CityC

Bozhidar Sabahov

Student Number: 17703831

Salvijus Mazeikis

Student Number: 17302181

07.05.2020

Technical Guide

1. Motivation

The motivation behind this project is that we have often found ourselves struggling to decide what to do in a big city as Dublin where a huge variety of places are available to visit. The goal was to develop a progressive web-app which would be available for browsers but also for Android and iphone. The application will generate a tailored plan of activities for the user. The activities will fit in the user's available time and it will take in consideration the location, budget, and type of transport. In addition, these activities will match the user interests but some of them will be completely new for the user. This will allow the customer to explore and try something completely different which they would like. The user will input their favourite categories at registration hence it will be possible to look for similar users or just search for activities within the liked categories, which would help if we come across the cold start problem at recommenders.

2. Research

2.1. Recommendation algorithm research

2.1.1. What is Yelp and how it is used within the project?

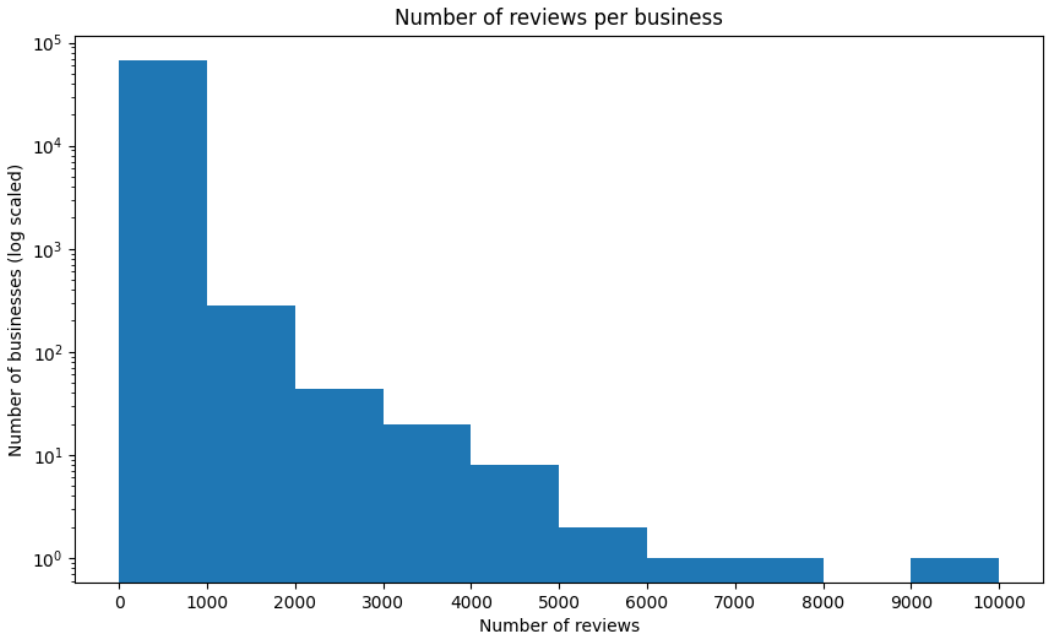
Yelp is a popular web-site for discovering local businesses ranging from restaurants, bars, cafes to spas, health and medical service and many other diverse types of activities. They have a public API which allows access to their data of businesses around the world. In addition, Yelp provides a subset of their businesses, reviews and user data in JSON format.

The recommender system is based on the data from Yelp and their API. Essentially, it is split in two parts.

2.1.2. Understanding the Yelp dataset

The Yelp dataset contains a variety of businesses that match our requirements. In addition, they provide an offline dataset which can be used to analyse their data. Furthermore, the business's data on Yelp contains a number of attributes such as: categories, locations, coordinates, rating, price and others which can be utilized, when recommendations or plans are generated, in order to fine-grain the algorithms to match each user preferences. Finally, Yelp provides endpoints through which their whole database can be queried. As a result, that allows to search for businesses and provides a possible way to deal with the cold start problem in recommenders.

Before choosing a recommendation algorithm, the first step is to understand the dataset on which the algorithm will be based. For the purpose of the project, the Yelp Academic dataset of businesses was used. It contains businesses from 37 states from the USA. The main features which were identified as useful to the project were the rating, categories and reviews of the businesses. The next step was to filter the dataset by categories and remove unnecessary attributes. For example, some of the parent categories did not meet the requirements of the project, such as: ‘Education’, ‘Pets’, ‘Local Services’, ‘Bicycles’, etc. Finally, the reviews were analysed.



Unfortunately, the result of analysis showed that the data is left skewed. The median is 19 which meant that most of the businesses do not have enough reviews to be able to extract some information from them. The observations concluded that the recommendation algorithm could be based on the categories and ratings only. Moreover, Yelp provides enough information such as location and open hours to be able to plan a set of activities for the user. In addition, Yelp API grants access to the latest information of all businesses in their database.

The recommendations system can be content-based or collaborative. To make the decision which one would fit Yelp’s dataset, a better understanding of them was required.

2.1.3. Content-based filtering

Content-based filtering is the recommendation of items similar to what the user has already liked. The recommendation is based on the similarity of the content of the available items to the content of the items the user has liked in the past. This is often transformed into a classification or regression problem. The recommender can be user-centered or item-centered.

Advantages

- Scalable because the model doesn't require data of other users.
- The model can capture specific interests of the user.
- A possible solution to the cold start problem of Collaborative filtering.

Disadvantages

- Features have to be extracted from the dataset (category, ratings, price, etc). The same could be done for the users (age, sex, etc). The model will be as good as the features.
- Users will not be able to discover new activities outside their interests.

2.1.4. Collaborative filtering

This method is often used in recommendation systems. Just like a machine learning algorithm it makes a prediction based on the user's past behaviour. In other words, it predicts user preferences for a set of items based on experience. It is referred to as a model based algorithm.

Advantages

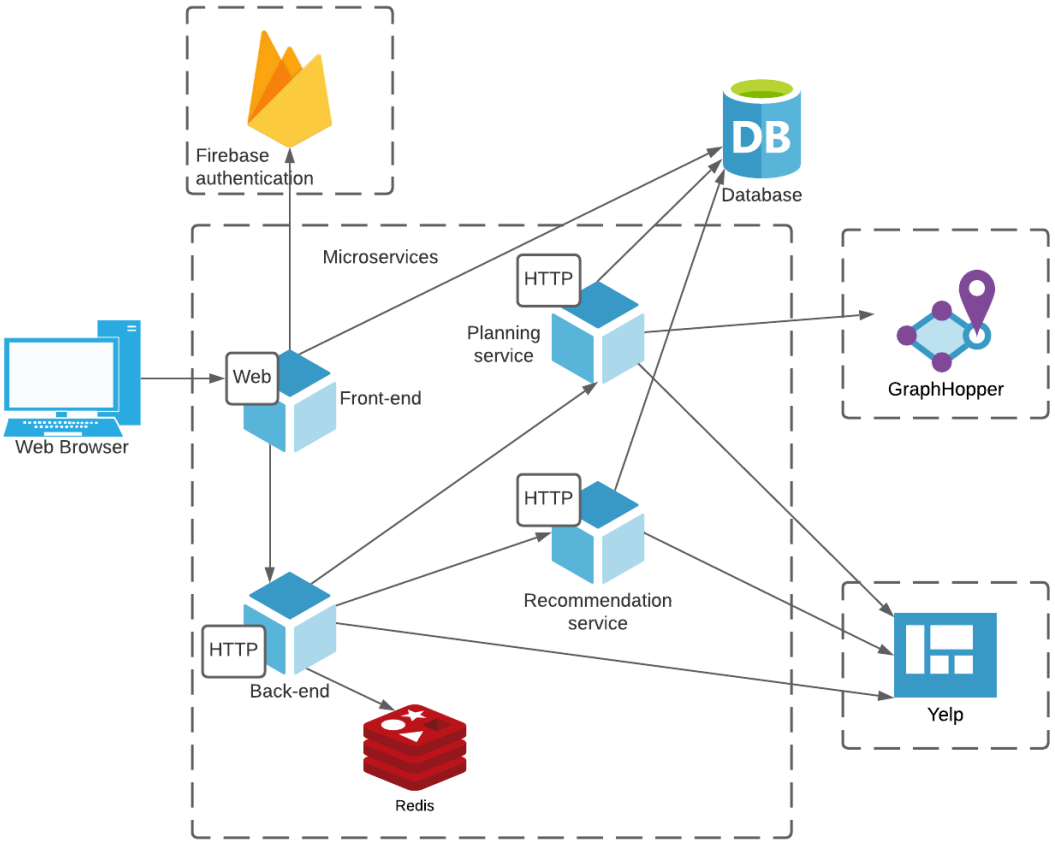
- Users can discover new activities because the model makes recommendations by looking at other users interests.
- Information about the user or the activities is not necessary. The features are not hand-engineered like in content-based filtering but they are automatically learned.

Disadvantages

- Suffers from cold start problems.
- Depends on the algorithm (Nearest neighborhood / Matrix factorization).

3. Design

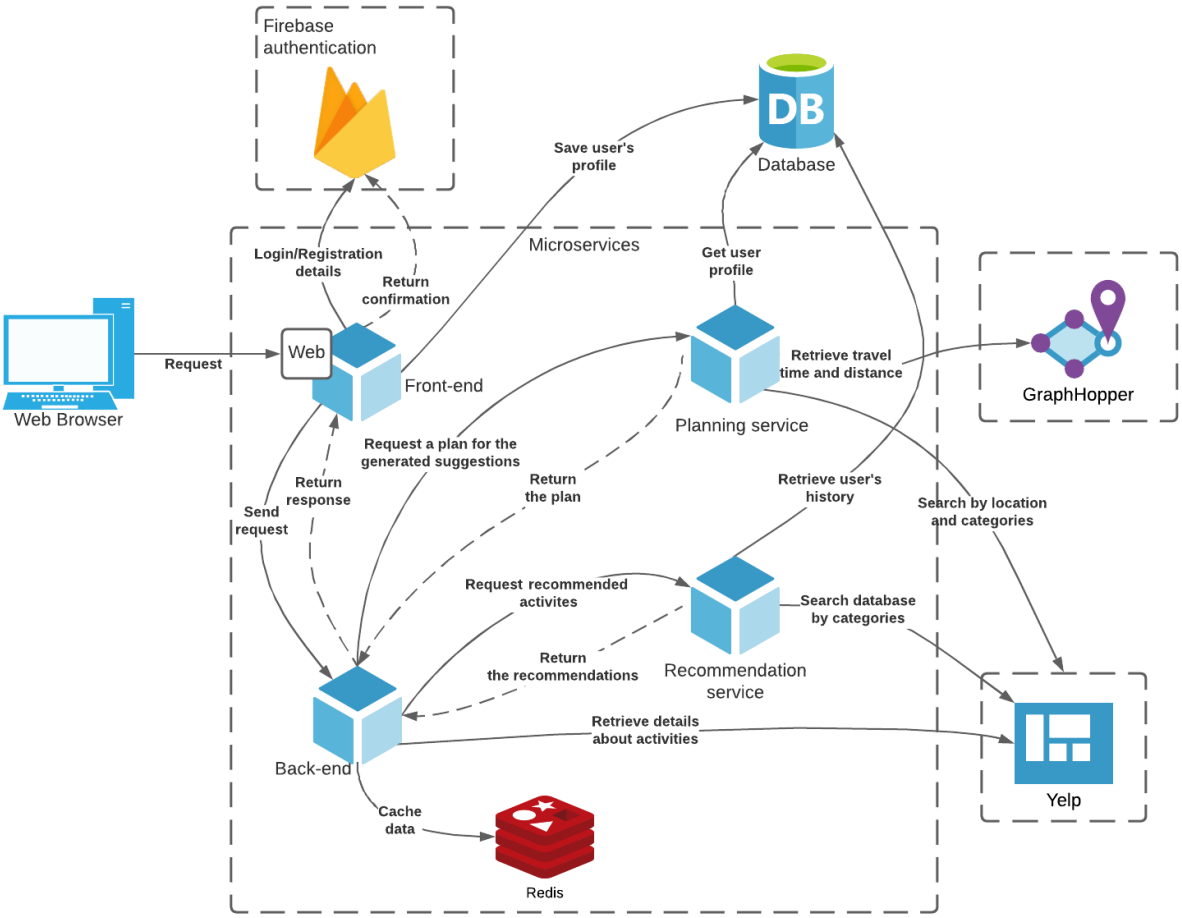
3.1. System architecture



The whole system is divided into microservices. Each microservice runs in a container and has its own responsibility. The back-end microservices handles the communication between the other services. Recommendation service generates suggestions and the planning service makes a plan.

Redis is run in a container, too. It allows the system to cache the user recommendations for 24 hours for consequent requests. Meanwhile, GraphHopper calculates the walking distance and time between activities. Finally, the Yelp database is queried for details of activities and searched for activities.

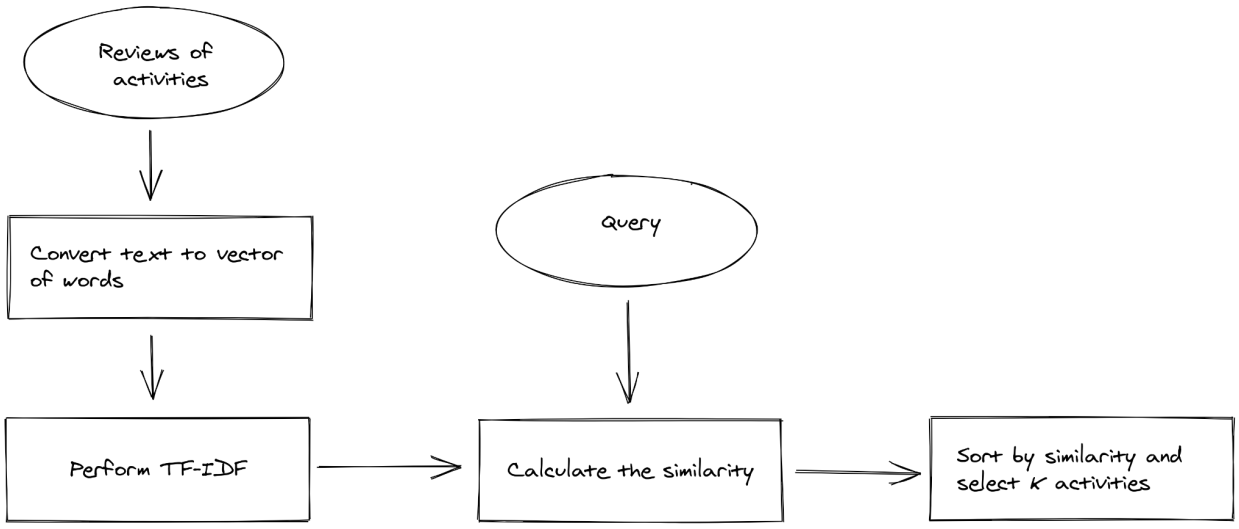
3.2. High-level design of the system



The diagram depicts the data sent between the microservices and the third party APIs.

3.3. Recommendation system

3.3.1. Initial design - Content based filtering with cosine-similarity



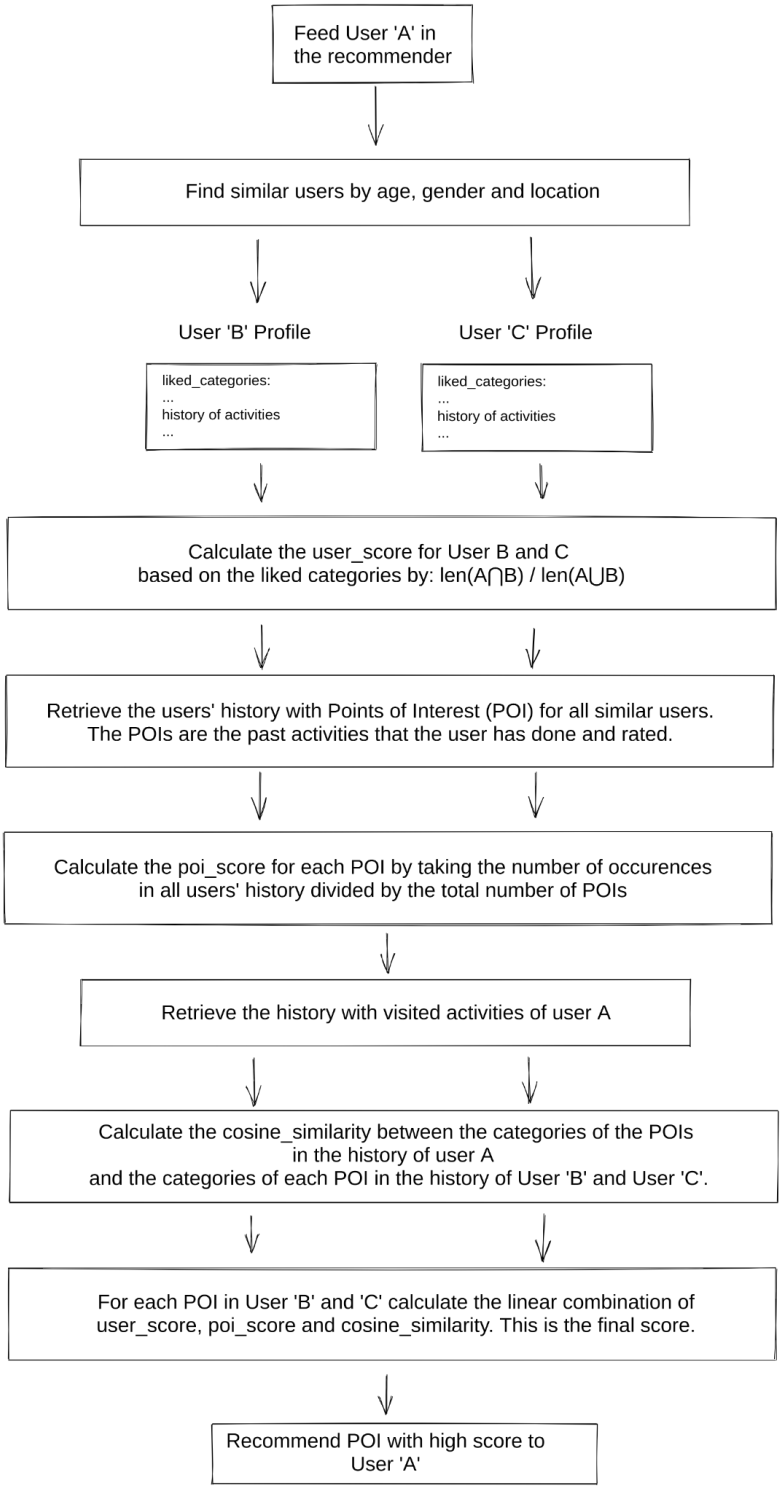
The idea of the initial recommender design was to collect the reviews of activities and perform information retrieval on the reviews, in order to extract commonly occurring words which describe the activity. For example, it could be adjectives such as expensive, fancy, cheap, dirty, etc. Then the query would be a sequence of words, for instance: requirements, interests, dislikes from the user. However, as it was mentioned in the research section, the businesses on Yelp do not have enough reviews to perform term frequency-inverse document frequency statistics.

3.3.2. Final design of the system

The final recommendation system is split in two parts. The first part is the recommendation algorithm which in parts is based on the Yelp dataset. The algorithm is hybrid and it combines the cosine similarity between the categories of the POIs and the similarity among groups of users. It depends on a large amount of a user base to work properly (diagram below). Therefore it requires a second part which deals with the problem by directly requesting businesses (activities) from the Yelp API. The Yelp API is queried for businesses that match the user's liked categories. This allows the system to collect enough data of the user's history for the recommendation algorithm to work.

3.3.2. Design of hybrid algorithm

For the algorithm to function properly, the user must be registered and they should have a history of visited activities. If they don't have history they are a new user and their suggested activities will be retrieved from Yelp, by searching by their liked categories.



3.3.3. Sample code of calculating the final score

```
def calculate_user_score(user, related_user):
    user_categories = set(user['liked_categories'])
    related_user = related_user.to_dict()
    related_user_categories = set(related_user['liked_categories'])

    intersection = related_user_categories.intersection(user_categories)
    union = related_user_categories.union(user_categories)

    user_score = len(intersection) / len(union)
    return user_score
```

The function above calculates the user score based on the liked categories of a similar user.

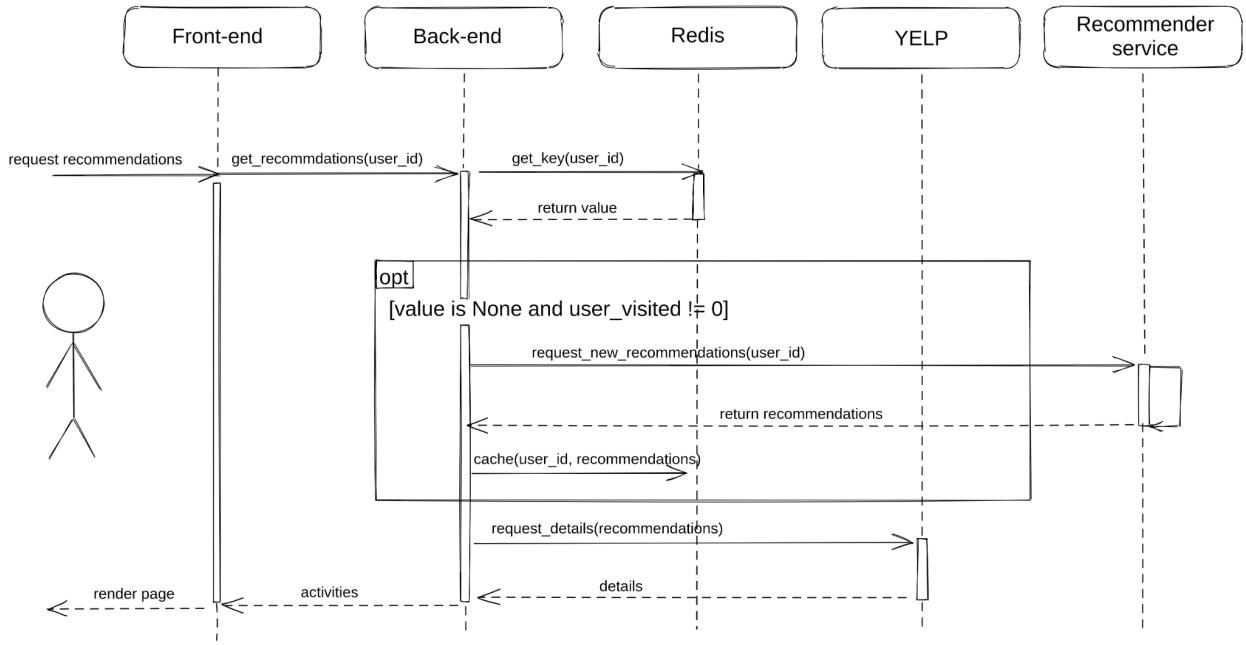
```
37      cv = CountVectorizer()
38      pois_categories = user_history
39      for poi_id in potential_reccomms.keys():
40          pois_categories.append(potential_reccomms[poi_id]['categories'])
41
42          count_matrix = cv.fit_transform(pois_categories)
43          cosine_values = cosine_similarity(count_matrix[-1], count_matrix[:-1])
44
45          user_score = potential_reccomms[poi_id]['user_score']
46          poi_score = (
47              potential_reccomms[poi_id]['poi_score']
48              / len(potential_reccomms.keys()))
49          cosine_sim = cosine_values.max()
50          score = 0.5 * cosine_sim + 0.3 * user_score + 0.2 * poi_score
51
52          potential_reccomms[poi_id]['final_score'] = score
53          pois_categories.pop()
```

The sample code shows how the recommender calculates the score of each potential recommendation for an user. It takes the categories from the history of visited activities of the input user and for each potential recommendation calculates the cosine similarity. Then the final score is a linear combination of the user score, poi score and cosine similarity. The cosine similarity has the most impact on the final score because it is the main metric. The user score is a valuable metric since it shows which users are similar in terms of likes and dislikes.

3.3.4. Request to the hybrid algorithm

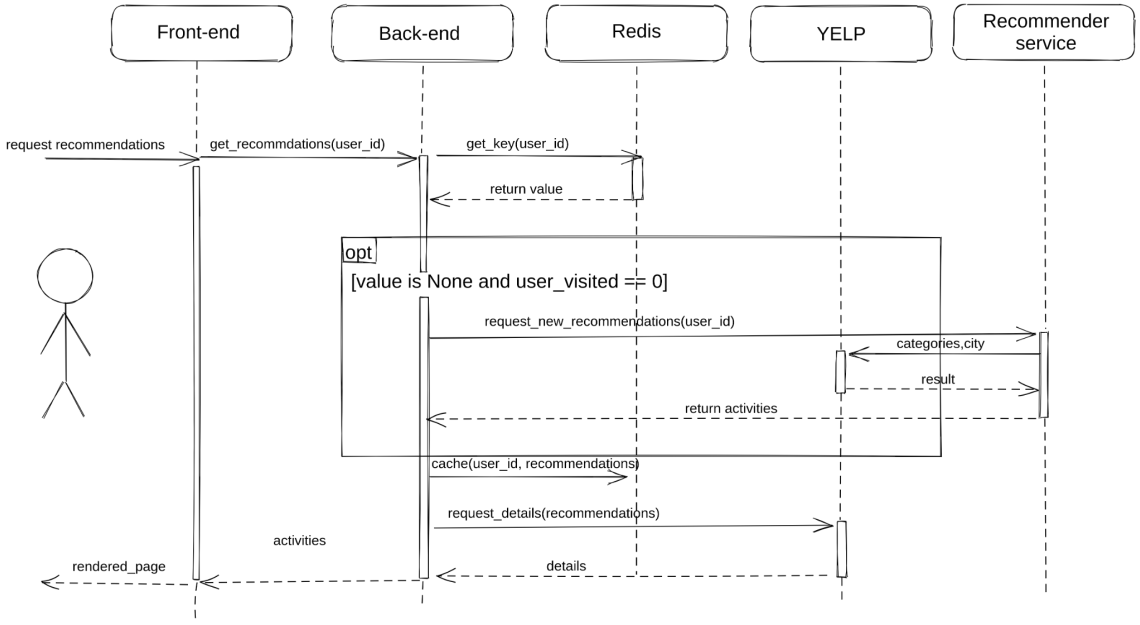
After the recommender returns the suggestions, they are cached. Therefore, the hybrid algorithm is activated only when the cache is empty and the user has more than 0 visited activities (as the diagram above shows). When the user has not visited any activities, then the cosine-similarity based algorithm is not called but the database of Yelp is searched by the liked

categories of the user and city. The diagram at 3.3.5 explains that below. This case helps to deal with the cold start problem.



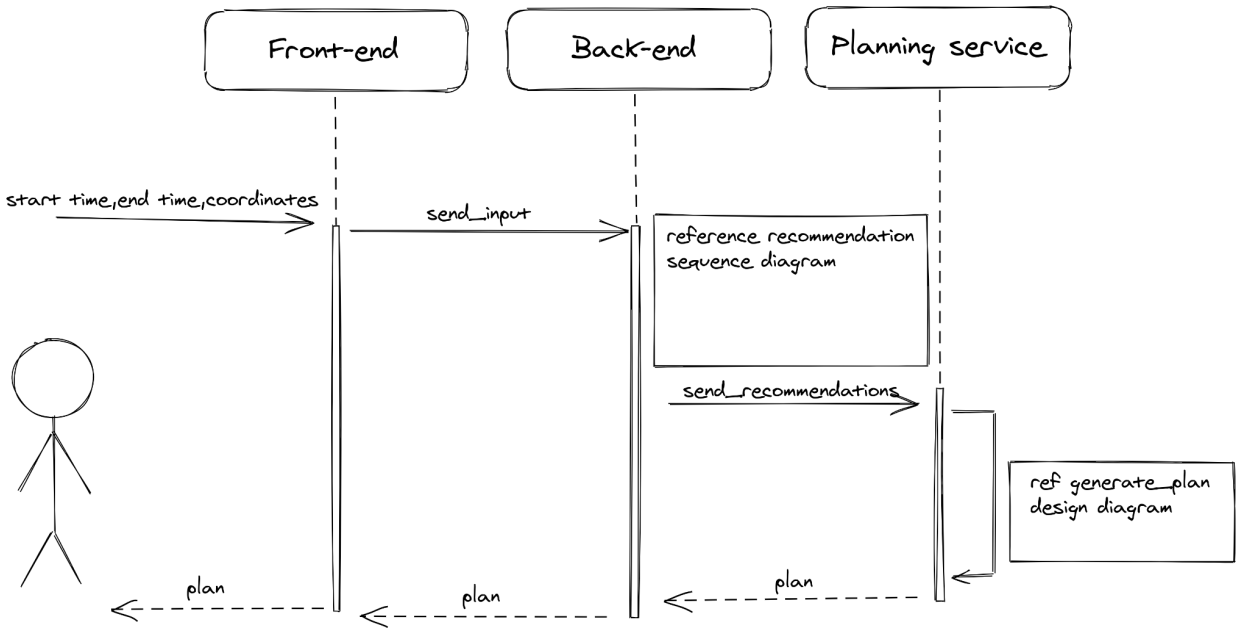
3.3.5. Requesting businesses from Yelp

Here the recommender service requests businesses from Yelp by sending the categories and the city of the user. The design can be improved more so that only one request is made to Yelp for this sequence.



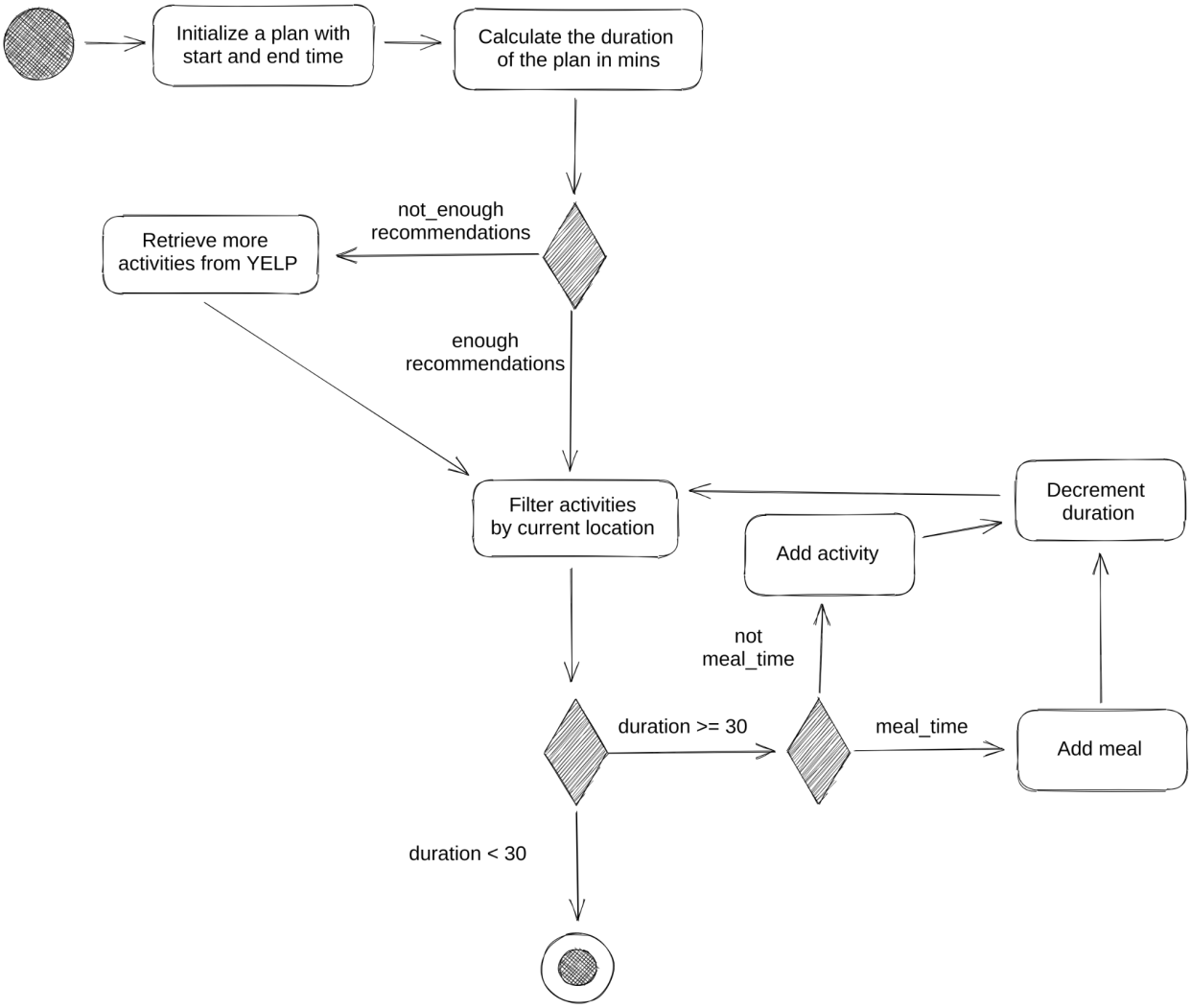
3.5. Planning service

The sequence diagram explains the communication between the microservices when a request for a plan is received. The back-end retrieves the recommendations before calling the planning service. The diagram references the recommendation sequence diagram at .



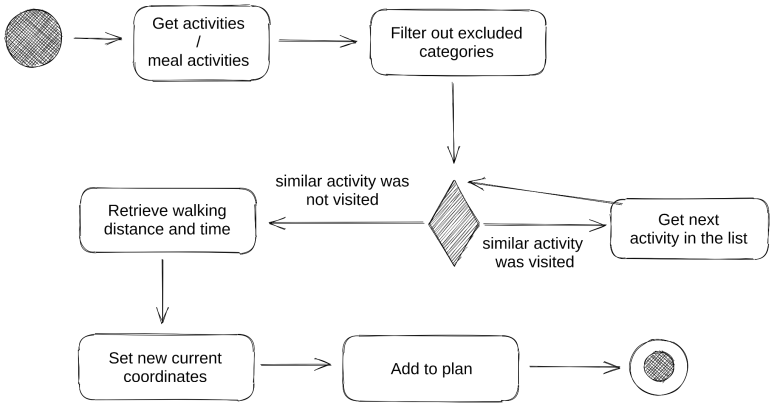
3.5.1. Generating a plan

This diagram shows how the planning service generates a plan. The recommendations and other parameters are passed from the back-end to the planning-service.



3.5.2. Adding activity to a plan

The diagram depicts how an activity or meal is appended to the plan. As well as, how the algorithm considers which activity to add to the plan.



3.5.3. Sample code of 'Adding activity to a plan'

```
68      def add_activity(self):
69          if not self.activity_keys:
70              return False
71
72          if self.timer.get_current_time() < datetime.strptime('18:00', '%H:%M'):
73              excluded_categories = ['Bars', 'Nightlife']
74          else:
75              excluded_categories = ['Active Life']
76
77          for key in self.activity_keys:
78              activity = self.activities[key]
79              exclude = False
80              for parent in activity['parents']:
81                  if parent in excluded_categories:
82                      exclude = True
83              categories = set(self.activities[key]['categories'])
84              visited_score = (
85                  len(self.visited.intersection(categories))/len(categories)
86              )
87              if visited_score >= 0.5:
88                  exclude = True
89              if exclude is False:
90                  self.activity_keys.remove(key)
91                  self.add_to_plan(key)
92                  return True
93
```

The piece of code above demonstrates how an activity is added to the plan. It returns False when there are no activities in the vicinity to choose from and True when it successfully has added an activity to the plan. The code considers whether it is day time or night at the "current time" in the plan (line 72). If it is before 18:00 it excludes any bars, if it is after 18:00 it excludes any "Active Life" categories. Furthermore, it determines whether a similar activity is already in the plan by calculating a score at line 85.

4. Implementation

4.1. Microservices

The system is divided into 4 components - front-end, back-end, recommendation service and planning service. The main service in the system which manages the incoming requests from the front-end is the back-end. It distributes the requests to the recommender or planner and sends back the results to the front-end. The recommendation and planning service do not know

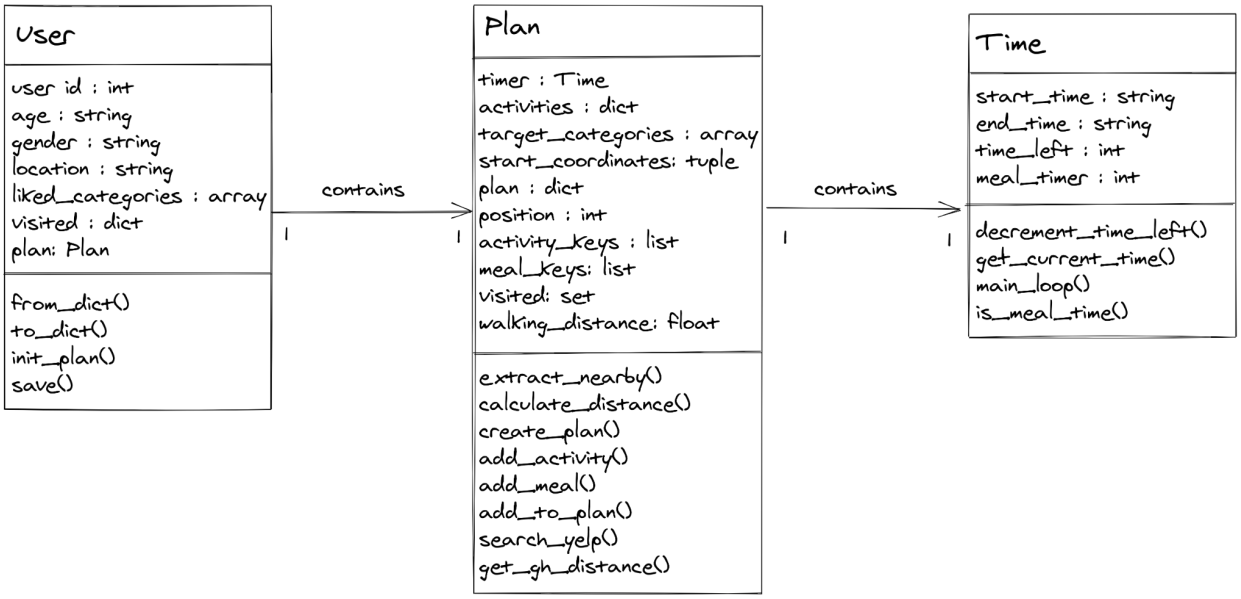
about each other and they work independently. Each service is a small Flask application run in a Docker container. The file structure of the applications are the same.

The system is initialized by docker-compose which creates multiple isolated environments on a single host which share one virtual network, where the services can communicate between each other. The front-end and the back-end have exposed ports to the public internet. Meanwhile, the recommonder and planner can be accessed only from the virtual network created by docker-compose.

4.2. Recommender

The recommender is implemented in Python with the help of SciKit. The user_score and poi_score of the POIs is calculated using only Python syntax since the calculations are simple. (refer to the diagram at 3.3.2) Meanwhile, SciKit is utilized to get the cosine similarly between POIs. To be able to calculate the similarity each POI has to be vectorized. Since it is being dealt with categorical values to vectorize them, they have to be one-hot encoded. The vectors will have the same size as the number of categories in the current vocabulary. In the implementation, the vocabulary is the categories of the POIs of the input user (the user passed to the recommender) and the category of the POI for which the similarity is calculated for. The size of the vocabulary depends on the number of different categories in the input user's history. For example, if the input user has visited 'Fishing, Bowling, Bars' and the similarity to POI with category 'Museum', then the vocabulary will be 'Museum, Fishing, Bowling, Bars'. The similarity is calculated between the POIs of the input user and each POIs of the similar users.

4.3. Planner



The planner is implemented in an object oriented approach where one user can have one Plan and Timer. The timer is responsible for keeping track of the current time left in the Plan. Once there is no time left the plan has reached its conclusion. Meanwhile, the Plan has access to the recommended activities of the User and its duty is to add activities to the plan by taking in consideration the walking distance to the activity and whether similar activity already exists in the plan. The Vincenty formula, which can measure the distance between two points on the surface of a spheroid, is used to calculate the walking distance to an activity. Moreover, the Plan can retrieve more activities from Yelp based on the liked_categories from the User whenever the plan runs out of activities to put in. These activities will be within “walking distance” which is calculated by the formula - `d/(200 + 2/d)` where ‘d’ is the duration of the plan in minutes. Before an activity is added to plan, the walking time is retrieved from GraphHopper which is an API that provides route planning.

6. Problems solved

6.1 Recommendations

One of the main features of the application is generating new recommendations for the users. The requirements are to give suggestions which the user would like based on their history, as well as new recommendations which the user might be interested in but have never visited before. The initial idea was to train a machine learning algorithm that is able to predict a type of activity based on a user's hobbies or interests. However, it turned out there is no such dataset to train the algorithm on. This is the reason why the Yelp dataset comes into play but it lacked useful features that uniquely describe an activity. Therefore, it was necessary to improvise with the only useful feature - the categories. Yelp has hundreds of different categories. The cosine similarity algorithm helps to find similar activities to their history. Meanwhile, the user score in the algorithm finds activities which the user has not experienced before but someone else with a similar profile has. Finally, the cosine similarity suggests new recommendations similar to the user's past activities.

6.2 Planning service

The next important feature in the application is the generation of a plan with activities for the user. To develop an algorithm that builds a well tailored plan to the customer needs, a number of factors have to be considered. Some of them are not known when the user is initially registered. For example, how much time they spend at a certain type of activity or in what sequence they would want like the activities in the plan. It would require a long time and a lot of data to train an algorithm that will adapt to each individual user. In the developed algorithm the sequence of

activities are fixed - 4 activities followed by a meal. Meanwhile, the duration of each activity is fixed to 45 mins and each meal to 50 mins. The current algorithm achieves the most important initial requirements and that is the generation of plan which fits in the user's timeframe and location while considering the time to get to the activity. As well as, it ensures that the activities are within walking distance and there are no similar activities in the plan.

6.3 Yelp API

Yelp provides endpoints that can be used to query their data with GraphQL which is a query language. However, they do not offer any Python package, hence GraphQL package for building queries had to be used, which is still in early development and provides only basic functionalities. For example, the query is built from a string. It is easy to use if only requests for a single resource are sent. Therefore, a string builder was created that takes the ids of the businesses and creates a query which requests their details.

8. Future work

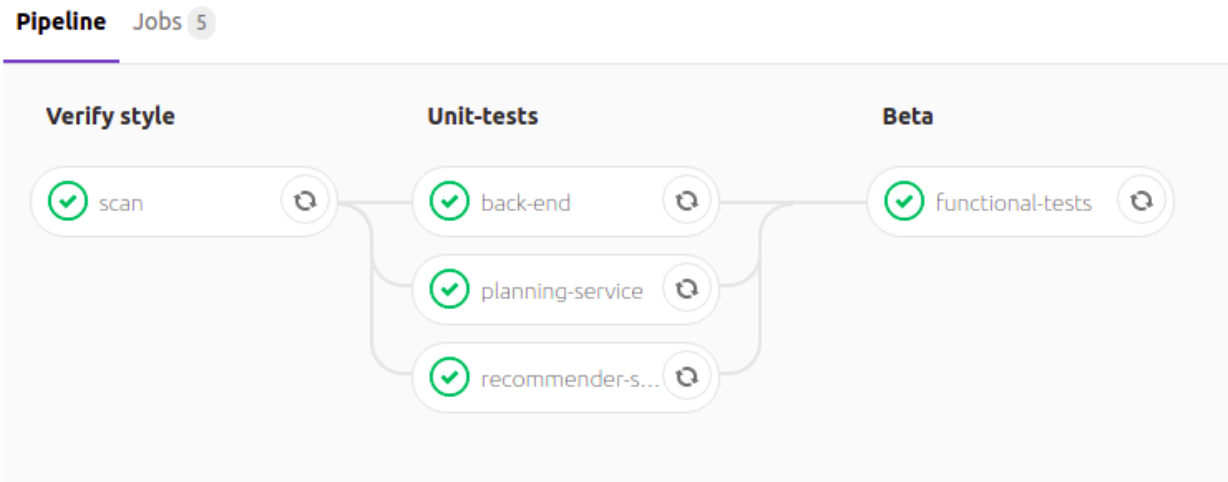
The next steps are to optimize and refactor the planning algorithm furthermore. Meanwhile, the front-end should be styled and made user-friendly. Then, the system can be deployed in alpha testing, where it will be evaluated by users. The result will determine whether the recommender, planning service and the front-end are well developed. As well as, it will give more information about the target user base which could be used to optimize and improve the algorithms.

The recommendation algorithm uses collaborative filtering and at the moment the generated suggestions, to new users, are activities which match the user liked categories. A better approach is to develop a content-based recommendation algorithm which would solve the cold start problem, too.

9. Testing

The Continuous Integration pipeline on GitLab is divided in three stages. In the first stage, the style of the code is scanned and verified that the code meets the basic coding standards. In the second stage the unit tests of each microservice are executed. Then finally in the last stage, functional tests are run to ensure that the microservices communicate successfully between

each other.



9.1. Unit testing

Each micro service has unit tests which can be run locally without the need of setting up a docker container.

9.1.1 Routes testing

Each route in the microservices should return appropriate error messages if no or wrong arguments are passed. They should return them without revealing any code that runs in the back-end. Therefore, custom error messages were created and they are tested.

9.1.2 Yelp API

Unit tests were created for the query builder which sends requests to the Yelp API. The Yelp API is in early development and it is good to have tests that ensure there is nothing new in API.

9.1.3 User, Plan, Time unit tests

To ensure the planning algorithms work as expected when new changes or refactoring is done. Unit tests for each class were implemented too. They run alongside the unit tests for routes and Yelp API.

9.1.4 Unit tests output

```
(.venv) teflon@teflon:~/FourthYear/2021-ca400-sabahob2-mazeiks2/src/planning-service$ pytest tests/unit/
============================= test session starts ==============================
platform linux -- Python 3.7.9, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/teflon/FourthYear/2021-ca400-sabahob2-mazeiks2/src/planning-service
plugins: flask-1.2.0
collected 13 items

tests/unit/test_plan.py ...
tests/unit/test_time.py .....
tests/unit/test_user.py ...
tests/unit/test_view.py .
tests/unit/test_yelp_api.py .

============================== 13 passed in 5.97s ==============================
```

```
(.venv) teflon@teflon:~/FourthYear/2021-ca400-sabahob2-mazeiks2/src/back-end$ pytest tests/unit/
============================= test session starts ==============================
platform linux -- Python 3.7.9, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/teflon/FourthYear/2021-ca400-sabahob2-mazeiks2/src/back-end
plugins: flask-1.2.0
collected 5 items

tests/unit/test_views.py ..
tests/unit/test_yelp_api.py ...

============================== 5 passed in 1.50s ===============================
```

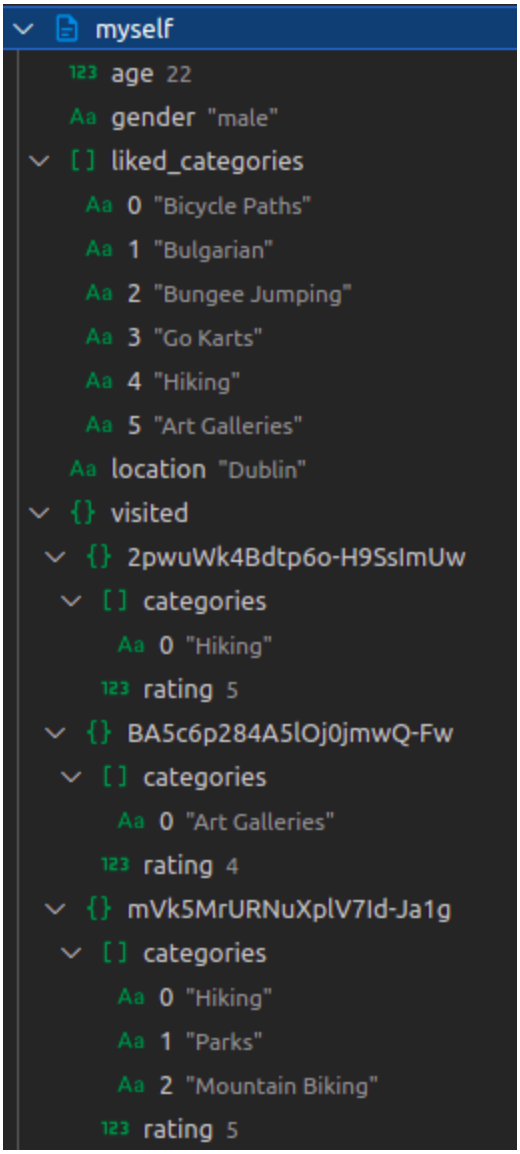
```
(.venv) teflon@teflon:~/FourthYear/2021-ca400-sabahob2-mazeiks2/src/recommender-service$ pytest tests/unit/
============================= test session starts ==============================
platform linux -- Python 3.7.9, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/teflon/FourthYear/2021-ca400-sabahob2-mazeiks2/src/recommender-service
plugins: flask-1.2.0
collected 3 items

tests/unit/test_views.py .
tests/unit/test_yelp_api.py ..

============================== 3 passed in 3.41s ===============================
```

9.2. Functional testing

To verify that the functionality of the planning and recommendations microservices meets the requirements, functional tests need to be implemented. First a environment for containers is built with docker-compose, then the back-end waits for the recommendation and planning service to start. Finally, the back-end executes test requests to the two services.

9.3. Hybrid recommender testing

The recommender was tested by creating a user with the profile below. The recommender was run and some of the recommended activities are on the right.

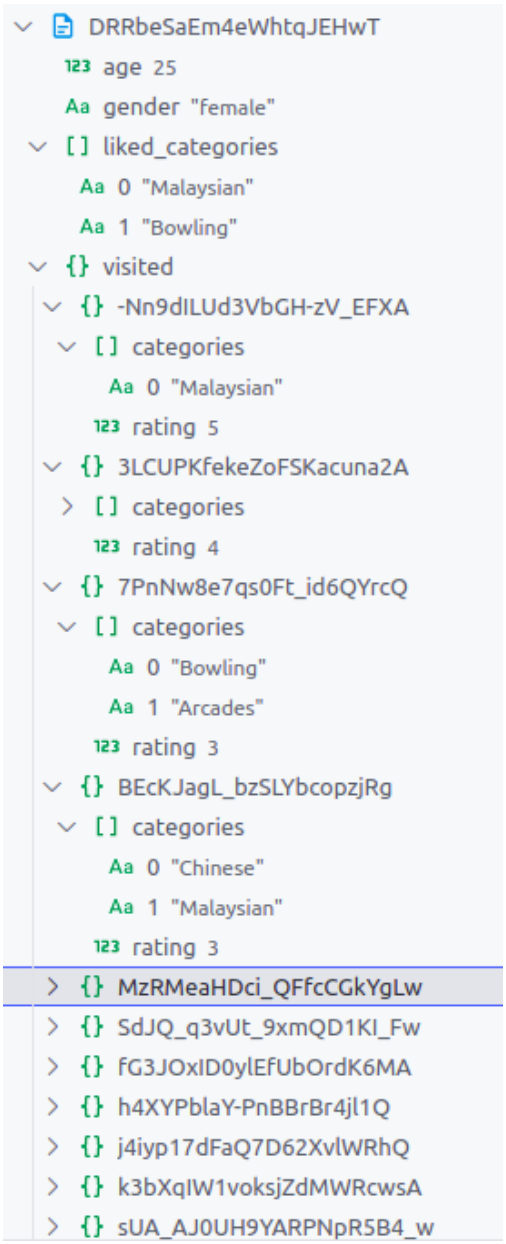



```
{
    "b0": {
        "categories": {
                "title": "Museums"
                "title": "Art Galleries"
        },
        "name": "Hugh Lane Gallery",
        "rating": 4.5
    },
   "b4": {
        "categories": {
                "title": "Home & Garden"
                "title": "Hiking"
},
        "name": "Colemans of Sandyford",
        "rating": 4.0
    },
   "b5": {
        "categories": {
                "title": "Local Flavor"
                "title": "Bookstores"
                "title": "Art Galleries"
}
        "name": "Heraldic Artists",
    "rating": 4.5
},    "b3": {
    "categories": [
            "title": "Art Galleries"
    "name": "National Photographic Archive",
```


The activities such as ‘Bulgarian’ and ‘Bungee Jumping’ are not available in Dublin, this is the reason they don’t appear on the list. In addition, after several tests it was noticed that often more than 50% of the recommendations are of the same type. The reason is that at the moment only randomly generated data is available in the database and it doesn’t cover all possible categories. In addition, it can be concluded that cosine similarity should be calculated only if the

user has visited a diverse range of activities in terms of their liked categories. Here 10 Art Galleries were recommended out of 15 activities. The rest were Parks and Hiking.

- Another extreme use case was tested when a dummy user has a small number of liked categories but they have visited many places matching the liked categories.




```
{
    "b0": {
        "categories": {
                "title": "Chinese"
                "title": "Malaysian"
        },
        "name": "The Noodle House",
    },
    "b1": {
        "categories": {
                "title": "Asian Fusion"
                "title": "Japanese”
        },
        "name": "J's Noodle & Sushi House",
    "b2": {
        "categories": {
                "title": "Japanese"
                "title": "Asian Fusion"
        },
        "name": "Eatokyo Asian Street Food",
},    "b3": {
        "categories": {
                "title": "Japanese"
                "title": "Asian Fusion"
                "title": "Korean"
        },
 "name": "Hailan Asian Cuisine Restaurant",}...
```


Here it is recommended not only Malaysian cuisine but also other Asian cuisines. However, nothing else outside Asian cuisine and Bowling was suggested. The reason is that the liked

categories are only two and there is a small chance to find similar users in our database. The returned categories were also low, only 7.

- Lastly, a randomly generated user with a lot of liked categories and visited activities was tested. The returned activities were 54.


```
"b22": {
    "categories": {
            "title": "Coffee & Tea"
            "title": "Breakfast & Brunch"
    },
    "name": "Bibi's Cafe",}
"b20": {
    "categories": {
            "title": "Malaysian"
    },
    "name": "Kopitiam Dublin",}
"b13": {
    "categories": {
            "title": "Dance Clubs"
    },
    "name": "Krystle",}
"b14": {
    "categories": {
            "title": "Bars"
            "title": "Tapas Bars"
            "title": "Fast Food"}
    "name": "The Market Bar",}
```


In addition to the recommended activities with categories similar to the visited, there were some new suggestions based on the user score, too. Such as Dance Club, Bars and Coffee.

It can be concluded that the cosine similarity based recommender requires a huge amount of data to function properly. In our database there were only 50 users. So users with “small”, not diverse enough profiles suffered. To resolve this problem, first only users with a diverse profile should be considered for the cosine similarity based recommender. Other users should be redirected to a content based recommender or in our case recommendations should come from Yelp.

9.4. Planner testing

The testing of the planner was done in a similar fashion as the recommender.

Input to the planner:

KEY	VALUE
user_id	FBUmHzsMQcEl6KAiAzNB
start_time	10:20
end_time	18:33
latitude	53.3471
longitude	-6.2719

Output, where time is walking time in minutes from the start/previous activity, the same applies to distance which is in km:

1. "name": "FAI National League","categories": [Sports Clubs] "time": 27.383333333333333 "distance": 2.282
2. "name": "St. Kevin's Park", "categories": ["Landmarks & Historical Buildings", "Parks"] "distance": 1.396, "time": 16.75
3. "name": "Trinity College Sports Centre", "categories": ["Gyms","Sports Clubs", "Swimming Pools"] "distance": 1.839, "time": 22.066666666666666
4. "name": "The Ramen Bar","categories": ["Ramen”],"time": 16.983333333333334 "distance": 1.415
5. "name": ["Irish Martial Arts Council -IMAC"], "categories": ["Sports Clubs","Amateur Sports Teams","Martial Arts"], "time": 21.416666666666668 "distance": 1.785
6. "name": "SPORTSCO”,"categories": ["Recreation Centers"],"time": 32.016666666666666, "distance": 2.662,
7. "name": "Blas Café", "categories": ["Cafes","Breakfast & Brunch"],time": 22.05, "distance": 1.837

The first issue which is noticed is the number of sports activities. They dominate in the plan because the user favors sports activities. However, they at least are different. This could be worked on in the future, by teaching the algorithm to understand synonyms of sport.
The walking distance varies between 1.3km and 2.6km which is normal for an 8 hours time frame.

user_id	5olMCywEKos9V1QH3QS4
start_time	10:50
end_time	14:33
latitude	53.3471
longitude	-6.2719

1. "name": "Park Hotel","categories": ["Amusement Parks","Hotels"],"time": 7.333333333333333, "distance": 0.611
2. "name": "Spraoi", "categories": ["Kids Activities"],"time": 0.0, "distance": 0.0
3. "name": "Ka Shing","categories": ["Szechuan","Dim Sum"],"time": 6.65 "distance": 0.554,
4. "name": "The Ark", "categories": ["Performing Arts","Art Classes","Playgrounds"],"time": 5.666666666666667 "distance": 0.47,

The number of activities added in the plans are reasonable for the given time. Here the activities are more diverse because the time frame is shorter.