CMPE 352
Fundamentals of Software Engineering

SPRING'22 / GROUP 6 - M2 Deliverables

Ali Kaan Biber
Alp Eren İnceoğlu
Aral Dörtoğul
Artun Akdoğan
Bedirhan Pamukçuoğlu
Berfin Şimşek
Beyza İrem Urhan
Hakan Balık
Hatice Erk
Yasir Dikbaş
Yusuf Erdem Nacar

Deliverables

Milestone Report 2

Executive Summary

Summary

The practice app we worked on is a Django REST Framework web application that offers a variety of functionalities that can be accessed via a central hub. The functionalities provided by the application vary from medical information systems to daily use cases.

URL of the Deployed Application

The URL of the deployed practice application is:
http://ec2-3-95-153-239.compute-1.amazonaws.com/

Functionalities

- **Advice**

  This user endpoint offers personalized medical advice depending on the information provided by the user on a wide variety of topics listed on the UI home page. To achieve this, the endpoint makes requests to the free-to-use MyHealthFinder API. The endpoint also provides various statistics about the previous users. The information provided by the endpoint is also available in JSON format.

- **Comment**

  This endpoint lists all the comments in the application's database. The user can view the details of a single comment, or create a new comment and add it to the database by providing the necessary fields such as text, author, publish date, parent post, etc. Moreover, the endpoint provides a random inspirational quote from an open-source web API called "quotable" on top of the comment list.

- **Coronavirus**

  This endpoint aims to present nearly daily updated coronavirus data. Users can choose a region from the list. They can type some substring of the region that they're looking for to find it. It uses CoronavirusAPI to fetch data published by Johns Hopkins University and returns in JSON format. Then, it takes the last Update, the number of confirmed cases, recovered cases, deaths, and the fatality ratio from the raw data and displays the corresponding fields for the chosen region.

- **News**

  This endpoint's aim is to retrieve news. It uses the NewsAPI service to get the news. The endpoint provides two options which are getting news with and without parameters. The parameterized one required the user to select a country, enter a keyword, select a start date and choose the sort option. These options are all optional except for the latest one. If wanted, the user can click on the image of the retrieved news to browse the original source.

- **Drugs and Side Effects**

  Aim of this endpoint is when the user requested to see the side effects of a medicine, to provide the known side effects to the user in an easy way. There are two other options that user can use, first one is adding some side effects to a medicine, with the name of the medicine and the side effect. Other one is, when you don't sure about the full name of a medicine, there is a search api which provides the image of the medicine and a link to it's description. A front-and view added in order to increase user experience.

- **Location Manager**

  Aim of this endpoint is to keep track of the users location information and access them when necessary. With this application, users are able to add-update-delete user location information, view users near location and a user's location information. For the dynamic endpoint, location dropdowns fetch required information effectively from database (the information on database is initialized from world-cities.csv) with api and choose user's location on dropdowns from their IP if possible. For location information, ip-api external api is used.

- **User**

  The goal of this endpoint is to control the information of the users. This endpoint allows users to list all users or list a specific user with a given username. They can access the usernames, e-mails, passwords and Zodiac signs of users. Also, when listing a specific user, they can access the daily horoscope of the user thanks to aztro. In addition to that, Users can create a user by providing a unique username, unique e-mail, a password and a zodiac sign.

- **Post**

  Aim of this endpoint is to create new posts and view these post details. Users can create a post by providing a title, description and post type label. Also, by providing a post ID, they can view a post's details such as ID, title, description, post type label and date of posting. For the weather forecast in Istanbul on homepage, OpenWeather external api is used.

- **Article**

  Goal of this endpoint is to search articles based on the user's subject choice and keep track of them. Users can search for articles by providing a subject and an integer representing the maximum number of articles to be found. To achieve this task, this endpoint makes use of the arXiv API. Users can also list the articles they have searched before and clear the article history if they wish.

- **Hospital**

  Goal of this endpoint is to list hospitals by using search bar according to their locations(states). Also user can add new hospital to the list by providing necessary information such as hospital id, state and name. In order to do this task I utilized from external api by making requests to Community Benefit Insight API. If valid input is provided api responds with a thanks page otherwise displays error.

Challenges

As a group, we faced several challenges throughout the development of the app. Although many of these challenges were about time, we beat these challenges and came through the process successfully.

The biggest challenge we faced was organizing right after the milestone 1 report and deciding on what we should do next. Because the end of milestone 1 coincided with spring break, many of us were exhausted and deserved a well-earned rest. However, by the end of the break, we got together and decided on what we should build and what we should use to build it.

The second challenge we faced was coordinating for the milestone 2 group submission. Although we were not prepared for such a task because we were informed that the submissions were individual, we came through the challenge rather quickly and split the work that needed to be done in a way that can be done in a day.

Status of the Deliverables

Group Milestone Report 2

Status

Milestone 2 report is done. The report summarizes the group effort in the development of the practice application, lists the status of the deliverables, and includes a detailed explanation of the practice app and its development process. The report also includes several other related items such as links to the GitHub wiki page, links to the code, and links to the application itself. Overall the report provides well-written documentation for everything related to the practice app.

Requirements

Status

The requirements for the practice app are done. It consists of sub-sections for each endpoint. Since many of the endpoints are not related functionally, categorizing the requirements by the endpoints they are associated with ensures better readability and conveys the information in a much clearer way. Overall, the requirements provide a high-level guideline to what each point should do and provide.

Design

Status

The UML diagrams which are use case, class, and sequence are done. The diagrams are designed in a partite way to better convey the information. The use cases are grouped under their respective endpoints. The class diagram is mostly composed of discrete class groups because many of the endpoints are not functionally related. Finally, there is a sequence diagram for each endpoint that describes the most crucial and/or complicated functionality that the endpoint offers. As a whole, the UML diagrams provide a holistic and high-level view of how the various endpoints of the practice app function.

URI Tag of the Application

Status

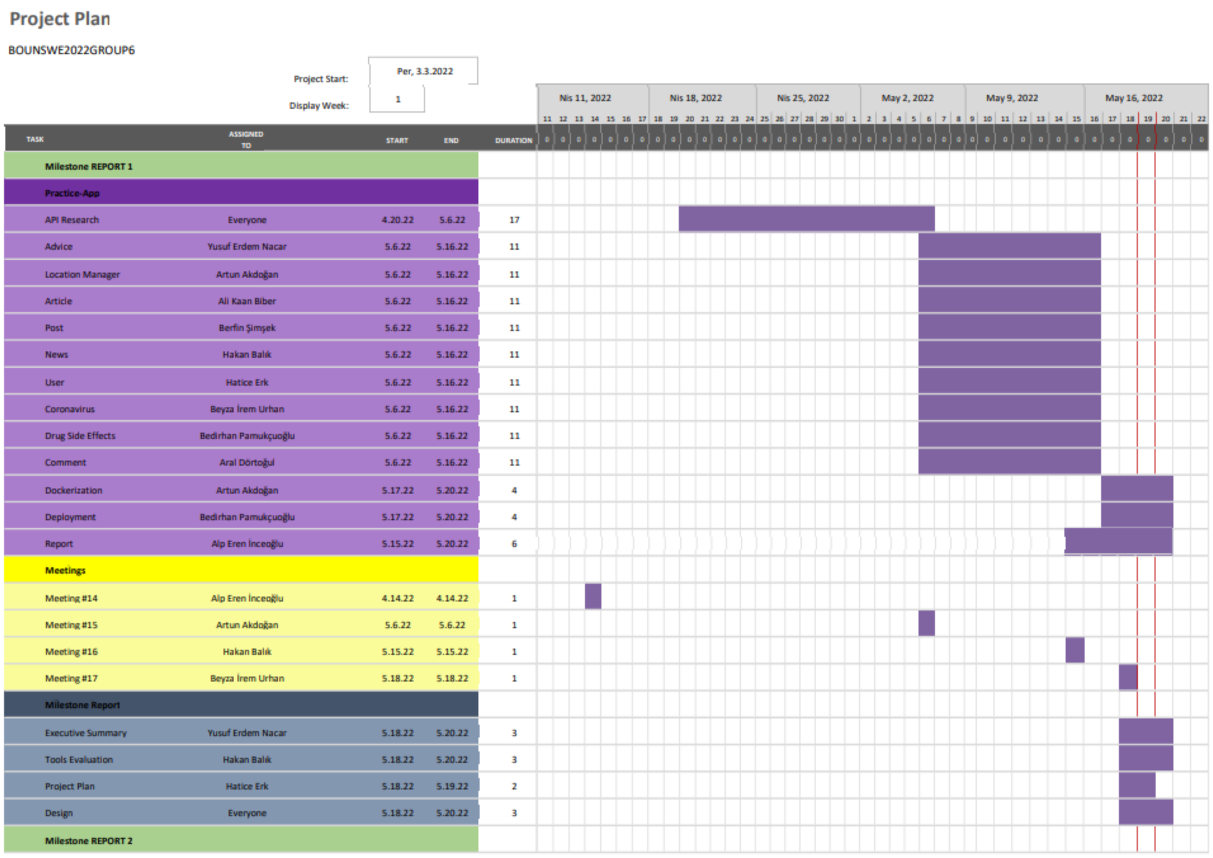
The URI Tag of the application is created with the appropriate description. Detailed information about the URI tag is present under the URI Tag of the Application deliverable title.

Code

Status

The code for the application is present on GitHub and it is well-reviewed and tested. The full instructions on how to build and run the application are present under the Code deliverable title.

Project Plan



Project Plan

BOUNSWE2022GROUP6

Project Start: Per, 3.3.2022

Display Week: 1

TASK	ASSIGNED TO	START	END	DURATION
Milestone REPORT 1				
Practice-App				
API Research	Everyone	4.20.22	5.6.22	17
Advice	Yusuf Erdem Nacar	5.6.22	5.16.22	11
Location Manager	Artun Akdoğan	5.6.22	5.16.22	11
Article	Ali Kaan Biber	5.6.22	5.16.22	11
Post	Berfin Şimşek	5.6.22	5.16.22	11
News	Hakan Balık	5.6.22	5.16.22	11
User	Hatice Erk	5.6.22	5.16.22	11
Coronavirus	Beyza İrem Urhan	5.6.22	5.16.22	11
Drug Side Effects	Bedirhan Pamukçuoğlu	5.6.22	5.16.22	11
Comment	Aral Dörtoğul	5.6.22	5.16.22	11
Dockerization	Artun Akdoğan	5.17.22	5.20.22	4
Deployment	Bedirhan Pamukçuoğlu	5.17.22	5.20.22	4
Report	Alp Eren İnceoğlu	5.15.22	5.20.22	6
Meetings				
Meeting #14	Alp Eren İnceoğlu	4.14.22	4.14.22	1
Meeting #15	Artun Akdoğan	5.6.22	5.6.22	1
Meeting #16	Hakan Balık	5.15.22	5.15.22	1
Meeting #17	Beyza İrem Urhan	5.18.22	5.18.22	1
Milestone Report				
Executive Summary	Yusuf Erdem Nacar	5.18.22	5.20.22	3
Tools Evaluation	Hakan Balık	5.18.22	5.20.22	3
Project Plan	Hatice Erk	5.18.22	5.19.22	2
Design	Everyone	5.18.22	5.20.22	3
Milestone REPORT 2				

Summary of Work Done

Team Member	**Contribution**
Ali Kaan Biber	● Attended all the group meetings. ● Researched about APIs and Django Rest framework ● Researched about how to use git pull requests, merging branches and branch operations. ● Learned Django framework ● Made a web search to find an external API to use in practice application ● Specified the basic functionalities for Article App. #103 ● Implemented the Article App for practice-app. ● Learned how to use css files for styling and found a few to visually enhance my implementation ● Learned how to write unit tests ● Wrote unit tests to test my app and opened a pull request. #107 ● Reviewed Comment Application pull request. #116 ● Reviewed News Application pull request. #130 ● Corrected the errors in unit tests for Article App and made a pull request. #129 ● Generated Use Case Diagram for the Article Application. #141 ● Generated Class Diagram for the Article Application. #141 ● Generated Sequence Diagram for the Article Application. #141 ● Documented the functionality of Article App to Milestone Report 2. #144 ● Listed the requirements for the Article App in Milestone Report 2. #144 ● Listed the individual work in Milestone Report 2. #145
Alp Eren İnceoğlu	●
Aral Dörtoğul	● Attended all group meetings. ● Learned how to use the Django framework. ● Did research about the git version management system. ● Studied about API generation and external API usage. ● Implemented the Comment Application with quotable external API in the practice application. ● Opened pull requests #116 and #133 for the Comment API. ● Reviewed Drug Side Effects Application pull request (#121). ● Reviewed Article Application pull request (#129).

	• Reviewed Advice App Documentation pull request. (#151). • Did research about dockerization and deployment. • Used Postman to test the functionality of the Comment Application. • Tested the properness and functionalities of all the sub-applications during Meeting #17. • Created an Amazon AWS EC2 instance with the team and deployed the application to the server using the docker image. • Generated Use Case Diagram for the Comment Application. (#141) • Generated Class Diagram for the Comment Application. (#141) • Generated Sequence Diagram for the Comment Application. (#141) • Prepared Individual Milestone 2 Report. • Documented the functionalities and requirements of the Comment API to the Milestone 2 Report. (#144) • Documented the individual work carried out by him in the Milestone 2 Group Report. (#145)
Artun Akdoğan	• Attended all meetings • Researched Restful APIs • Practiced with the code snippets handed over after the lecture • Practiced git by pushing a file to the repository, creating a pull request, and reviewing another pull request • Read the practice application description document • Refreshed my docker knowledge • Researched APIs that I can use for developing the practice application • Learned about Django REST Framework • Developed and wrote tests for Location Manager App (#105) • Reviewed Article App pull request (#107) • Reviewed Advice App pull request (#104) • Wrote Dockerfile for Practice App (#140) • Wrote Home page for Practice App (3f019d) • Wrote init.sh and init-docker.sh to initialize application(#140) (#105) • Deployed Practice App on Amazon AWS with team • Generated Use Case Diagram for the Location Manager. (#141) • Generated Class Diagram for the Location Manager App. (#141) • Generated Sequence Diagram for the Location Manager App. (#141) • Wrote Running Practice App section of Milestone 2 Group Report • Prepared Individual Milestone 2 Report • Documented the functionalities and requirements of the Location Manager API to the Milestone 2 Report (#144) • Documented the individual work carried out by me in the Milestone 2

	Group Report (#145) ● Other minor fixes directly committed to master branch
Berfin Şimşek	● Attended all group meetings. ● Researched the Django Rest framework. ● Researched about pull requests, merging and branches. ● Researched API generation and external API usage. ● Researched HTML tables and bootstrap. ● Implemented the Post App in practice-app. ● Opened a pull request for the post app. ● Learned unit tests for testing my app. ● Researched different external API's to use in the post app. ● Added unit tests and weather forecast external API to post app. #127 ● Reviewed User Application pull request. ● Reviewed Advice Application pull request. ● Generated Use Case Diagram for the Post Application. ● Generated Class Diagram for the Post Application. ● Generated Sequence Diagram for the Post Application. ● Filled Creating a Post requirement. ● Filled Viewing the details of a Post requirement. ● Filled functionalities of Post Application in the "Executive Summary" part of this document. ● Helped generating the project plan and reviewed it. ● Reviewed Yusuf's API documentation.
Beyza İrem Urhan	● Attended all the group meetings about the demo application except for #15. ● Created the meeting notes for Meeting 17. ● Researched, learned and practiced about Django Rest framework ● Refreshed my knowledge on HTML ● Made a web search to find an appopriate external API to use in my practice application ● Listed the basic functionalities for Coronavirus App. #144 ● Work on the implementation of Coronavirus App for practice-app. #136 ● Reviewed Comment Application pull request. #133 ● Reviewed Location Manager Application pull request. #106 ● Generated Use Case Diagram for the Coronavirus Application. #141 ● Generated Class Diagram for the Coronavirus Application. #141 ● Generated Sequence Diagram for the Coronavirus Application. #141 ● Documented the functionality of Coronavirus App to Milestone Report 2. #144

	● Listed the requirements for the Coronavirus App in Milestone Report 2. #144 ● Listed the individual work in Milestone Report 2. #145
Hakan Balık	● Attended all group meetings regarding demo application except for #15. ● Created the meeting notes for Meeting 16. ● Learned Django framework. ● Studied and refreshed my knowledge about API generation and external API usage. ● Created News Application in practice-app. ● Investigated and commented on the first test pull request. ● Reviewed Advice Application pull request. ● Reviewed User Application pull request. ● Reviewed Post Application pull request. ● Reviewed Drug Side Effect Button pull request. ● Created News App pull request. ● Created Dockerization pull request. (PR #140) ● Created News App Test pull request. ● Reviewed Home App Button Style pull request. (PR #150) ● Reviewed Advice App Documentation pull request. (PR #151) ● Used Postman to test the functionality of the News Application. ● Tested the properness and functionalities of all the sub-applications during the meeting. ● Attended and helped with the dockerization and deployment in the meeting where we worked on the demo application. ● Wrote the evaluation of tools part of this document. ● Generated Use Case Diagram for the News Application. ● Generated Class Diagram for the News Application. ● Generated Sequence Diagram for the News Application. ● Filled Getting the News requirements. ● Filled Viewing the News requirements. ● Filled functionalities of News Application in the Executive Summary part of this document. ● Reviewed the Executive Summary of this document. ● Filled the work I did during the demo application here. ● Reviewed, exported and uploaded each use case, class and sequence diagrams to Milestone Report 2. (#153) ● Cropped and uploaded Project Plan Image to this document. ● Worked on the styling of this document. ● You can visit my wiki page or check my individual report to view it better.

Hatice Erk	● Attended all group meetings. ● Researched about the Django Rest framework. ● Researched about Git such as pull requests, merging and branches. ● Studied on HTML tables and UI. ● Worked on API generation and external API usage. ● Reviewed the Initial Files pull request. #98 ● Implemented the User-App in practice-app. #109 ● Added a daily horoscope functionality to use external api. #115 ● Implemented tests for User-App. #124 ● Created pull request to merge my work on user-app to master. #108, #125, #139 ● Reviewed the first Article App pull request. #107 ● Reviewed the first Post App pull request. #112 ● Reviewed the first Comment App pull request. #116 ● Reviewed Post App: External API and Tests pull request. #127 ● Reviewed Advice App: Database Support, Bad Input Handling and Tests pull Request #128 ● Reviewed the first News App pull request. #130 ● Reviewed Comment App: Updated pull request. #133 ● Researched about dockerization and deployment. ● Made minor updates on the master branch. ● Created a shared Lucidchart to work on design tasks together. #141 ● Generated Use-Case Diagram for the User-App endpoint. ● Generated Class Diagram for the User-App endpoint. ● Generated a Sequence Diagram for the User-App endpoint. ● Updated the project plan with the Practice-App project. #148 ● Documented the basic functionalities of User API. #144 ● Listed the requirements of User API. #144 ● Documented my individual work in the Group Report. #145 ● Created Individual Milestone 2 Report. ● Updated my wiki page with weekly efforts.
Bedirhan Pamukçuoğlu	● Attended all group meetings. ● Learned how to use the Django framework. ● Did research about the git version management system. ● Studied about API generation and external API usage. ● Searched APIs from rapidapi.com ● Implemented the Drug Side Effects in practice-app. ● Opened pull requests #121 for the Drugs and Side Effects APP. ● Reviewed Location Manager Application pull request (#106).

	• Reviewed Coronavirus Application pull request (#136). • Reviewed Hospital List Application pull request (#142). • Did research about dockerization and deployment. • Updated init.sh to remove if exists database folder. (master) • Registered to AWS system to create an instance. • Created an Amazon AWS EC2 instance with the team and deployed the application to the server using the docker image • Generated Use Case Diagram for the Drugs and Side Effects App. (#141) • Generated Use Case Diagram for the Drugs and Side Effects App. (#141) • Generated Sequence Diagram for the Drugs and Side Effects App. (#141) • Prepared Individual Milestone 2 Report • Documented the functionalities and requirements of the Drugs and Side Effects App to the Milestone 2 Report. • Documented the individual work carried out by me in the Group Report. • Opened pull requests #121 for the Drugs and Side Effects APP. • Reviewed News App Tests pull requests (#147) • Approved Dockerization pull request (#140)
Yasir Dikbaş	• Attended all group meetings except for #17. • Made research about Django Rest Framework, read documentation and watched this video to get a better grasp of it. • Documented my weekly efforts on my personal wiki page. • Brushed up my git knowledge by watching this video. • Generated Use Case Diagram for the Hospital App. (#141) • Generated Class Diagram for the Hospital App. (#141) • Generated Sequence Diagram for the Hospital App. (#141) • Researched HTML and bootstrap. • Researched API examples from rapidapi and learned how to integrate external API to the API that I developed. • Implemented Hospital App in the practice_app folder on master. • Prepared personal Milestone Report 2 • Prepared part of Group Milestone Report 2 • Opened Hospital List Application pull request and added 10 commits (#142). • Made research on dockerization and deployment. • Reviewed pull requests #121 for the Drugs and Side Effects APP • Reviewed pull request regarding Home Page Change (#150) • Documented the basic functionalities of Hospital API. #144 • Learned how to write unit tests • Learned how to use Postman to test the functionality of my Application.

	● Listed the individual work in Milestone Report 2. #145
Yusuf Erdem Nacar	● Did preliminary research on various web application development frameworks such as Django, Flask, Masonite, etc. to get an idea about what we should use. ● Proposed using Django REST Framework for the practice application. ● Replicated the functionalities of the example flask codes used in the lecture in Django REST Framework for exercise before starting the development of the practice application. ● Attending meeting #15 to discuss what we should learn and what we should build. As a group, we decided that everyone should be free to choose what functionalities their apps should provide. ● Decided on creating an app that offers personalized medical advice. ● Did research about medical APIs. Found MyHealthFinder API and decided to use it. ● Generated the initial project files in Django REST Framework and initialized the project files on the GitHub repository. Issues: #99. Pull Requests: #98. ● Created an application named "advice" that provides personalized medical advice according to the information provided by the user and supports GET and POST requests. Issues: #101, #118. Pull Requests: #104. ● Did in-depth research about Django Templates to create a user interface for the advice app. ● Added a simple user interface to the advice app using Django Templates. Issues: #101, #102. Pull Requests: #104. ● Discussed the necessity of database support with Suzan Üsküdarlı over at Slack and decided to add user statistics to the advice app. Issues: #110. ● Did in-depth research about Django Models to store the information of the users. ● Added user statistics to the advice app using Django Models. Issues: #110. Pull Requests: #128. ● Attended meeting #16 to set deadlines for the development of individual apps and plan what can and should be done in parallel and after the development is done. ● Did in-depth research about Django Tests to prepare unit tests for the advice app advice-api view class. ● Added the unit tests for the advice app advice-api view class. Issues: #120. Pull Requests: #128. ● Reviewed Berfin's pull request. Found a bug and reported it. Pull Requests: #127.

	● Reviewed Hatice's pull request. Suggested changing the status code for some cases. Pull Requests: #125. ● Reviewed and tested Ali's bugfix for the article app tests. Pull Requests: #129. ● Attended meeting #17 for dockerizing and deploying the application on an AWS EC2 instance and distributing the work for the milestone 2 group report. Volunteered for the executive summary part of the milestone 2 group report. ● Reviewed and tested Artun's back button addition to the location manager app. Pull Requests: #137. ● Reviewed and tested Hatice's status code update suggested in #125. Pull Requests: #139. ● Revived the Dockerfile and init-docker.sh additions. Pull Requests: #140. ● Wrote the executive summary part for the milestone 2 report on the shared google doc document. Issues: #144. ● Added UML diagrams for the advice app to the shared Lucidchart project. Issues: #141. ● Reviewed and tested Hakan's test and back button additions. Pull Requests: #147. ● Created API documentation for the advice app. Pull Requests: #151. ● Reviewed Bedirhan’s comment addition. Pull Requests: #152.

Evaluation of Tools

1) **GitHub:**

   As discussed in the previous milestone, GitHub is the most common and most popular Git repository hosting environment. In our demo application project similar to our prior work we usually benefitted from issues and wiki pages for documentation. But also, we used branches and pull requests which helped us develop the project in a parallel manner and eased the job of tracking.

2) **Django:**

   Django is a Python-based web framework that is free and open-source. It is very useful for fast development since most of the setup jobs are pre-implemented. We initialized the root of our demo application by using the *manage.py* file and similarly each individual created their own sub-applications using the same file. This framework also eases the job of database interactions since it is integrated with SQL. On top of that, each sub-project contains its respective test files and it is really easy to test them as a whole or separately using pre-implemented commands.

3) **REST API:**

   REST is an architectural style for distributed hypermedia systems that enables the users to interact with RESTful web services. While developing our sub-projects, we were asked to use external APIs. That’s why we used REST APIs with the requests library of Python to retrieve data from open and different resources.

4) **Docker:**

   Docker is one of the most popular container systems. Some of the advantages of using docker are: consistency, cost-effectiveness, enables fast deployment, the ability to run everywhere, collaborativeness, and easy automation. During our demo application journey, we used Docker to create an image which will be then used on the EC2 instance of AWS for deployment.

5) **AWS:**

Amazon Web Services is a platform with various cloud-based products and solutions. It is popular since it is easy to use, as mentioned before it has a lot of diverse tools, it has unlimited server capacity, and last but not least it is very secure. We used the EC2 (Elastic Compute Cloud) container instance to deploy our demo application using the image that we have generated using Docker.

6) **Postman:**

Postman is mostly used for investigating APIs. It is also often used to build APIs as well. There are a lot of tools that help accelerate the API Lifecycle and also it offers to work collaboratively with the team workspace feature. In our demo application, we used Postman to test both external and implemented APIs if they were working properly and giving the expected results.

Requirements

1. **Creating a Post**
   - 1.1. Users shall provide a title when creating a post.
   - 1.2. Users shall provide a description when creating a post.
   - 1.3. Users shall provide a post type label when creating a post.

2. **Viewing the Details of a Post**
   - 2.1. Users shall be able to see a post's details such as ID, title, description, post type label, and posting date when they provide the ID of the post that they want to view.

3. **Weather Forecast**
   - 3.1. Users shall be able to see the temperature in Istanbul on the homepage of the post app.

4. **Creating a Comment**
   - 4.1. Users shall provide a description text when creating a comment.
   - 4.2. Users shall provide an author when creating a comment.
   - 4.3. Users shall provide a publish date when creating a comment.
   - 4.4. Users shall provide an upvote count when creating a comment.
   - 4.5. Users shall provide a downvote count when creating a comment.
   - 4.6. Users shall provide a parent post when creating a comment.
   - 4.7. Users may provide NSFW tag when creating a comment.

5. **Viewing the Details of a Comment**
    5.1. Users shall be able to receive a trigger warning if the comment is tagged as NSFW.
    5.2. Users shall be able to see the comment’s ID.
    5.3. Users shall be able to see the comment’s description text.
    5.4. Users shall be able to see the comment’s publish date.
    5.5. Users shall be able to see the comment’s author.
    5.6. Users shall be able to see the title of the comment’s parent post.
    5.7. Users shall be able to see the ID of the comment’s parent post.
    5.8. Users shall be able to see the upvote count of the comment.
    5.9. Users shall be able to see the downvote count of the comment.
    5.10. Users shall be able to go back to the list of all comments from the detail page.

6. **Inspirational Quotations**
    6.1. Users shall be able to see a random inspirational quote on top of the comment API’s home page.

7. **Viewing the List of All Comments**
    7.1. Users shall be able to see the list of all the comments in the order of publish date, with the author and the time that has passed since the publish date.
    7.2. Users shall be able to go back to the main page of the practice app from the comment list page.

8. **Getting the News**
    8.1. Users shall be able to get news without giving any parameters.
    8.2. Users shall be able to get news by providing the following fields as optional: country, keyword, start date and sort option.

9. **Viewing the News**
    9.1. Users shall be able to see the title of the news.
    9.2. Users shall be able to see the description of the news.
    9.3. Users shall be able to see the image of the news.
    9.4. Users shall be able to redirect to the source link of the news once they click on the news.

10. **Managing User’s Location Information**
   10.1. Users shall be able to add/update their location information.
   10.2. Users shall be able to delete their location information.

11. **Viewing User’s Location Information**
   11.1. Users shall be able to view specific user’s location informationç
   11.2. Users shall be able to view users that are near a location.

12. **User’s Location Information Endpoint**
   12.1. Endpoint shall be able to pull location list effectively according to selected items on dropdown
   12.2. Endpoint should be able to automatically select the user's location information on dropdowns according to the user's IP.

13. **Creating a Drug - Side Effect Pair**
   13.1. Users shall be able to add a new drug side effect pair by providing the name of the medicine and the list of side effects in a comma separated list.
   13.2. Endpoint should check if the medicine already exists in the database or not.

14. **Getting a Drug - Side Effect Pair**
   14.1. Users shall be able to get the side effects of a specified medicine by providing the full name of the medicine.
   14.2. Endpoint should return “No medicine found!” if the medicine does not exist in the database.

15. **Searching a Medicine**
   15.1. Users shall be able to search a medicine by providing a part of the name of the medicine.
   15.2. API should return the image, full name and a link to description of the possibles.

16. **Creating a User**
   16.1. Users shall provide a unique username when creating a user.
   16.2. Users shall provide a password when creating a user.
   16.3. Users shall provide a unique email when creating a user.
   16.4. Users shall provide a Zodiac sign when creating a user.

17. Viewing the Details of a User

17.1. Users shall be able to see a user's details such as ID, username, password, email and Zodiac sign when they provide the username of the user that they want to view.

17.2. Users shall be able to see the daily horoscope of the viewed user's Zodiac sign on the detail page of the user app.

18. Searching for Articles

18.1. Users shall provide a search subject when searching for articles.

18.2. Users shall provide an integer representing the maximum number of articles to be found when searching for articles.

18.3. Articles' title, summary, authors and a link to the original article shall be fetched from the search results and added to the article history.

19. Viewing Article History

19.1. Users shall be able to view the previously searched articles without providing any parameters.

19.2. Users shall be able to view the title of the articles.

19.3. Users shall be able to view the summary of the articles.

19.4. Users shall be able to view the authors of the articles.

19.5. Users shall be able to view a link which redirects to the original article.

20. Deleting Article History

20.1. Users shall be able to delete the article history by clicking a button.

21. Advice App

21.1 User Requirements

21.1.1 Users shall be able to view the advice categories.

21.1.2 Users shall be able to view the age distribution of the past users.

21.1.3 Users shall be able to view the sex distribution of the past users.

21.1.4 Users shall be able to view the tobacco use status distribution of the past users.

21.1.5 Users shall be able to view the sexual activity status distribution of the past users.

21.1.6 Users shall be able to get personalized medical advice by providing their age, sex, tobacco use status, and sexual activity status.

21.2 System Requirements

21.2.1 The server shall make a request to the MyHealthFinder API to get the list of advice categories.

21.2.2 The server shall fetch the age distribution data from the database to display.

21.2.3 The server shall fetch the sex distribution data from the database to display.

21.2.4 The server shall fetch the tobacco use status distribution data from the database to display.

21.2.5 The server shall fetch the sexual activity status distribution data from the database to display.

21.2.6 The server shall save the information provided by a user in the database when a user makes a request for personalized advice.

21.2.7 The server shall make a request to the MyHealthFinder API with the information provided by the user to get the personalized advice list when a user makes a request for personalized advice.

22. Coronavirus App

22.1 User Requirements

22.1.1 Users shall be able to view the list of regions.

22.1.2 Users shall be able to search region by keyword.

22.1.3 Users shall provide the region to search data for.

22.2 System Requirements

22.2.1 The server shall make a request to the Coronavirus API to fetch raw data.

22.2.2 The server shall fetch the “last update” data of the region whose name is chosen by the user from the return of the Coronavirus API to display.

22.2.3 The server shall fetch the “confirmed cases” data of the region whose name is chosen by the user from the return of the Coronavirus API to display.

22.2.4 The server shall fetch the “recovered cases” data of the region whose name is chosen by the user from the return of the Coronavirus API to display.

22.2.5 The server shall fetch the “deaths” data of the region whose name is chosen by the user from the return of the Coronavirus API to display.

22.2.6 The server shall fetch the “case fatality ratio” data of the region whose name is chosen by the user from the return of the Coronavirus API to display.

23. Creating a Hospital

23.1. Users shall be able to add a new hospital by providing the necessary attributes such as city, state, name and so on.

23.2. Endpoint should check for the uniqueness of the ID provided.

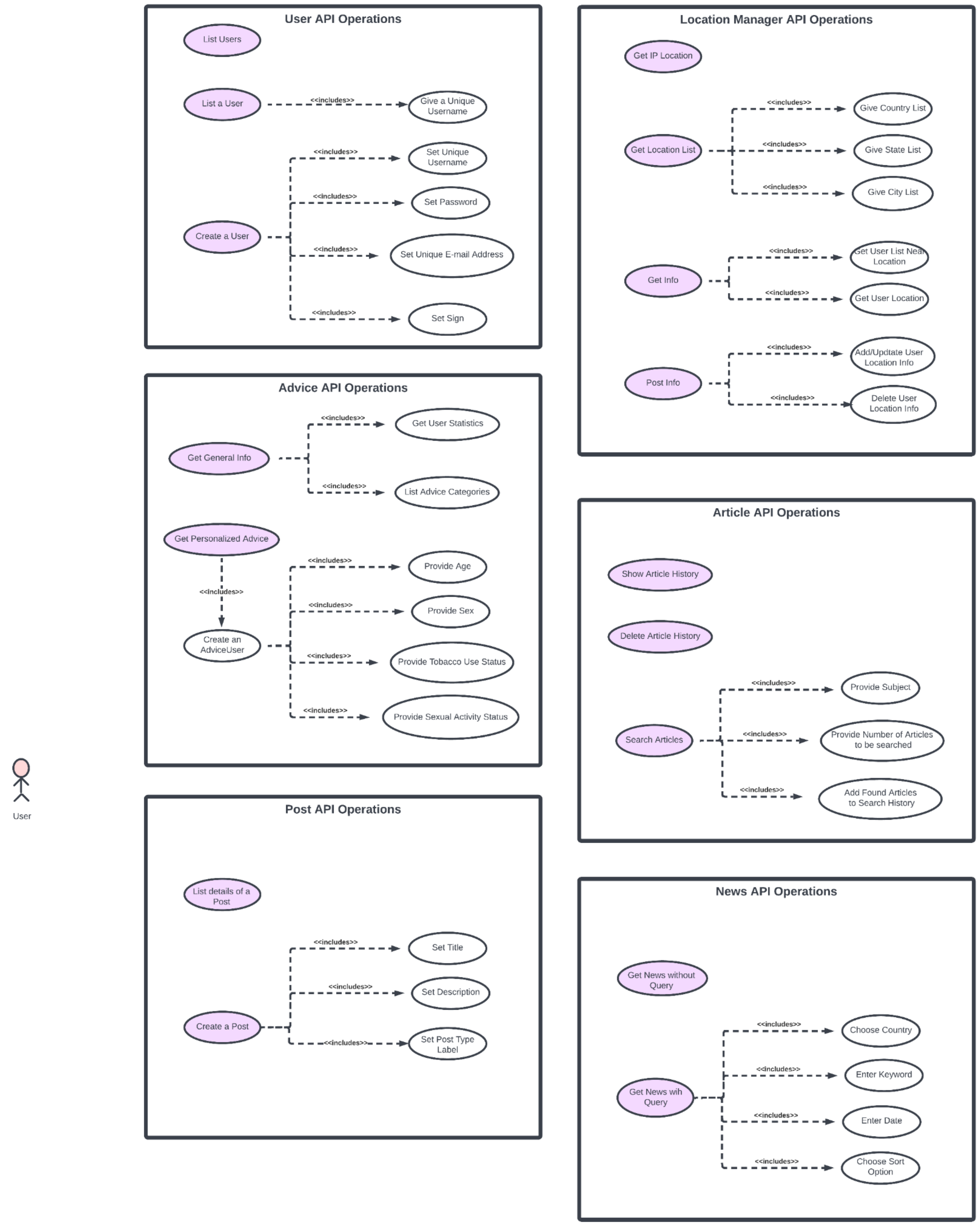
24. Getting Hospital List

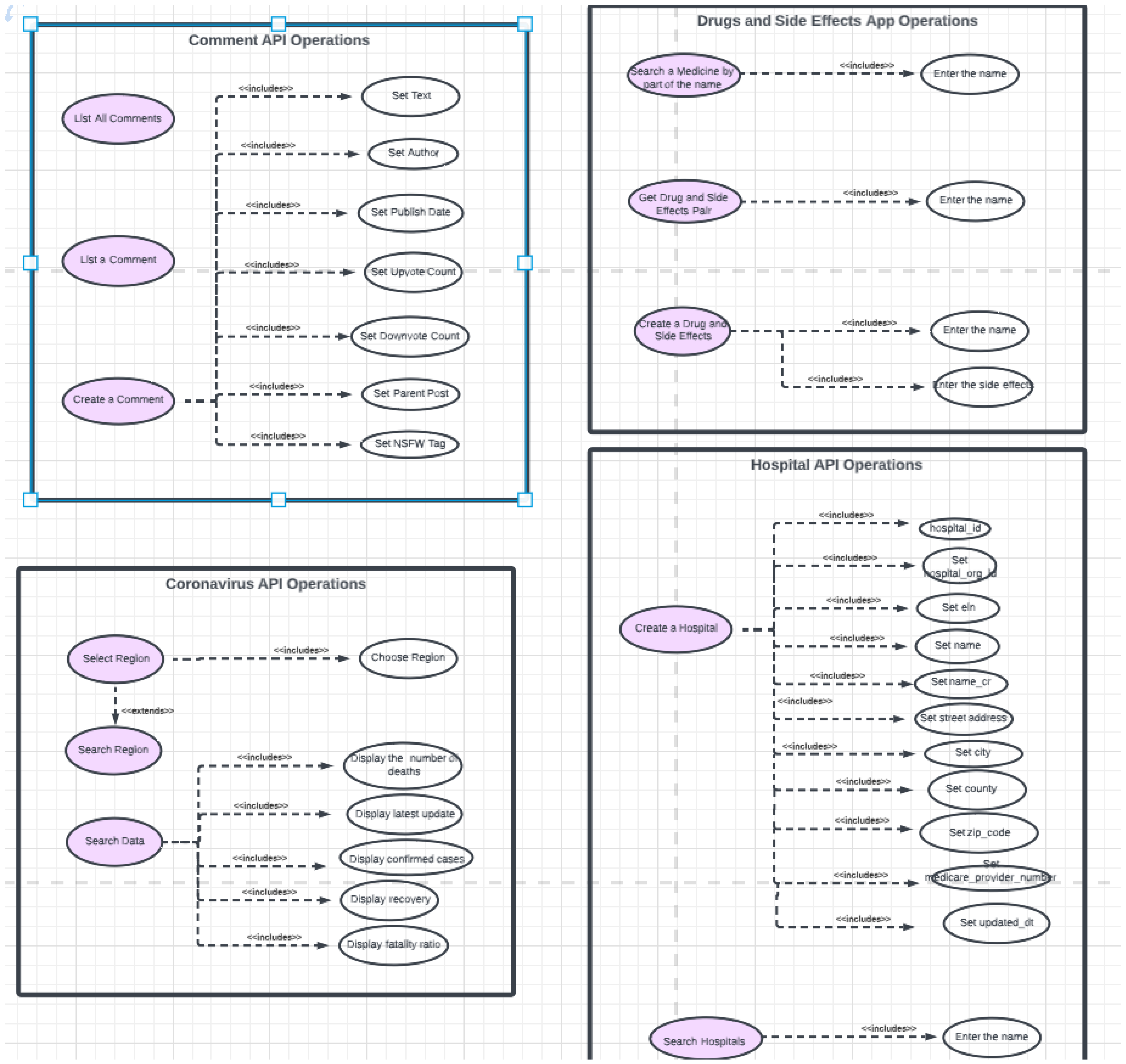
24.1. Users shall be able to get a list of hospitals by providing the name of the state they want to view.

24.2. Endpoint should return “No Hospitals” if the state name is either wrong or not added to the database.

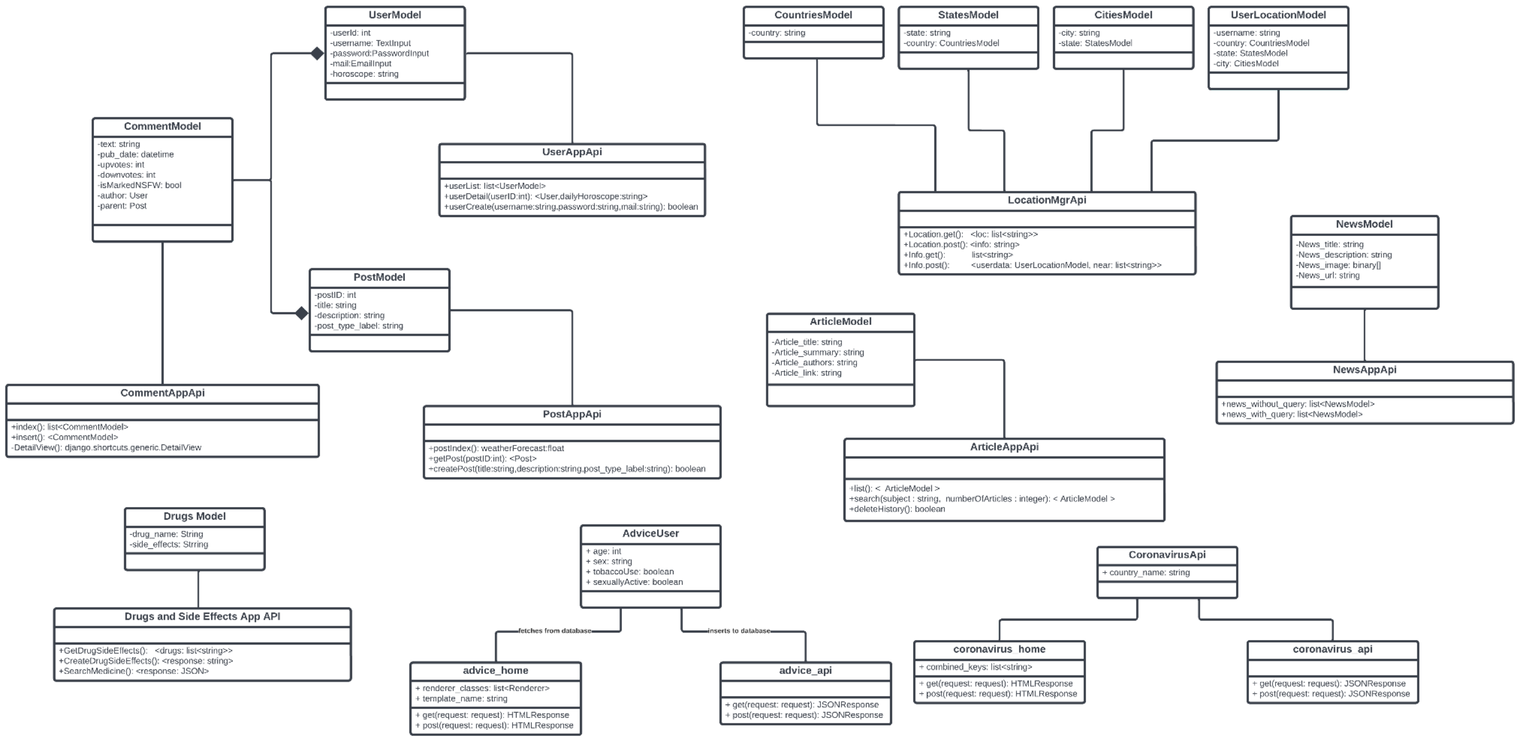
Design

Use Case Diagram



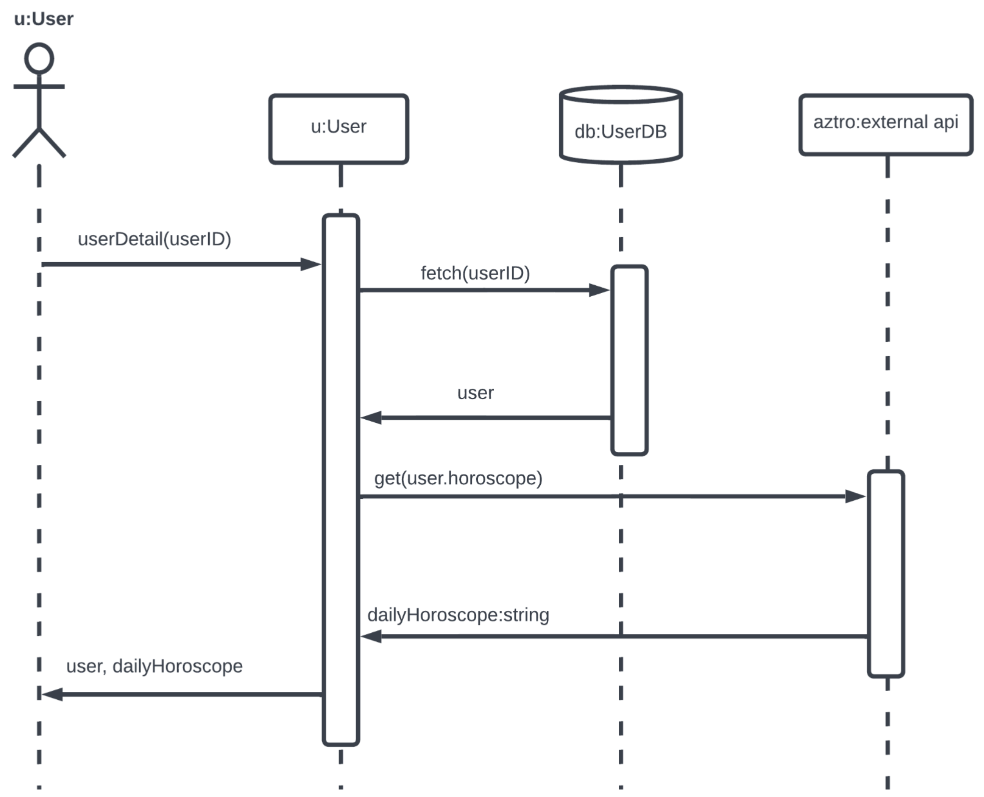

Comment API Operations
List All Comments
List a Comment
Create a Comment
<<includes>>
Set Text
Set Author
Set Publish Date
Set Upvote Count
Set Downvote Count
Set Parent Post
Set NSFW Tag
Drugs and Side Effects App Operations
Search a Medicine by part of the name
Enter the name
Get Drug and Side Effects Pair
Create a Drug and Side Effects
Enter the side effects
Coronavirus API Operations
Select Region
Choose Region
<<extends>>
Search Region
Search Data
Display the number of deaths
Display latest update
Display confirmed cases
Display recovery
Display fatality ratio
Hospital API Operations
Create a Hospital
hospital_id
Set hospital_org_id
Set ein
Set name
Set name_cr
Set street address
Set city
Set county
Set zip_code
Set medicare_provider_number
Set updated_dt
Search Hospitals

Class Diagram

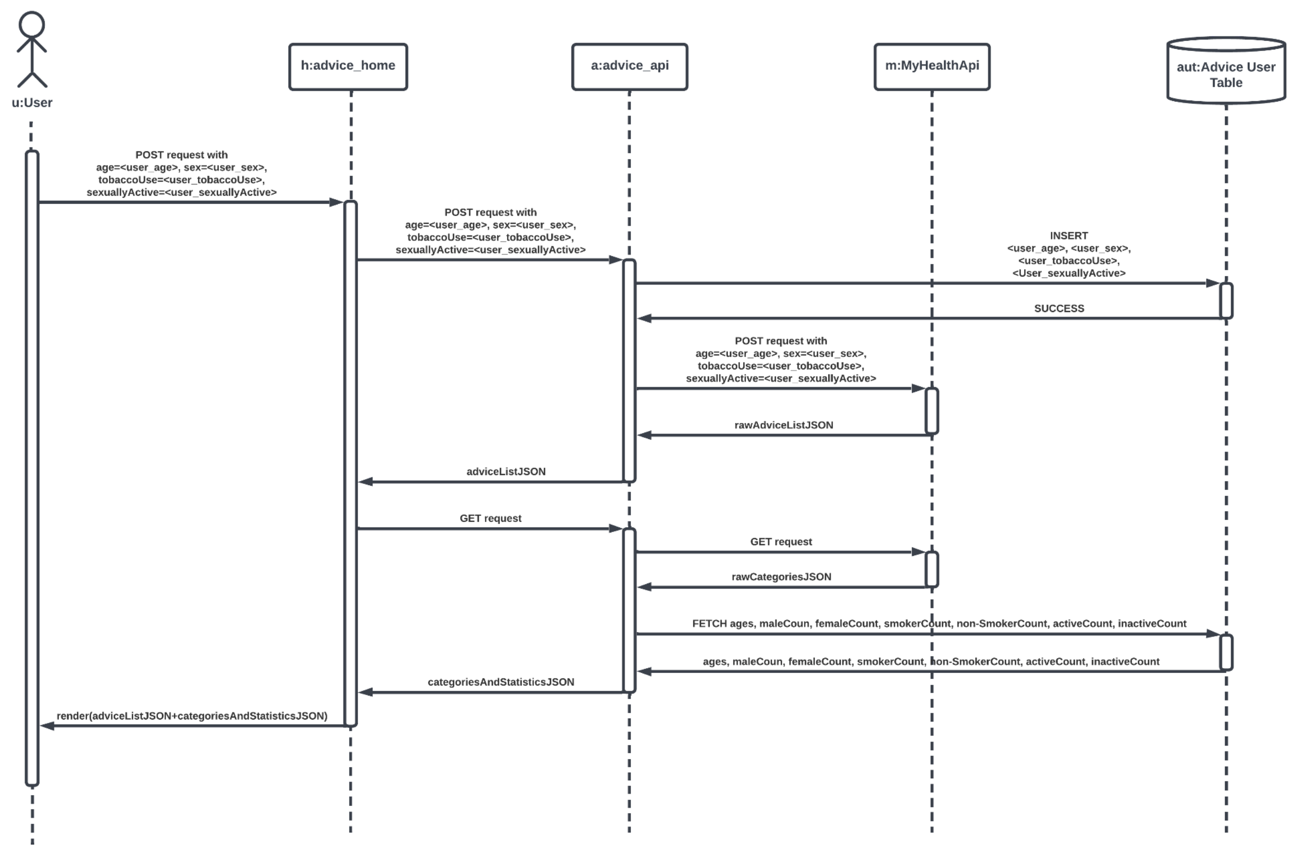

UserModel
-userId: int
-username: TextInput
-password:PasswordInput
-mail:EmailInput
-horoscope: string
UserAppApi
+userList: list<UserModel>
+userDetail(userID:int): <User,dailyHoroscope:string>
+userCreate(username:string,password:string,mail:string): boolean
CommentModel
-text: string
-pub_date: datetime
-upvotes: int
-downvotes: int
-isMarkedNSFW: bool
-author: User
-parent: Post
PostModel
-postID: int
-title: string
-description: string
-post_type_label: string
CommentAppApi
+index(): list<CommentModel>
+insert(): <CommentModel>
-DetailView(): django.shortcuts.generic.DetailView
PostAppApi
+postIndex(): weatherForecast:float
+getPost(postID:int): <Post>
+createPost(title:string,description:string,post_type_label:string): boolean
CountriesModel
-country: string
StatesModel
-state: string
-country: CountriesModel
CitiesModel
-city: string
-state: StatesModel
UserLocationModel
-username: string
-country: CountriesModel
-state: StatesModel
-city: CitiesModel
LocationMgrApi
+Location.get(): <loc: list<string>>
+Location.post(): <info: string>
+Info.get(): list<string>
+Info.post(): <userdata: UserLocationModel, near: list<string>>
NewsModel
-News_title: string
-News_description: string
-News_image: binary[]
-News_url: string
NewsAppApi
+news_without_query: list<NewsModel>
+news_with_query: list<NewsModel>
ArticleModel
-Article_title: string
-Article_summary: string
-Article_authors: string
-Article_link: string
ArticleAppApi
+list(): < ArticleModel >
+search(subject : string, numberOfArticles : integer): < ArticleModel >
+deleteHistory(): boolean
Drugs Model
-drug_name: String
-side_effects: Strring
Drugs and Side Effects App API
+GetDrugSideEffects(): <drugs: list<string>>
+CreateDrugSideEffects(): <response: string>
+SearchMedicine(): <response: JSON>
AdviceUser
+ age: int
+ sex: string
+ tobaccoUse: boolean
+ sexuallyActive: boolean
fetches from database
inserts to database
advice_home
+ renderer_classes: list<Renderer>
+ template_name: string
+ get(request: request): HTMLResponse
+ post(request: request): HTMLResponse
advice_api
+ get(request: request): JSONResponse
+ post(request: request): JSONResponse
CoronavirusApi
+ country_name: string
coronavirus_home
+ combined_keys: list<string>
+ get(request: request): HTMLResponse
+ post(request: request): HTMLResponse
coronavirus_api
+ get(request: request): JSONResponse
+ post(request: request): JSONResponse

Sequence Diagrams

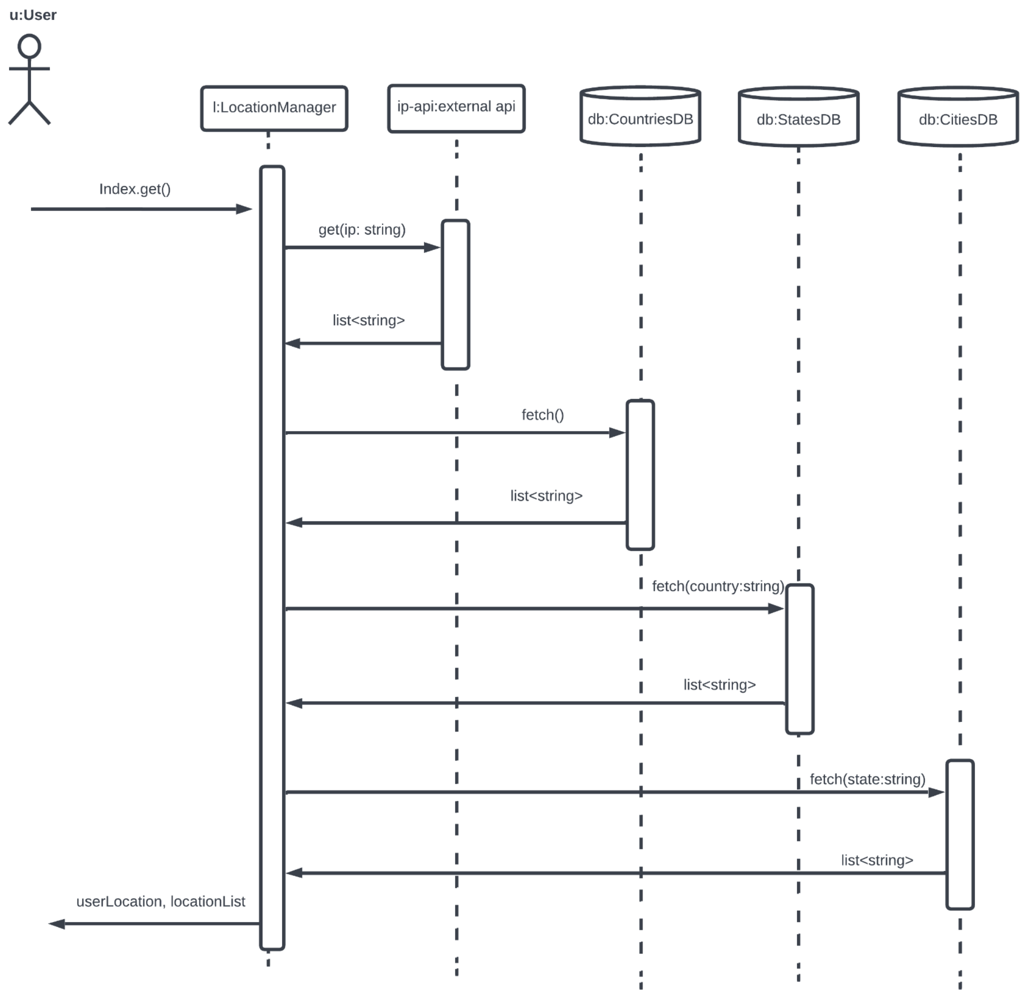
User App Diagram



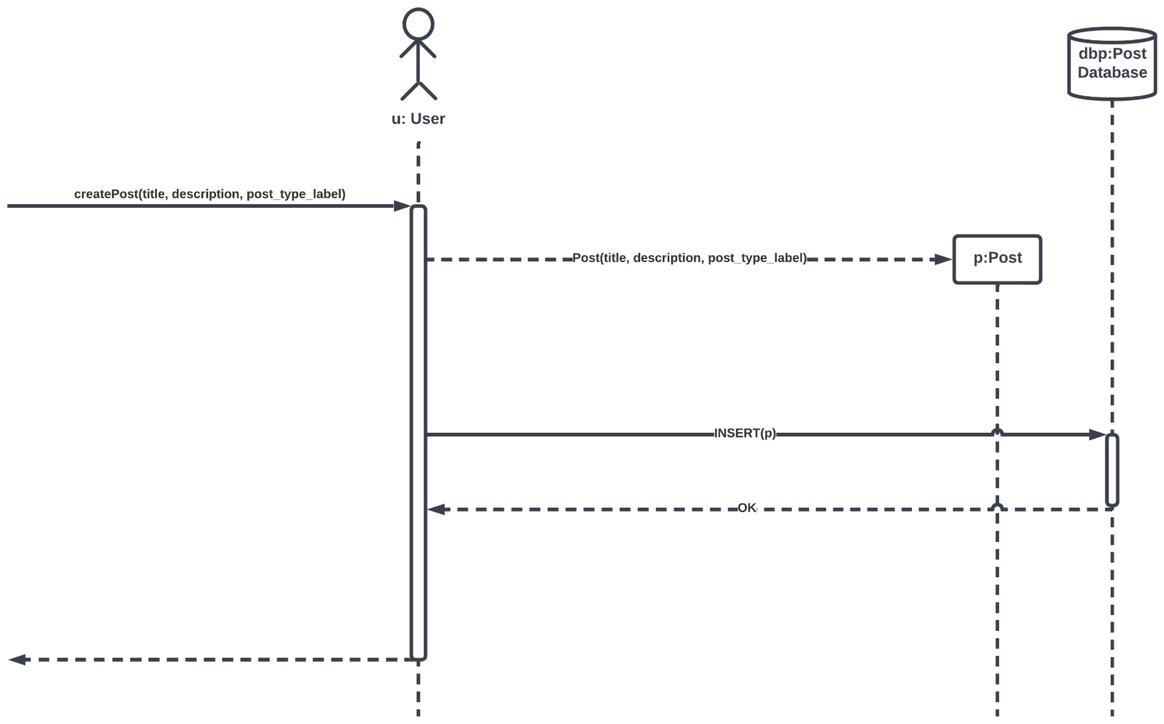
Advice App Diagram


u:User
h:advice_home
a:advice_api
m:MyHealthApi
aut:Advice User Table
POST request with age=<user_age>, sex=<user_sex>, tobaccoUse=<user_tobaccoUse>, sexuallyActive=<user_sexuallyActive>
POST request with age=<user_age>, sex=<user_sex>, tobaccoUse=<user_tobaccoUse>, sexuallyActive=<user_sexuallyActive>
INSERT <user_age>, <user_sex>, <user_tobaccoUse>, <User_sexuallyActive>
SUCCESS
POST request with age=<user_age>, sex=<user_sex>, tobaccoUse=<user_tobaccoUse>, sexuallyActive=<user_sexuallyActive>
rawAdviceListJSON
adviceListJSON
GET request
GET request
rawCategoriesJSON
FETCH ages, maleCoun, femaleCount, smokerCount, non-SmokerCount, activeCount, inactiveCount
ages, maleCoun, femaleCount, smokerCount, non-SmokerCount, activeCount, inactiveCount
categoriesAndStatisticsJSON
render(adviceListJSON+categoriesAndStatisticsJSON)

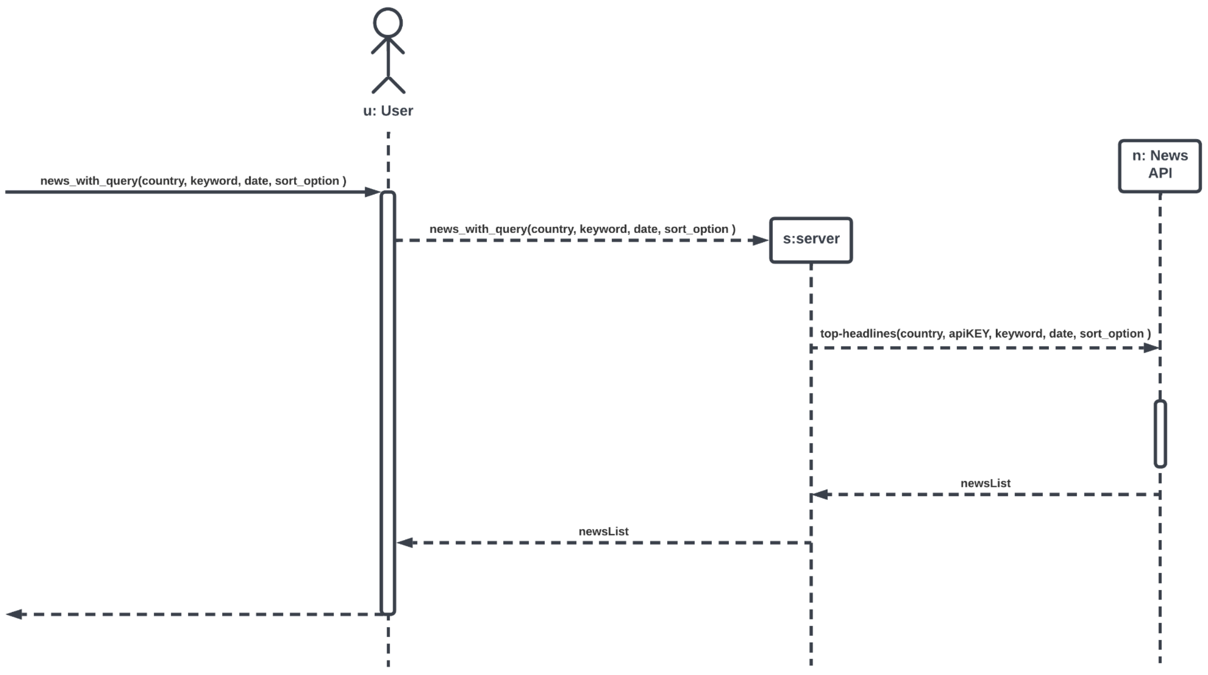
Location Manager App Diagram



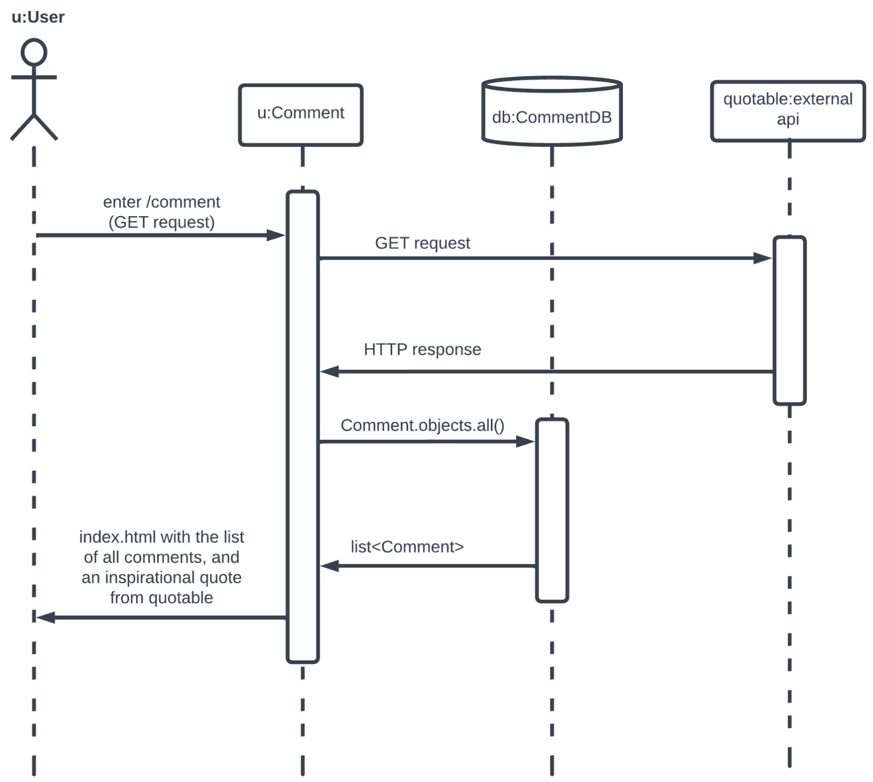
Post App Diagram


u: User
dbp:Post Database
createPost(title, description, post_type_label)
Post(title, description, post_type_label)
p:Post
INSERT(p)
OK

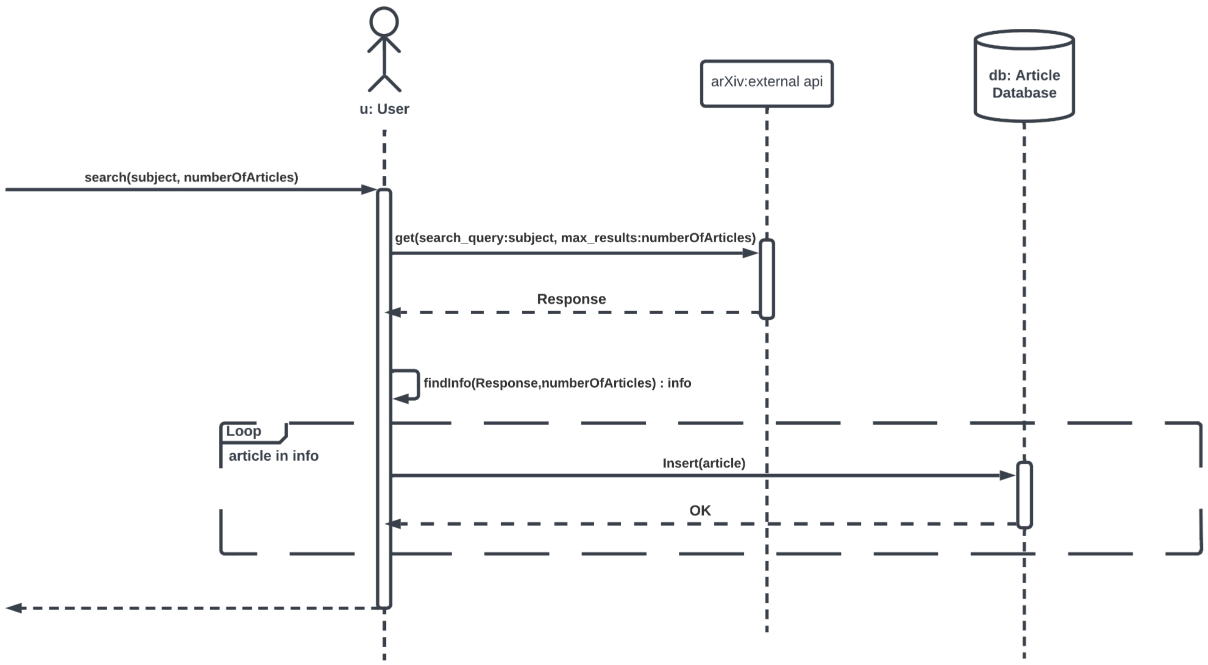
News App Diagram


u: User
n: News API
news_with_query(country, keyword, date, sort_option)
news_with_query(country, keyword, date, sort_option)
s:server
top-headlines(country, apiKEY, keyword, date, sort_option)
newsList
newsList

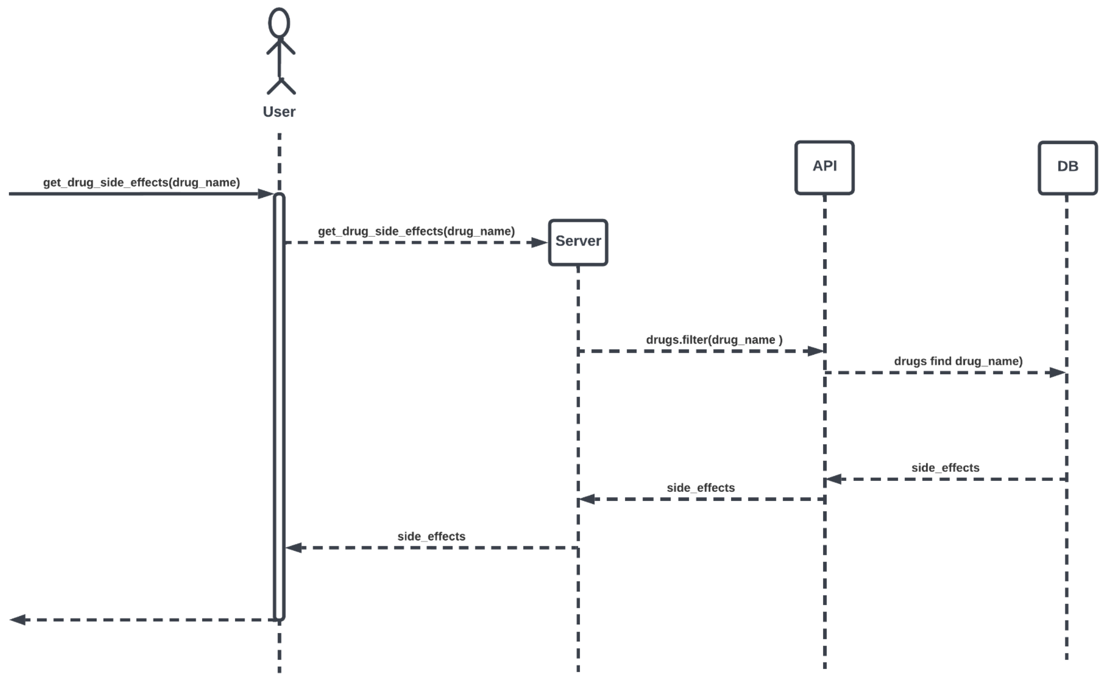
Comment App Diagram


u:User
u:Comment
db:CommentDB
quotable:external api
enter /comment (GET request)
GET request
HTTP response
Comment.objects.all()
list<Comment>
index.html with the list of all comments, and an inspirational quote from quotable

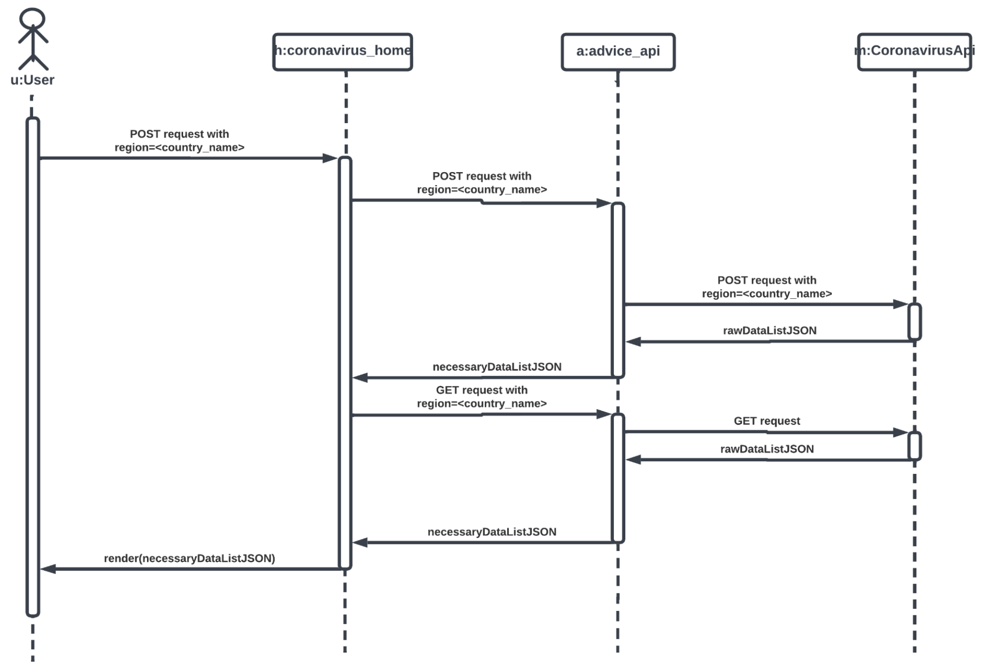
Article App Diagram


u: User
arXiv:external api
db: Article Database
search(subject, numberOfArticles)
get(search_query:subject, max_results:numberOfArticles)
Response
findInfo(Response,numberOfArticles) : info
Loop
article in info
Insert(article)
OK

Drug and Side Effect App Diagram



Coronavirus App Diagram


u:User
h:coronavirus_home
a:advice_api
m:CoronavirusApi
POST request with region=<country_name>
POST request with region=<country_name>
POST request with region=<country_name>
rawDataListJSON
necessaryDataListJSON
GET request with region=<country_name>
GET request
rawDataListJSON
necessaryDataListJSON
render(necessaryDataListJSON)

URI Tag of The Application

You can view the tag of the project here.

Code

You can view the code here.

Running Practice App

Please run all commands while you are in the outer practice_app folder

To run the application on local computer:
Run the following commands on the terminal:

- ./init.sh
- python manage.py runserver

Application will be accessible on http://127.0.0.1:8000/

To run the application on docker container:
Run following commands on the terminal:

- DOCKER_BUILDKIT=1 docker build . --tag practice-app
- docker run -p 80:8000 practice-app

Application will be accessible on http://127.0.0.1:8000/