Bogazici University Fundamentals of Software Engineering

CMPE 352

Spring 2022 - Group 5

Milestone Report 2

Contributors:

- Alper Canberk Balcı
- Baver Bengin Beştaş
- Burak Mert
- Buse Tolunay
- Engin Oğuzhan Şenol
- Halil Burak Pala
- Kardelen Demiral
- Mehmet Akif Yılmaz
- Mehmet Emre Akbulut
- Oğuzhan Demirel
- Sinan Kerem Gündüz
- Yavuz Samet Topçuoğlu

TABLE OF CONTENTS

Executive Summary

Summary of Project and Overall Status

Introduction

We are a group of 12 computer engineering students from Boğaziçi University who are taking the course “Fundamentals of Software Engineering” in the 2022 spring term. We are creating a demo known as a practice app for the medical experience sharing platform we will develop next semester.

This is the second milestone of our project. It includes our progress so far, our future plans and the deliverables.

Description

Our platform provides its users an environment to share their medical experiences with each other, being informed about medical topics and also gives them the opportunity to receive responses from verified doctors. Anyone can use Android and the web versions of our application without being charged.

As a demo of the medical experience sharing platform explained above, we implemented essential APIs of the application. This demo is called as “Practice App”.
Practice App is developed by using Python, Django and Django Rest Framework. Also SQLite is used for the project. Every member developed API and unit tests on their branches as documented in the Meeting Notes 10.2.

Essential parts of the Practice App can be summarized as User, Article, Post and Comment. A registered user can use the APIs in the home page after logging in the system. Also a guest user can sign up by using the “Create User” API and view all users from clicking the sign up button in the login page.

All the APIs are listed in the homepage.We decided not to use the endpoints we decided in our previous meeting and created a new structure for our practice-app.

Endpoints and functionalities:

	URL	GET	POST
1	/post/api	Get all posts	Create a post
2	/post-info/api/<post-id>	Get post with id post-id	Update the post with id post-id
3	/category/api	Get all categories	Create a category
4	/category-info/api/ <category-id>	Get posts with category	Update category
5	/comment/api/<post-id>	Get all comments of a post	Create a comment under the post
6	/comment-info/api/<comment-id>	Get comment with id comment-id	Update comment with id comment-id
7	/article/api	Get all articles	Create an article
8	/article-info/api/<article-id>	Get article with id article-id	Update article with id article-id
9	/rate-post/api/<post-id>	Get number of upvotes and downvotes of a post with id post-id	Upvote or downvote a post with id post-id
10	/rate-comment/api/ <comment-id>	Get number of upvotes and downvotes of a comment with id comment-id	Upvote or downvote comment with id comment-id

11	/rate-article/api/<article-id>	Get upvote and downvote numbers of an article with id article-id	Upvote or downvote article with id article-id
12	/user/api	Get all users	Create a new user

URL of the Deployed Application

http://3.69.144.13:8888/

API URLs

- **/ :** Root URL of the application. Front-end implementation of login is shown if the user is not logged in, the home page of the application is shown if the user is logged in.
- **/article :**
  - **/article/api:** RESTful API handling:
    - getting all the articles in the database via GET method
    - posting an article to the database via POST method.
  - **/article/:** Frontend page that serves the purpose of a bridge between main parts of the endpoint and the home page. User can go to GET or POST pages from here.
  - **/article/get:** Frontend page which lists all the articles in the database.
  - **/article/create:** Frontend page which shows the newly created article data.
- **/article-info :** Frontend implementation of the article-info API.
  - **/article-info/api/<article-id> :** RESTful API for getting the article with given id with GET and updating an article with given id with POST methods.
  - **/article-info/get :** A page showing the requested article.
  - **/article-info/update :** A page for updating an article with a given id.
- **/category:** Frontend implementation of the category API.
  - **/category/api/ :** RESTful API which handles fetching all categories in a GET method, and creating a new category in a POST method.
  - **/category/:** Frontend page which displays operations related category endpoint.
  - **/category/list:** Frontend page which displays all categories.

    - **/category/create:** Frontend page includes a form to create a new category.
- **/category-info**
- **/comment :** Frontend implementation of the comment API.
    - **/comment/api/<post-id> :** RESTful API which handles fetching all comments of a specific post in a GET method, and creating new comment for a post in a POST method.
    - **/comment/list :** Frontend page which displays all the comments of a post.
    - **/comment/create :** Frontend page which is displayed after a successful creation of a comment.
- **/comment-info :** Frontend implementation of the comment-info API.
    - **/comment-info/api/<post-id>:** RESTful page for the comment-info API.
        - Gets comment by id.
        - Updates comment.
    - **/comment-info/SearchResult:** The page showing the requested comment.
    - **/comment-info/UpdatedResult:** The page showing the updated comment.
- **/post :** Frontend implementation of the post API.
    - **/post/api :** RESTful page for the post API.
        - List all posts with the GET method.
        - Create a new post with the POST method.
    - **/post/get :** A page that contains data for all the posts.
    - **/post/create :** A page contains data of the created post.
- **/post-info :** Frontend implementation of the post-info API.
    - **/post-info/api/<post-id>:** RESTful API used for:
        - getting a post providing a post id with GET.
        - updating a post with POST.
    - **/post-info/get:** The page showing the requested post.
    - **/post-info/update:** The update post page.
- **/rate-article :** Frontend implementation for rate-article API.
    - **/rate-article/api/<article-id> :** RESTful API for rate-article API.
    - **/rate-comment/get :** Page that shows vote counts for an article.
    - **/rate-comment/post :** Page that the user can vote on an article.
- **/rate-comment :** Frontend implementation for rate-comment API.
    - **/rate-comment/api/<cmrateid> :** RESTful API for rate-comment API.
    - **/rate-comment/get :** Page that shows vote counts for a comment
    - **/rate-comment/post :** Page that can be voted for a comment
- **/rate-post :** Frontend implementation for rate-post API.
    - **/rate-post/api/<post_id> :** RESTful API for rate-post API.

    - **/rate-post/get :** Page that shows vote counts for a post
    - **/rate-post/post :** Page that can be voted for a post
- **/user :** Frontend implementation of the post API.
    - **/user/**: Main page in which user can APIs fetching all users and creating new user.
    - **/user/api**: RESTful API which handles user functionalities.
        - GET: Returns all users registered to system
        - POST: Creates a new user with username,email and password
    - **/user/api/<username>**: RESTful API which returns details of a specific user.
        - GET: Returns details of a specific user.
    - **/user/allUsers**: Page that shows all users by using GET request to /user/api
    - **/user/createUser**: Django view to handle “Create User” form from html, send POST request to /user/api endpoint and redirect to main page with error message or success message

Basic Functionality of the Project

This project is a demo application for our actual Medical Experience sharing platform project. This practice application can be described as a small forum consisting of users, categories, posts of these categories, comments of these posts and articles of categories. Every post, comment and article has upvote and downvote numbers.

A user can register to the system, can create categories, posts, comments and articles. S/he can also rate a post, comment or an article by giving upvote or downvote. S/he can also add new users to the system. S/he can view all categories, all posts, a specific post, all comments of a post, a specific comment and all users. S/he can update a category, a post, a comment or an article.

Categories have a name and a definition of this name. Definition of a category is fetched from an external API if there exists a definition for that word.

Posts have a title, a body, a category, a country info, Covid-19 cases information of the country in the time of the creation of the post. Covid-19 cases information is fetched from an external API.

A comment of a post has a body, a city name info, weather information of that city at the time of the creation of the comment. Weather information is fetched from an external API.

Articles are similar to Posts except they don’t have covid-19 info and there cannot be comments for an article.

Some Challenges We Met

Since we were used to working on individual coding projects, it was quite difficult to understand each other's code and follow what's going on with the project continuously. However, having learned how to use git for team projects, the problem became less overwhelming.

Sometimes external APIs we used in the project can crash or don't work properly. To overcome this situation, we researched for more stable API's and used them.

Web development was also something most of us were not very familiar with. Therefore, we spent a considerable amount of time on writing HTML and/or CSS. This situation slowed down our progress.

List of Status Of Deliverables

Deliverable	**Status**	**Last Update Date**
Milestone Report	Completed	20.05.2022
Requirements	Completed	18.05.2022
Use Case Diagram	Completed	18.05.2022
Class Diagram	Completed	18.05.2022
Sequence Diagrams	Completed	20.05.2022
Project Plan & RAM	Completed	19.05.2022
Practice Application	Completed	20.05.2022

Project Plan

Project Plan, RAM

Summary of Work Performed by Each Member

Alper Canberk Balcı	• Conducted research about HTTP Protocols, what an API is, RESTful API's, unit testing, git and git commands, github pull requests, how to use them. • Learned Django Django REST framework, some HTML, which

	we use in practice-app implementation. • Attended Meeting 10.1, took the meeting notes, and discussed our practice-app version with the group members. • Attended Meeting 10.2 and decided who will implement which API's of the practice app. • Added Windows commands for Windows users to README.md file issue #79 • Opened my issue #105 for endpoint article/. Created a branch from master, according to the Meeting 10.2 notes. Implemented the initial structure of API for article/ endpoint, but that branch is closed and opened later, work done in that branch is copied. • Reviewed and merged the pull request: #123. • Reviewed pull request #107. It brought some issues, and the source of the problem was found. The PR canceled and was opened here PR #125. • Reviewed the issues: #102, #137, #97, #108, #144 by merging Pull Requests: #136, #139, #125, #132, #146. • Made some changes in the implementation. Make commits finally. Issue #105. • Opened a Pull Request of that commit: #129 for issue #105. • Tested my API in Postman. Prepared and published the Documentation under the issue #105. • Opened issue #140. Extension to #105. Implemented Unit tests, front-end, and some enhancements in the API. • Opened a Pull Request #154 for issue #140, mentioned it in the comments under. • Attended Meeting 12.1 helped to take meeting notes. Contributed to the requirements, use-case and class diagrams for the practice-app, finished them together with the group members. Contributed to the Milestone 2 report, project plan, and RAM finished them. • Meeting 12.1 action item 4: Opened issue #158. Checked the correctness of the files and uploaded the Project Plan and RAM to the Wiki. • Prepared "Create Article" sequence diagram.
Baver Bengin Beştaş	• Researched about Git, Git commands, Github, Github commands. • Researched about API, RESTful API's and unit testing. • Researched the Django REST framework for practice-app. • Attended Meeting 10.1, we discussed our roadplan and the things that need to be done for the practice-app. • Attended Meeting 10.2, we did work sharing for practice-app and discussed what to do with even further details. • At Meeting 10.2, I was assigned with /rate-article endpoint.

	Opened the issue #100 for implementing the /rate-article endpoint for our practice-app, created a branch named rate-article. ● Got assigned as a reviewer for the issues: #105, #109, #134, #140, #156. I reviewed them as best as I could. ● Got assigned to the pull requests #129 and #154. Approved the pull requests. Merged them after other reviewers reviewed them. ● Made implementation of /rate-article endpoint. After necessary changes and implementing unit tests for /rate-article endpoint, I created a pull request #151 and assigned co-workers merged the pull request. ● Attended Meeting 12.1 and we worked on use-case diagrams and class-diagrams. We finalized them in the meeting. Also created milestone-2 report. ● Prepared Rate-Article sequence diagram. ● Approved pull request #167 about final bug fixes and improvements.
Burak Mert	● Made research about Git commands and Github pull requests. ● Made research about web development, how APIs and HTTP work. ● Read the documentation of Django and REST framework. ● Made some practice for Django and REST framework before starting the practice application. ● Attended Meeting 10.1 to discuss APIs and initial structure of the practice app. ● Implemented the initial structure of the practice app and pushed to the github during our meeting. ● Attended Meeting 10.2 to make last decisions about our practice-app and create models of the application. ● Opening issue #88 about implementing the /category endpoint of the practice application. ● Implementing the initial structure of the application and pushing to the branch. ● Prepared a Postman documentation for /category endpoint. ● Implementing initial structure of the category endpoint and frontend for it. You can see the related branch, and the pull request for this branch. ● Detecting a bug about misdirecting urls in the application, opening an issue, #113, for the bug, creating a new branch and a pull request for a corrected version of the application. ● Opening a new issue, #117, for implementing unit tests of the application. ● Implementing unit tests for the category endpoint of the

	application. You can see the related branch and pull request. ● Reviewing these pull requests: #120, #130, #131. ● Made research about Docker and Aws EC2. ● Attended Meeting 12.1. Working on use-case diagrams and class diagrams collectively. ● Prepared individual Milestone 2 Report. ● Created a sequence diagram related /category endpoint.
Buse Tolunay	
Engin Oğuzhan Şenol	● Researched how to use GitHub, using pull requests, commits, pushes, and other necessary features. ● Researched how to use Django and REST framework. ● Attended Meeting 10.1, discussed how to implement APIs and the initial structure of the practice app. ● Attended Meeting 10.2, decided what we are going to do for the practice app. Divided the tasks for each member, which mine is rate-comment. ● Opened issue #109 for implementing rate-comment endpoints. ● Created a new branch named practice-app/feature/rate-comment#109 for the relevant issue. ● Committed and pushed what is implemented for rate comment, and created a pull request for the relevant issue. ● Opened issue #134 for unit tests of implementing rate-comment endpoints. ● Created a new branch named practice-app/testing/rate-comment#134 for the relevant issue. ● Committed and pushed the unit tests for rate comment, and created a pull request for the relevant issue. ● Opened issue #156 for bugs in rate-comment endpoints. ● Created a new branch named practice-app/bug-fix/rate-comment#156 for the relevant issue. ● Committed and pushed corrected rate comment endpoints, and created a pull request for the relevant issue. ● Reviewed and merged the pull request: #153 ● Attended Meeting 12.1. Contributed to the use-case and class diagrams for the practice app. Contributed to the Milestone 2 report, project plan, and RAM. ● Attended Meeting 12.2. Dockerized and deployed our practice app. ● Opened issue #161. Uploaded sequence diagrams to our wiki page. ● Prepared individual Milestone 2 Report. ● Prepared Rate Comment sequence diagram. ● Prepared the documentation of the rate-comment API using

	Postman.
Halil Burak Pala	• Studied git and git commands. • Studied web development basics. Learned what API and HTTP protocol are. • Studied Django and its REST framework before implementing our practice app. • Attended Meeting 10.1 in which we brainstormed to design an API for our practice app. • Thought about the possible implementation schemas for our practice-app. Did some pre research for our Meeting 10.2, tried out Django to create an initial structure of our practice app. • Attended Meeting 10.2 in which we made last decisions about our practice-app, we implemented our models. • Took notes of Meeting 10.2. Created issue #78 about the notes of the meeting, created a new page for the meeting and uploaded my notes. • During Meeting 10.2, created the very first pull request (#77) of our repo to demonstrate to other group members how the pull request and branching mechanism of Github works. • Created a new branch for the initial structure of our practice-app we created, created a pull request to merge it (#80). • Implemented the initial API structure of comment/api endpoint of our practice-app. Relevant issue: #86 , Relevant PR: #91 • Made research about how to hide external API keys. Learned how to create a .env file. • Fixed a bug related to showing the external API key in the comment/api endpoint. Hid the API key in an environment variable in a .env file. Relevant issue: #94 , Relevant PR: #95 • Created the initial version of the frontend of comment/api endpoint. Relevant issue: #112, Relevant PR: #120 • Implemented some improvements which include showing error messages in different error cases in the frontend of the comment/ endpoint. Relevant issue: #112, Relevant PR: #130 • Created unit tests for the comment/ endpoint. Wrote unit tests for testing views, models, forms and URLs. Relevant issue: #122, Relevant PR: #131 • Removed unnecessary packages from requirements.txt. Relevant commit : here. • Fixed a bug caused by importing an unnecessary package in the project, removed unnecessary files and migrations. Relevant issue: #152, Relevant PR: #153 • Did a research about Docker. Learned how to create an image from an application and how to dockerize this image. Made some demos before our actual dockerization.

	• Did a research about Amazon AWS Web services. Learned how to create AWS EC2 instances, learned how to deploy our application. • Attended Meeting 12.1. Contributed to the completion of our use case and class diagrams and our project plan for this practice app. Contributed to the Milestone 2 report. • Prepared sequence diagram for creating a comment. • Wrote my individual work to our group's Milestone 2 report. • Wrote my own individual Milestone report. • Prepared the documentation for comment API: Practice App Comment API • Attended Meeting 12.2 in which we dockerized and deployed our application. Took the meeting notes. • Created a guide for running our app using docker and put it in our readme file of the practice app. Relevant issue: #162 • Reviewed issues: #84, #111, #142, #143, #145, • Reviewed and merged pull requests: #90, #110, #124, #141, #150, #155
Kardelen Demiral	• Made a research about API's to get the idea of why and how to use and implement them. • Learned about Django and REST framework to use it in practice-app. • Attended Meeting 10.1, discussed how to implement our practice-app with the group members. • Attended Meeting 10.2, made our final decision about the structure of practice-app and subdivision of the tasks. • Opened the issue #85 which is about implementing the API for post-info, self-assigned it. • Implemented the initial structure of the API for post-info and UI of it. See the relevant branch. Made a pull request after finishing it. • Opened the issue #115 which is about implementing the unit tests for post-info, self-assigned it. • Implemented the unit tests for post-info. See the relevant branch. Made a pull request after finishing it. • Reviewed and merged the pull requests: #82, #89, #103, #118, #159 • Opened the issue #135 which is about some enhancements in the post-info API, self-assigned it. • Made some modifications in API for post-info and UI of it as the issue #135 says. • Changed the external API used in post-info API as the issue #137 says. • Made a pull request containing some enhancements explained above. Pull request: #138.

	• Attended Meeting 12.1. Contributed to the use-case and class diagrams for the practice-app, finished them together with the group members. Contributed to the Milestone 2 report and project plan, finished them. • Prepared my individual Milestone 2 Report. • Prepared “Update Post” sequence diagram. • Opened the issue #149 which is for the preparation of the documentation of post-info API, self-assigned it. • Prepared the documentation of the post-info API. See: post-info API documentation. • Opened the issue #163 and made the corresponding pull request #164. Fixed a bug.
Mehmet Akif Yılmaz	• Done research about API’s and what they are used for. • Done research about Django and rest framework and how to use them. • Done research about git, git commands and how to use it with Github for version management. • Attended Meeting 10.1 and discussed about the structure of the practice app. • Attended Meeting 10.2 and decided on the API’s of the practice app and divided it among group members. • Implemented the /article-info API and its User Interface on a newly created branch as mentioned in issue #97 and pull requests #107 and #125. See related branch. • Prepared a documentation for the API including sample calls using Postman. • Implemented Unit tests for the article info API by opening a new branch as mentioned in issue #108 and pull request(#132). See related branch. • Fixed some bugs related to the article info API as mentioned in issue #144 and pull request #146. See related branch. • Reviewed and merged these pull requests: #119, #129, #154 • Reviewed issues : #105, #117, #140 • Attended Meeting 12.1 where we worked on Use case and Class diagram, requirements, project plan and Milestone 2 report. • Uploaded the Use Case diagram to the wiki page. • Prepared “Update Article” sequence diagram. • Prepared my individual Milestone 2 report. • Done research about Docker and AWS. • Attended Meeting 12.2 where we dockerized and deployed our main application.
Mehmet Emre Akbulut	• Read and watched some tutorials about what is an API and why we use them.

	• Learned Django and Django Rest Framework and did some practice APIs by using them. • Studied Django built-in User model and read its documentation for authentication. • Attended Meeting 10.1, discussed how to implement our practice-app with the group members. • Attended Meeting 10.2, made our final decision about the structure of practice-app and subdivision of the tasks. • Opened the issue #81 which is about implementing the API for user, self-assigned it. • Initial structure of API is implemented. Fetching all users and creating new user functionalities are added. • Endpoints are tested via Postman. • Postman documenter is used to create a document for API. • Also learn environment variables with ".env" to store the API Key. • See the relevant branch. Made a pull request after finishing it. • Opened the issue #87 which is about fixing some bugs and implementing some UI for API, self-assigned it. • Fixed bugs. See the relevant branch. • Implemented the UI for the API. See the relevant branch. Made a pull request after finishing fixing bugs and another pull request for UI implementation. • Reviewed and merged the pull requests: #77, #80, #92, #99, #106, #128, #139, #151 • Opened the issue #104 which is about the Unit Tests about User API, self-assigned it. • See the branch. • Made a pull request containing some enhancements explained above. Pull request: #118. • Attended Meeting 12.1. Contributed to the use-case and class diagrams for the practice-app, finished them together with the group members. Contributed to the Milestone 2 report and project plan, finished them. • Prepared my individual Milestone 2 Report. • Prepared "Create User" sequence diagram. • Opened the issue #121 and issue #148 which is UI structure of the practice app login page and home page, self-assigned it. • Research some codes on the internet to use for Login and Home Page of practice app. Listed all APIs in a card view. • Authentication and signup is implemented for the practice app. • Initial branch and enhancement branch for this issue. • Pull request for initial brach and pull request for enhancement branch is created. • Prepared the documentation of the post-info API. See: User API documentation.

	• Learn Docker and deployment with AWS EC2 instance.
Oğuzhan Demirel	• I did some study on APIs and how they are used. • I did some study on Django and the REST framework, as well as how to use them. • I did some research on git, git commands, and how to use Github for version control. • Attended Meeting 10.1, where we discussed how to put our practice app into action. • Attended Meeting 10.2, where the APIs for the practice app were decided and distributed among group members. • As described in issue #96 the /rate-post API and accompanying User Interface were implemented on a newly established branch. • The GET and POST components of the rate-post endpoint have been implemented. I documented them on the postman with examples of how to use them. • Unit tests for the rate-post API were implemented • We worked on the Use case and Class diagram, requirements, project plan, and Milestone 2 report at Meeting 12.1. • Prepared a sequence diagram for "Rate Post." • With pull request #128, we merged my improvements to the master. • I completed my Milestone 2 report. • I did some study on Docker and AWS systems.
Sinan Kerem Gündüz	• Researched about API's, RESTful API, Django and some external API's that we can use in our app. • Attended Meeting 10.1, we have discussed about how to implement our practice app. • Attended Meeting 10.2. In this meeting, we have finalized our discussions and divided the labor of the endpoints among 12 group members. My main assigned responsibility was implementing the comment-info endpoint. Together with the group, in the meeting we have decided on and implemented the models. • I have opened the issue #84 which is about implementing the comment-info endpoint. In general, comment-info endpoint is simply gets a specific comment and updates it with the given fields. • Together with Halil Burak Pala, we have found an external API for fetching the weather data of a given city. You can access the API with this link. • I have implemented the GET and POST parts of the comment-info endpoint. Documented them on the postman with the example usages.

	• After writing the backend of the API and documenting, I have created a pull request #90 related with issue #84 • At first, I have made a very simple UI with some HTML and CSS codes, then pushed them together with the backend of the API as they are satisfying the simple requirements. • With pull request #110, I have pushed the simple UI and the backend of the endpoint. Then closed the related issue #84 • I have opened issue #111 which is about implementation of the unit tests of the endpoint. • I have implemented url and view unit tests. Committed and pushed them to our GitHub. Merged them with the pull request #124. After merging and getting reviews of my group mates, Issue #111 automatically closed. • I have opened issue #143, which is the overall enhancements in the endpoint. Such as, some minor bug fixes, moving the API key to the .env file. Also, I have created a button that goes to our home page in the app to my index page. • With pull request #155, I have merged these changes with the reviews of my group mates and again, issue #143 is automatically closed after merging the relevant pull request. • Reviewed issues and pull requests: #86, #91, #94, #95, #98, #135, #137, #138 • I have researched dockerization and AWS systems. • Participated the preparation of the requirements, class and use case diagrams of our practice app. • I have drawn the sequence diagram of the comment-info endpoint.
Yavuz Samet Topçuoğlu	• Researched about HTTP Protocols, API's and RESTful API's. • Implemented a single page application for practice and build the backend of a project using django. • Attended Meeting 10.1. We discussed about the implementation and the description of practice-app. We talked about the APIs researched. • Attended Meeting 10.2. We divide labor for the practice-app. The initial models and some necessary files are implemented during the meeting. I took the responsibility for post app which creates a post and returns all posts. • Started to work on post app by creating issue #83. • Implemented the backend of the post app by implementing Post API. See pull request #92 • Added external API to the app. See pull request #99 • Tested the API from Postman. • Documented the API from Postman Documentation and published it. • Made research about testing in django and forms in django.

	Watched these videos: testing, form. ● Implemented the frontend of the post app by implementing additional methods, HTML templates and CSS. ● Implemented unit tests for post app. See issue #101 ● Implemented new feature to the post app. User info is fetched from session. See issue #121. See pull request #136 ● Occasionally made improvements. See issue #102 ● Fixed a bug related to external API. See issues #135, #137. See pull request #139 ● Fixed URL direction problem of post app. See pull request #123 ● Updated the href links of buttons. See pull request #141 ● Country name input is made case-insensitive. See pull request #159. See issue #135 ● Handled pull requests: #80, #98, #114, #116, #127, #133, #138, #157. ● Updated and uploaded Practice App: Class Diagram to corresponding wiki page. ● Drew the sequence diagram. Practice App: Creating a post. ● Studied Docker and AWS.

Evaluation of the Tools Used

Slack

Slack is a proprietary business communication platform. It is our main communication tool to communicate with the customer. Besides communicating with the customer, we used Slack for discussing what should be done throughout the project using different channels on the platform.

Discord

Discord is an instant messaging and digital distribution platform where users communicate with voice calls, video calls, and/or text messaging. Since it is easier to use than Google Meets, and it has the ability to save chats, we decided to switch to Discord as our main communication platform between group members.

Zoom

Zoom Meetings is a proprietary video telephony software program. Zoom is our main video communication tool to communicate with the customer. Since it has a time limit of 40 minutes for non-premium users, we did not use Zoom between group members.

Lucidchart

Lucidchart is a web-based proprietary platform that allows users to collaborate on drawing, revising and sharing charts and diagrams. We used Lucidchart to create Use Case, Class, and Sequence Diagrams. Being able to work collectively and simultaneously on a single diagram massively helped us to save a great amount of time.

ProjectLibre

ProjectLibre is an open source project management software solution. We used ProjectLibre to illustrate our project plan. After using the software enough to get used to it, it came in handy but the adaptation period was not negligible.

GitHub

Github provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project. We used these features of GitHub mainly through issues and communicated with each other on the platform. The variety of features helped us throughout the process. Also creating new branches and merging them to master via pull requests is essential to develop the practice app.

Google Sheets

Google Sheets is a spreadsheet program**.** We used Google Sheets to form the Responsibility Assignment Matrix (RAM). As in LucidChart, collective and simultaneous work helped us to save time.

Google Docs

Google Docs is an online word processor. We used Google Docs to document the Milestone 2 Report. Google Docs unburdened us to put the works together. Every member documented on the same file collectively.

Visual Studio Code

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS.Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. In the Stack Overflow 2021 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 70% of 82,000 respondents reporting that they use it. In our project, it accelerates the project development process.

Pycharm

Pycharm is an IDE that most of the Python developers are familiar with so some of us decided to use it. It was not easily used with git compared to Visual Studio so it is probably a worse choice. We had to make all the commits in the terminal. Also, some of the code that works properly (especially the import lines) was giving errors for no reason in Pycharm.

Docker

Docker, a subset of the Moby project, is a software framework for building, running, and managing containers on servers and the cloud. The term "docker" may refer to either the tools (the commands and a daemon) or to the Dockerfile file format. It’s very helpful to dockerize the the project before the deployment.

Amazon Web Services

AWS (Amazon Web Services) is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.
We use an EC2 instance to deploy the project. It is very easy to use.

Requirements

You can find the requirements for the practice app here.

Design

You can find the Use Case Diagram here.
You can find the Class Diagram here.
You can find Sequence Diagrams here.

URI of the Tag of Our Project

https://github.com/bounswe/bounswe2022group5/releases/tag/v1.0.1

Code of Our Project and Instructions

You can find the code of our project here.

How to run in local:

1. Clone our repo to your local:

   git clone https://github.com/bounswe/bounswe2022group5.git

2. cd into the practice-app folder:

   cd practice-app

3. Create a virtual environment:

   python3 -m venv env

4. Activate the virtual environment:

   source env/bin/activate

5. Install the dependencies:

   python3 -m pip install -r requirements.txt

6. Provide the necessary environment variables for external API keys in a .envfile in the current directory.

7. Run the application:

    python3 manage.py runserver

8. App should be available at http://localhost:8000/.

How to run using docker:

1. Make sure you installed and are able to use docker. Make sure you can use docker command.

2. Clone our repo and cd into the practice-app. Provide a valid .env file to use external APIs.

3. Create an image named practice-app-group-5 of our app:

    docker build -t practice-app-group-5 .

4. Create a container and run our application:

    docker run -p 8888:8000 practice-app-group-5

5. App should be available at http://localhost:8888/.
   Since we specified port 8000 in the Dockerfile, our app is reachable by 8000 port of the container. We created a connection between the port 8000 of the container with our localhost's 8888 port. Ports can be changed.