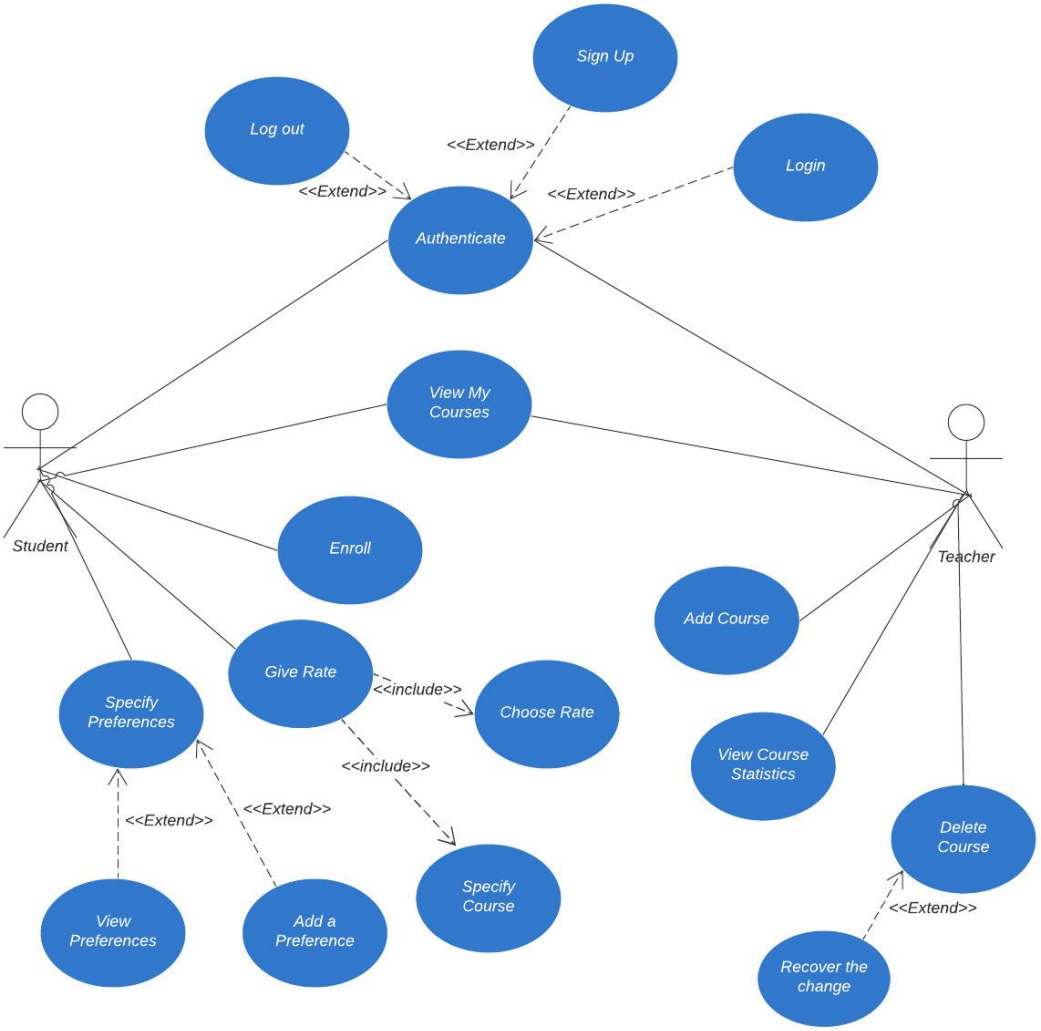
- 2.5.1 The project should satisfy the rules of W3C Standards.
- 2.5.2 The semantic should be based on wikidata.org.

5.2 UML Design

5.2.1 Class Diagram



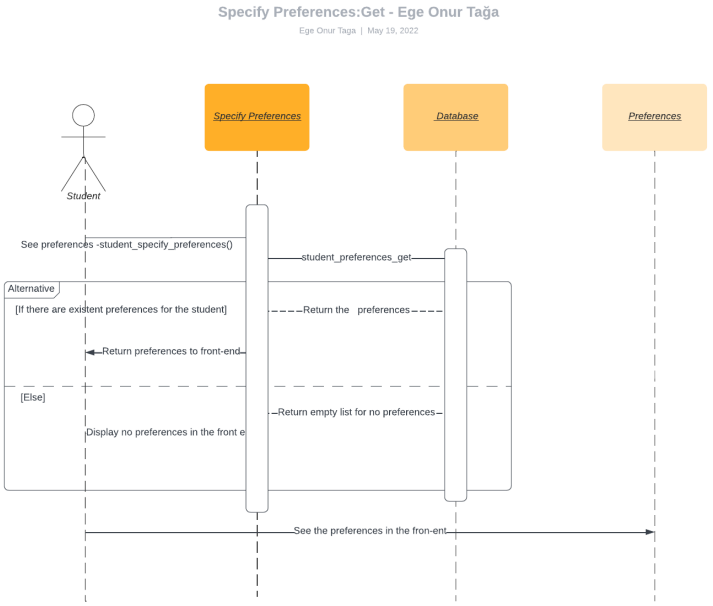
5.2.2 Use Case Diagram



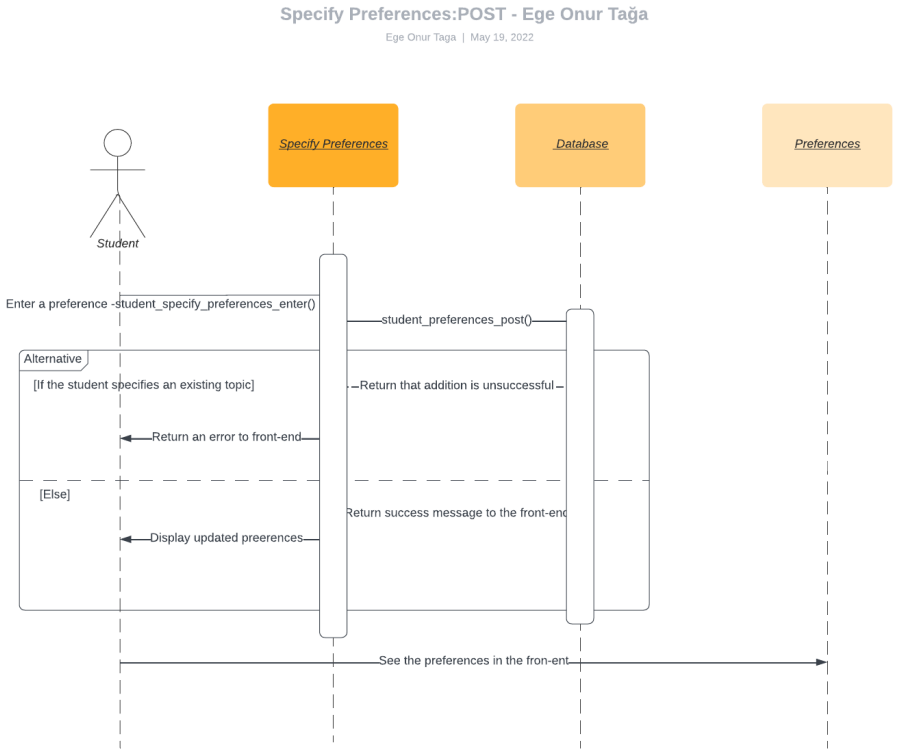
5.2.3 Sequence Diagrams

Wiki Link: Sequence Diagrams

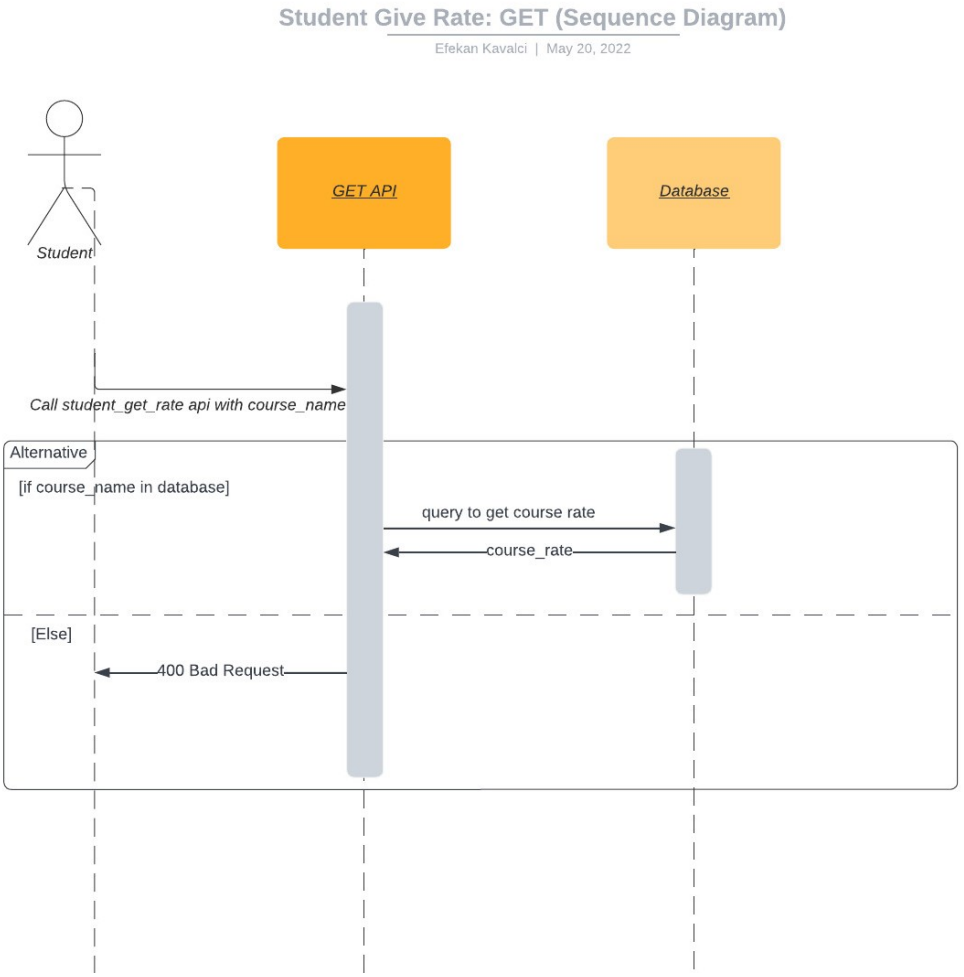
- **Specify Preferences:GET (Ege Onur Tağa)**



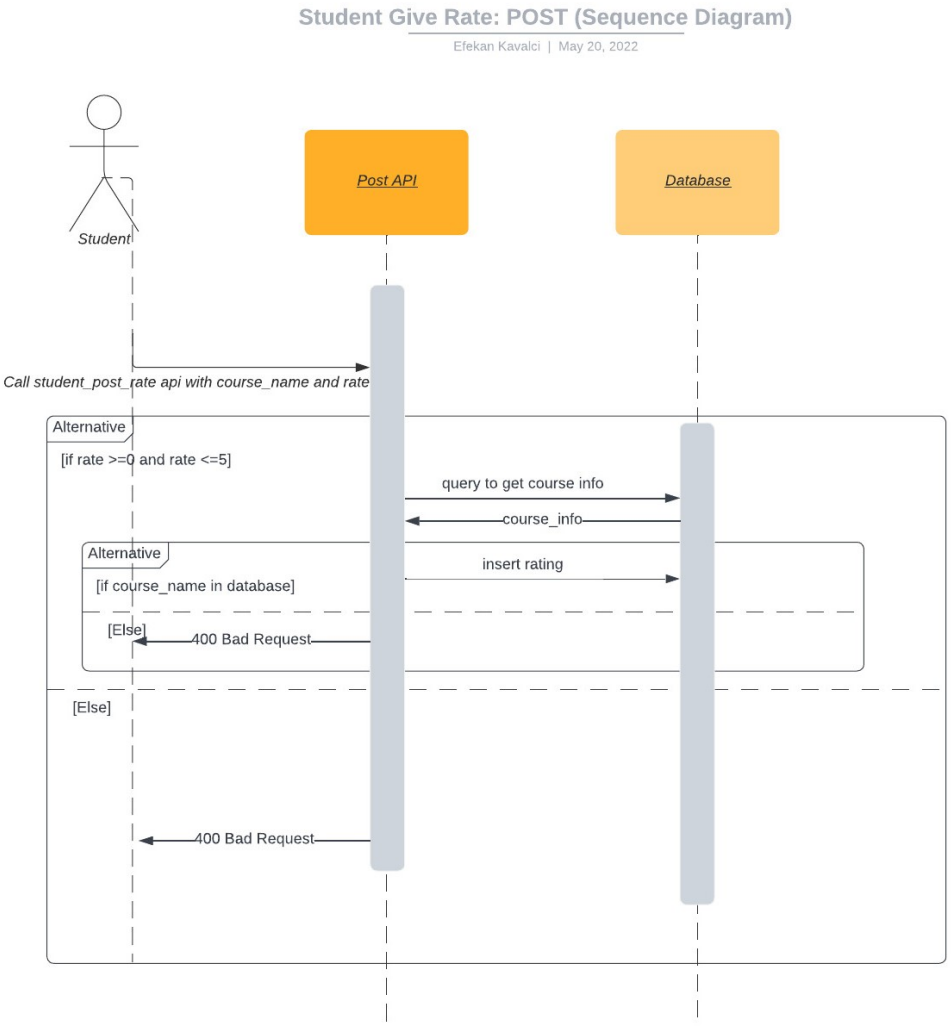
- **Specify Preferences:POST (Ege Onur Tağa)**



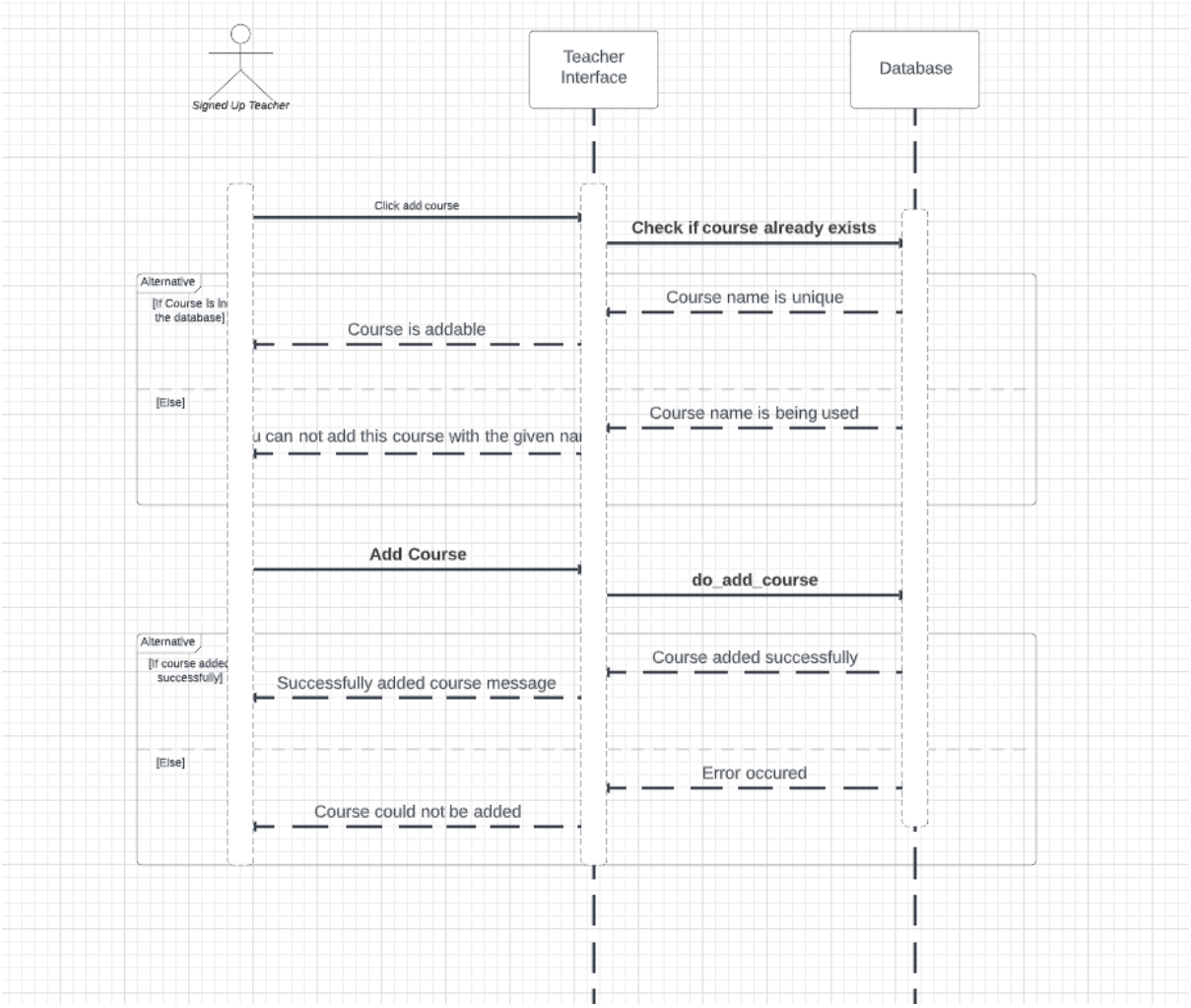
- **Give Rate:GET (Efekan Kavalcı)**



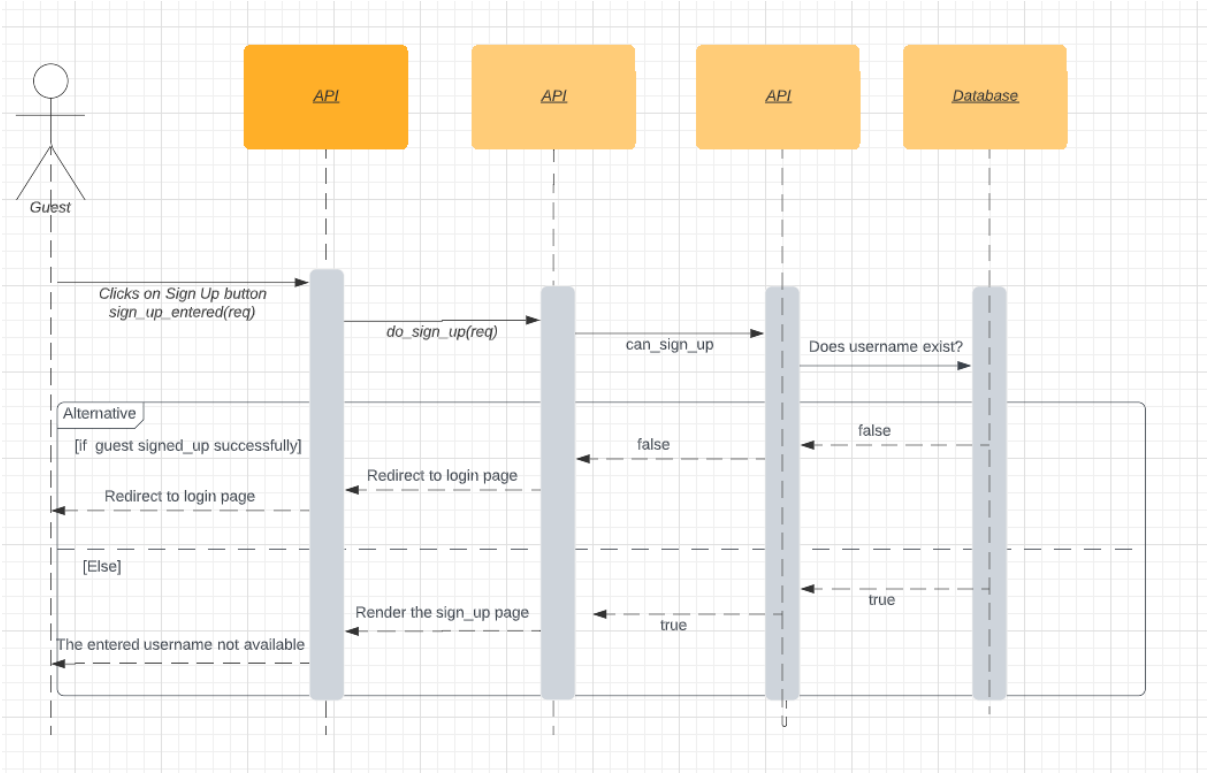
- **Give Rate:POST (Efekan Kavalcı)**


Student Give Rate: POST (Sequence Diagram)
Efekan Kavalci | May 20, 2022
Student
Post API
Database
Call student_post_rate api with course_name and rate
Alternative
[if rate >=0 and rate <=5]
query to get course info
course_info
Alternative
[if course_name in database]
insert rating
[Else]
400 Bad Request
[Else]
400 Bad Request

- **Add Course: POST (Ahmet YAZICI)**



Ömer Özdemir - Sign UP



5.3 URI of the Tag of the Project

TAG

<tag>

<name>

Group 1 Practice App Deliverable

</name>

<description>

A project that serves students and teachers in education through APIs

</description>

</tag>

5.4 Code and Brief Guide

Code is in the zip file and also in the

````
## Requirements (COPY)

- MySQL
- Python(>3.8) and pip module.

If you have these, then run the following code:
`pip install -r requirements.txt`
In order to prevent any possible conflicts, you can set up a virtual environment. You can learn more about virtual environments on [here](https://docs.python.org/3/library/venv.html#module-venv)

## Deployment

First, Create a database on a sql ide (mysql) with <YOUR_DB_NAME>

Second, create an .env file in "practiceapp" folder (folder with the settings.py file), and insert:

```
MYSQL_DATABASE=<YOUR_DB_NAME>
MYSQL_USER=<YOUR_USERNAME>
MYSQL_ROOT_PASSWORD=<YOUR_PASSWORD>
MYSQL_PASSWORD=<YOUR_PASSWORD>
MYSQL_HOST="localhost"
teacher_course_statistics_api="e5e0aa9024mshdbd00e1fc1a3f31p14545ejsn9af52ee101e7"
RandomUsername_API_KEY=768d5c5ca7msh0de1bf3500804cbp1b12f3jsn930da0c69205
API_KEY_give_rate='UXzo3fo9cg3c3/V4CF17Jw==o9S0YYHLBairenXf'
```

After that, ensure that your database server is up and run these commands to set up the database to Django configurations:

```
python manage.py makemigrations
python manage.py migrate
```
````

```
    This will create some Django related tables on the database (Do
not alter them otherwise framework may fail).

    Then, you can run this command to create up and fill relevant
tables (Note: execute this command in the root folder of project):

    ```
    python practiceapp/create_db.py
    ```

    Finally, run the command:
    `python manage.py runserver`
    and check whether the website is accessible at:
http://127.0.0.1:8000/learningPlatform/

    You can test the system with the command
    `python manage.py tests/*`
```

With docker:

1. First, download and install Docker, which is suitable for your operating system, from https://docs.docker.com/get-docker/ .
2. Download and install Mysql from https://dev.mysql.com/downloads/mysql/ .
3. Clone the code from github to your computer and checkout Master Branch.
4. Then go to the directory where the docker-compose.yml file is located from the terminal.
5. Create Mysql username and table and add this information to the relevant places in the docker-compose.yml file.
6. Run following commands in order:

docker-compose build

docker-compose up

7. Currently docker is running but there are a few more things to do. Open a different terminal and navigate to your project's directory.
8. Run these commands:

docker container ls -> This command Shows the list of containers. There you will see the name of the docker you just ran.

docker exec -it <name_of_the_docker> /bin/bash

9. You are now inside the docker image. G oto practiceapp/practiceapp/ folder and run -> python create_db.py
10. Go up 1 directory -> cd ..
11. Run -> python manage.py makemigrations
    è python manage.py migrate
12. Tables have been created in your database and your application is running in your local environment.