FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Web application for recommendations of Points Of Interests
Student:	Bc. Maryna Kryvosheienko
Supervisor:	Ing. Jaroslav Kuchař, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

The goal of the thesis is a design, implementation and evaluation of a proof of concept web application acting as a POI recommender system.
- Analyse existing POI recommender systems and investigate methods for recommendations.
- Design the web application that will use a selected hybrid method for the recommendations of POIs.
- Design and implement a server-side of the application including REST API. Use Python for the server-side.
- Design a user interface. Implement a client-side that will use implemented REST API. For the client-side use frameworks like Bootstrap, React.
- Test the web application.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 16, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Web application for recommendations of Points Of Interests

Bc. Maryna Kryvosheienko

Department of Software Engineering
Supervisor: Ing. Jaroslav Kuchař, Ph.D.,

May 7, 2019

Acknowledgements

I would like to thank Ing. Jaroslav Kuchař, Ph.D., for all the suggestions and expert advices he gave me during writing the thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 7, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Kryvosheienko, Maryna. *Web application for recommendations of Points Of Interests.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Hlavním cílem této práce je návrh, implementace a vyhodnocení webové aplikace, která funguje jako doporučující systém míst zájmů. Práce se zaměřuje na zkoumání metod, které mohou být využité pro vývoj doporučujícího systému, navrhování hybridního algoritmu na základě provedeného výzkumu, návrh a implementace webové aplikace, která využívá navrhovaný hybridní přístup. Součástí práce je také zhodnocení výsledků použité metody a výsledků testování vyvinuté webové aplikace.

Klíčová slova doporučující systém, hybridní doporučující systém, místo zájmu, webová aplikace

Abstract

The main goal of this thesis is a design, implementation and evaluation of a proof of the concept web application acting as the Points Of Interests recommender system. The thesis focuses on the investigation of methods, that can be used for developing a recommender system, designing the hybrid algorithm, based on provided research, designing and implementing the web application which uses the proposed hybrid approach for making recommendations. Also,

the thesis includes evaluation of the used method's results and results of the testing of the developed web application.

Keywords recommender system, hybrid recommender system, Points Of Interest, web application

Contents

List of Figures

List of Tables

Introduction

Recommender systems relate to systems that can predict the future preference of a set of items for a user, and recommend the best of them. For achieving this task a lot of different approaches have been proposed up to now.

In our days, people more often use recommender systems in different spheres of daily life. Recommender systems facilitate the problem of choosing, which products to buy, which movie to watch, which book to read, etc. They save users' time for picking and at the same time help to attain higher satisfaction.

With the development of the mobile Internet, people start to use different applications and web sites that help to find Points Of Interest (POI) such as restaurants, bars, sports clubs and others. In this situation, a good POI recommender system could facilitate the search and, based on a user's previous behaviour, show personalized recommendations of the best POIs fitting the user preferences. Personalized POI recommendation is especially important and useful when a user travels to the new city or country, where he has little knowledge about this area.

This thesis focuses on the developing of the web application that will act as POI recommender system. This application will help users find new exciting places in their home city or in the city which they plan to visit in short terms. All user recommendations will be based on user's preferences. The application will use a hybrid method that will combine different approaches, which are used for building a recommender system. Combining several methods to one is a good approach because it allows reducing the shortcomings of each method that will be used in the final algorithm.

This work consists of five chapters. The first one describes what recommender system is, different types of recommender systems and methods that are used for developing recommender system. The second chapter focuses on analysing existing POI recommender systems and algorithms that can be used for developing POI recommender system. Also, the application requirements are described in this chapter. Next chapter represents the design of

the developing web application. It includes the detailed description of the algorithm which lies at the basis of the application, design of application architecture, database, description of technologies that will be used and user interface prototype. The fourth chapter relates to the implementation of the web application. And the last chapter contains the evaluation of the used recommender algorithm and user testing of the developed web application.

Chapter 1

Recommender systems

1.1 Introduction

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [1]. The suggestions relate to different decision-making processes, for example, what products to buy, what books to read, what places to visit, etc. So an "item" can be anything that user is looking for. The main aim of RS is to suggest to every user the most appropriate item in which he might be interested. This suggestion might be personalised for each user and based on his preferences. Personalisation involves matching the context in sense of the user specifics, preferences and history to infer on the selection procedure and provide relevant results [2]. In that case, different users or user groups get different recommendations.

RSs are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number of alternative items that a Web site, for example, may offer [3]. RS makes a ranked list of items, where it tries to predict the most relevant product or service, which will be interesting for a user. This prediction based on information about user preferences that was collected explicitly or implicitly.

- Explicitly – typically by collecting users' ratings, evaluating of likes and dislikes for an item. The problem occurs when the user does not rate the item, and RS does not know if the user likes this item or not.

- Implicitly – typically by monitoring users' behaviour, for example, browsing history, search patterns, or even mouse movements, etc.

Also, RS can use different demographic user characteristics, such as gender, age, nationality; social information like followers, followed, twits, and posts, is commonly used in Web 2.0.; or even information from Internet of things (e.g., GPS locations, real-time health signals) [4].

There are a lot of different techniques that are used by RS for making recommendations. The most common and widely-used are collaborative filtering methods and content-based recommender methods. The *collaborative filtering* methods are based on user-item interactions. It can be ratings or buying behaviour. The *content-based* methods are based on information about an item. It can be a description of an item, its categories or keywords. Besides, RSs can use knowledge-based recommender methods. In knowledge-based methods users explicitly set their requirements for items they want to receive. Instead of using historical rating or buying data, external knowledge bases and constraints are used to create the recommendation [5]. Some recommender systems combine different recommender methods to design hybrid recommender systems. Hybrid RSs use the strengths of other methods and usually provide better results.

1.2 Collaborative Filtering Recommender System

Collaborative recommendation is probably the most familiar, most widely implemented and most mature of the technologies [6]. Collaborative filtering (CF) methods create user-specific recommendations which are usually based on items' ratings. CF doesn't need any additional information about either items or users characteristics. This method recommends to the active user the items that other users with similar tastes liked in the past. The similarity in the taste of two users is calculated based on the similarity in the rating history of the users [7].

Typically there is a set of users and a set of items, and each user usually rates only a small subset of presented items. So the user-item matrix, which is used for CF, is extremely sparse. It is the primary challenge which CF methods must deal with. The other problem in using CF methods is the cold start. This problem appears when a new user starts using the system. It's hard to make good recommendations for him because he hasn't rate any items yet or he has rated only a few items, so it's difficult to find users with similar tastes. Also, the cold start problem can occur when a new item will be added to the system. This item will not be recommended to the user until some other users will rate it.

There are two types of methods that are commonly used in collaborative filtering: *memory-based* methods and *model-based* methods [8]:

- ***Memory-based*** methods operate over the entire user database to make predictions. The most popular memory-based CF approaches are user-based CF and item-based CF.

- In ***Model-based*** collaborative filtering, in contrast, uses the user database to estimate or learn a model, which is then used for predictions.

CF categories	Representative techniques	Main advantages	Main shortcomings
Memory-based CF	- Neighbor-based CF (item-based/ user-based CF algorithms with Pearson/vector cosine correlation) - Item-based/user-based top-N recommendations	- Easy implementation - New data can be added easily and incrementally - Need not consider the content of the items being recommended - Scale well with co-rated items	- Are dependent on human ratings - Performance decrease when data are sparse - Cannot recommend for new users and items - Have limited scalability for large datasets
Model-based CF	- Bayesian belief nets CF - Clustering CF - MDP-based CF - Latent semantic CF - Sparse factor analysis - CF using dimensionality reduction techniques, for example, SVD, PCA	- Better address the sparsity, scalability and other problems - Improve prediction performance - Give an intuitive rationale for recommendations	- Expensive model -building - Have trade-off between prediction performance and scalability - Lose useful information for dimensionality reduction techniques

Table 1.1: Overview of collaborative filtering techniques.

Each of these named methods has it's own advantages and shortcomings. A brief overview of CF techniques is depicted in Table 1.1 [9].

1.2.1 Memory-based Collaborative Filtering

Memory-based collaborative filtering algorithms are one of the earliest algorithms that were designed for collaborative filtering. These algorithms are also called *neighborhood-based* algorithms. Neighborhood-based algorithms are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings [5].

The neighborhood-based CF algorithm, a prevalent memory-based CF algorithm, uses the following steps [10]:

1. calculate the similarity or weight, $w_{i,j}$, which reflects distance, correlation, or weight, between two users or two items, i and j;

2. produce a prediction for the active user by taking the weighted average of all the ratings of the user or item on a certain item or user, or using a simple weighted average.

There are two main types of memory-based CF algorithms [5]:

- *User-based collaborative filtering:* In this case, the ratings provided by similar users to a target user A are used to make recommendations for A. The predicted ratings of A are computed as the weighted average values of these "peer group" ratings for each item.
- *Item-based collaborative filtering:* In order to make recommendations for target item B, the first step is to determine a set S of items, which are most similar to item B. Then, in order to predict the rating of any particular user A for item B, the ratings in set S, which are specified by A, are determined. The weighted average of these ratings is used to compute the predicted rating of user A for item B.

The most important step in memory-based CF algorithms is similarity computation between users or items. There are a lot of different methods to compute the similarity, for example, the cosine similarity, the Pearson's correlation coefficient, or the Jaccard coefficient and others. The higher the value of the similarity means the closer (i.e. the more similar) the users' or items' vectors are.

The cosine similarity measures the cosine of the angle between users' or items' vectors. Formula 1.1 is used for computing cosine similarity.

$$sim(u,v) = \frac{\sum_{i \in I} r_{u,i} \times r_{v,i}}{\sqrt{\sum_{i \in I} r_{u,i}^2} \sqrt{\sum_{i \in I} r_{v,i}^2}} \tag{1.1}$$

where I is the set of items or users and $r_{u,i}$ and $r_{v,i}$ are the ratings given to item i by users u and v, respectively [11].

Jaccard coefficient compares two sets of items with shared and distinct members and can be calculated by Formula 1.2.

$$sim(u,v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} \tag{1.2}$$

where I_u and I_v are the sets of items rated by users u and v.

The Pearson's correlation coefficient is the most commonly used. This coefficient can be found by Formula 1.3.

$$sim(u,v) = \frac{\sum_{i \in I}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I}(r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{i \in I}(r_{v,i} - \overline{r}_v)^2}} \tag{1.3}$$

where the $i \in I$ summations are over the items that both the users u and v have rated and $\overline{r}_u$ is the average rating of the co-rated items of the uth user [9].

Top-N recommendation is getting a set of N top-ranked items that will be interesting to the particular user. For the task of generating a top-N recommendation, the algorithms look for most similar users or items (nearest neighbours), calculate similarities and aggregate the neighbours for getting the top-N most frequent items. These items will be the user recommendations.

1.2.2 Model-based Collaborative Filtering

In model-based methods, a summarized model of the data is created up front, as with supervised or unsupervised machine learning methods. Therefore, the training (or modelbuilding phase) is clearly separated from the prediction phase [5]. The methods that are usually used for this purpose are clustering models, Bayes classifiers, rule-based methods, regression models, SVD methods and others. Almost all these models can be generalized to the collaborative filtering scenario, just as k-nearest neighbour classifiers can be generalized to neighbourhood-based models for collaborative filtering. This is because the traditional classification and regression problems are special cases of the matrix completion (or collaborative filtering) problem [5]. These mentioned model-based CF algorithms have been developed to solve the shortcomings of memory-based CF algorithms.

1.3 Content-Based Recommender System

Content-based recommender systems are designed to make suggestions that are based on items, which can be described with the set of attributes. Content-based algorithms use items' characteristics to understand the user's taste, find items with similar characteristics and recommend them to the particular user. This type of recommender systems tries to match users to items that have similar attributes with items what they have rated with high rates in the past. Unlike collaborative systems, which explicitly leverage the ratings of other users in addition to that of the target user, content-based systems largely focus on the target user's own ratings and the attributes of the items liked by the user [5]. This approach is advantageous when a new item will be added. Content-based methods can extract the attributes from the new item, and then it can recommend this new item to the users, based on extracted attributes. So it solves the cold-start problem for new items. On the other hand, the cold-start problem for new users cannot be addressed with content-based recommender systems. Furthermore, by not using the ratings of other users, one reduces the diversity and novelty of the recommended items [5].

Content-based systems are largely used in scenarios in which a significant amount of attribute information is available at hand [5]. Usually, these attributes are keywords, which are extracted from the item descriptions.

At the most basic level, content-based systems are dependent on two sources of data [5]:

1. The first source of data is a description of various items in terms of content-centric attributes. An example of such a representation could be the text description of an item by the manufacturer.

2. The second source of data is a *user profile*, which is generated from user feedback about various items. The user feedback might be explicit or implicit.

The user profile regularly consists of various types of information. The most common is the model of the user's preferences and history of the user's interactions. The model of the user's preferences, it is usually a description of items that the user is interested in. There are many possible alternative representations of this description, but one common representation is a function that for any item predicts the likelihood that the user is interested in that item. For efficiency purposes, this function may be used to retrieve the n items most likely to be of interest to the user. The history of the user's interactions may include storing the items that a user has viewed together with other information about the user's interaction, (e.g., whether the user has purchased the item or a rating that the user has given the item) [12].

The recommendation process is performed in three steps, each of which is handled by a separate component [7]:

- *Content Analyzer*: The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps. This representation is the input to the *Profile Learner* and *Filtering Component*.

- *Profile Learner*: This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile.

- *Filtering Component*: This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. This step might be very efficient because the suggestions need to be performed in real time.

Content-based recommender systems have several advantages and shortcomings. Main advantages are:

- New items can be easily added to the system. They do not need to have any ratings to be recommended to the users.

- It is user independence approach. Content-based RS need not have ratings from other users for making recommendations. Systems might not have a significant amount of users and ratings for making good recommendations.

- Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations [7].

Nevertheless, content-based systems have some shortcomings:

- Even though content-based methods are effective at providing recommendations for new items, they are not effective at providing recommendations for new users. In order to get accurate recommendations, the user must have a enough ratings [7].

- Limited understanding of content. It might be hard to include all features that mark the aspects that make content favorable to a user, which means that the system can easily misunderstand what the user likes [13].

1.4 Knowledge-based Recommender System

Knowledge-based recommender systems suggest items based on specific domain knowledge about how items satisfy user preferences. These systems are useful for a recommendation in domains of items that are rarely bought, for example, cars, houses and so on. Moreover, users often need to specify their requirements explicitly. A user may often be willing to accept a movie recommendation without much input, but he would be unwilling to accept recommendations about a house or a car without having detailed information about the specific features of the item [5]. The benefit of knowledge-based systems is that users have better control of items that will be suggested because they can specify their requirements in a very detailed way. Knowledge-based recommender systems will be suitable in the following situations [5]:

1. Customers want to explicitly specify their requirements. Therefore, interactivity is a crucial component of such systems. Collaborative and content-based systems do not allow this type of detailed feedback.

2. It is difficult to obtain ratings for a specific type of item because of the greater complexity of the product domain in terms of the types of items and options available.

3. In some domains, such as computers, the ratings may be time-sensitive. The ratings on an old car or computer are not very useful for recommendations because they evolve with changing product availability and corresponding user requirements.

The main characteristic of knowledge-based systems is a high level of customization to the specific domain. This customization is achieved through the use of a knowledge-base that encodes relevant domain knowledge in the form of either constraints or similarity metrics. Some knowledge-based systems

might also use user attributes (e.g., demographic attributes) in addition to item attributes, which are specified at query time. In such cases, the domain knowledge might also encode relationships between user attributes and item attributes [5].

There are two types of methods that are commonly used in knowledge-based RS:

- *Constraint-based*: In constraint-based systems users typically specify requirements or constraints (e.g., lower or upper limits) on the item attributes. Furthermore, domain-specific rules are used to match the user requirements or attributes to item attributes. These rules represent the domain-specific knowledge used by the system. Such rules could take the form of domain-specific constraints on the item attributes (e.g., "Cars before year 1970 do not have cruise control.") [14].

- *Case-based*: In case-based recommender systems, specific cases are specified by the user as targets or anchor points. Similarity metrics are defined on the item attributes to retrieve similar items to these targets. The similarity metrics are often carefully defined in a domain-specific way. Therefore, the similarity metrics form the domain knowledge that is used in such systems. The returned results are often used as new target cases with some interactive modifications by the user. For example, when a user sees a returned result that is almost similar to what she wants, she might reissue a query with that target, but with some of the attributes changed to her liking [15, 16].

Knowledge-based recommender systems are generally designed for domains in which the items are highly customized, and it is difficult for rating information to directly reflect greater preferences. In such cases, it is desirable to give the user greater control in the recommendation process through requirement specification and interactivity. Knowledge-based systems are largely based on user requirements, and they incorporate only a limited amount of historical data. Therefore, they are usually effective at handling cold-start issues. The drawback of this approach is that historical information is not used for "filling in the gaps". In recent years, methods have also been designed for incorporating a greater amount of personalization with the use of historical information from user sessions [5].

1.5 Hybrid Recommender System

Hybrid recommender systems combine several approaches for recommender systems. The main aim of the hybrid RS is to combine the power of different, avoid shortcomings of the simple RS methods especially the cold-start problem and provide better results.

There are three primary ways of designing hybrid recommender systems [13]:

- *Monolithic*: It takes different components of several recommender systems and join them together in a new ways to improve overall performance.
- *Ensemble*: The idea is to calculate recommendations using several full recommenders and then somehow combine results to the final recommendation.
- *Mixed*: It runs a number of recommenders, returning all of them.

The main difference between an ensemble and a mixed RS is that the first one will show only the final combined result, while the mixed type of RS will show all results for every recommender.

All hybrid recommender systems can be classified into the following categories [6]:

- *Weighted*: The scores of different recommender systems are combined into a single unified score by computing the weighted aggregates of the scores from individual components [5]. Usually, the methodology for weighting the results of recommender systems is based on heuristic.
- *Switching*: This technique allows choosing between different recommender systems and applying the selected one that provides the best suggestion at a given period.
- *Mixed*: Recommendations from different recommenders are presented to the user at the same time.
- *Feature Combination*: Features derived from different knowledge sources are combined together and given to a single recommendation algorithm.
- *Feature Augmentation*: One recommendation system computes a feature or set of features, and then this features are used as part of the input to the next RS.
- *Cascade*: In this case, one recommender system refines the recommendations given by another. In generalized forms of cascades, such as boosting, the training process of one recommender system is biased by the output of the previous one, and the overall results are combined into a single output [5].
- *Meta-level*: One recommendation system is applied and produces some sort of model, and then it is used as the input to the next recommender system.

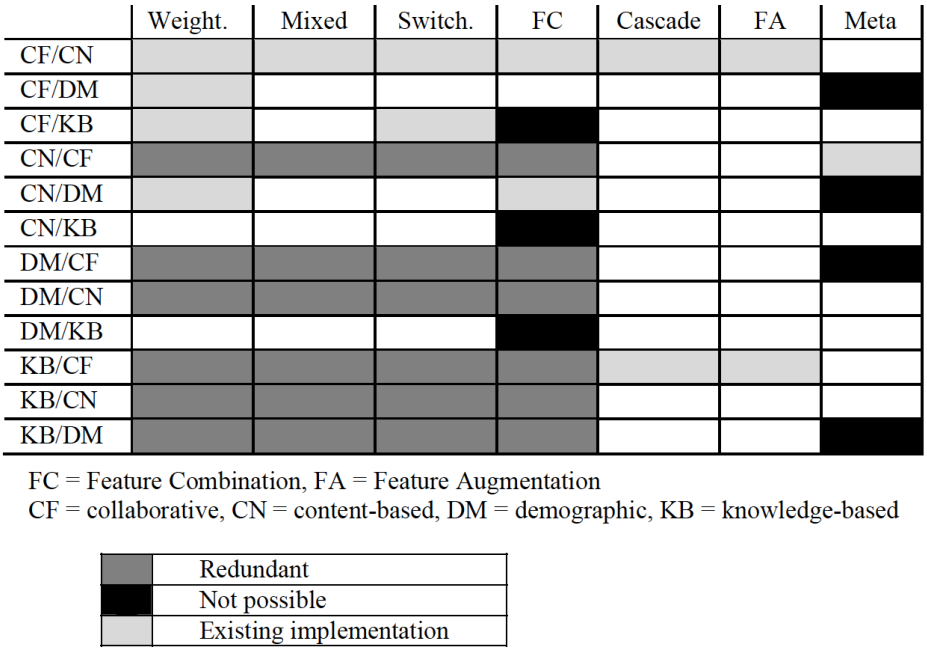


Figure 1.1: The space of possible hybrid recommender systems

In the Figure 1.1 [12] is represented the summary of designing hybrid recommender systems, that are built on different recommender systems methods.

1.6 Evaluation of Recommender System

Each recommender system has its own advantages and shortcomings. So the evaluation is critical for the understanding of the effectiveness of different recommendation methods. The quality of RS methods is based on the results of the evaluation.

Empirical evaluations of recommender systems usually focused on the evaluation of a recommender system's accuracy. An accuracy metric empirically measures how close a recommender system's predicted ranking of items for a user differs from the user's true ranking of preference. Accuracy measures may also measure how well a system can predict an exact rating value for a specific item [17].

There are three main types of accuracy metrics: *predictive* accuracy metrics, *classification* accuracy metrics and *rank* accuracy metrics.

1.6.1 Predictive Accuracy Metrics

Predictive accuracy metrics measure how close the recommender system's predicted ratings are to the true user ratings [17]. For this purpose usually used

Mean Absolute Error (MAE), *Normalized Mean Absolute Error (NMAE)* and *Root Mean Squared Error (RMSE)*.

Mean Absolute Error is the most widely-used metric for evaluation of recommender system. It calculates the average of the absolute difference between the predictions, that are made by RS, and true users' ratings [9]. It uses the Formula 1.4.

$$MAE = \frac{\sum_{\{i,j\}} |p_{i,j} - r_{i,j}|}{n} \tag{1.4}$$

where n is the total number of ratings over all users, $p_{i,j}$ is the predicted rating for user i on item j, and $r_{i,j}$ is the actual rating. The lower value of the MAE means the better prediction of the item's rating.

Beyond measuring the accuracy of the predictions at every rank, there are two other advantages to mean absolute error. First, the mechanics of the computation are simple and easy to understand. Second, mean absolute error has well studied statistical properties that provide for testing the significance of a difference between the mean absolute errors of two systems [17].

Different recommender systems may use different numerical rating scales. Normalized Mean Absolute Error normalizes MAE to express errors as percentages of full scale [18]. It can be calculated by Formula 1.5.

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{1.5}$$

where r_{max} and r_{min} are the upper and lower bounds of the ratings.

Root Mean Squared Error amplifies the contributions of the absolute errors between the predictions and the true values [9]. RMSE can be found by using Formula 1.6.

$$RMSE = \sqrt{\frac{1}{n} \sum_{\{i,j\}} (p_{i,j} - r_{i,j})^2} \tag{1.6}$$

where n is the total number of ratings over all users, $p_{i,j}$ is the predicted rating for user i on item j, and $r_{i,j}$ is the actual rating again.

Although accuracy metrics have greatly helped the field of recommender systems, the recommendations that are most accurate are sometimes not the ones that are most useful to users, for example, users might prefer to be recommended with items that are unfamiliar with them, rather than the old favorites they do not likely want again [19].

1.6.2 Classification Accuracy Metrics

Classification accuracy metrics measure the frequency with which a recommender system makes suitable or unsuitable suggestions of items for RS users. Classification accuracy metrics do not attempt to directly measure the ability of an algorithm to accurately predict ratings. Deviations from actual ratings

	Selected	Not Selected	Total
Relevant	N_{rs}	N_{rn}	N_r
Irrelevant	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

Table 1.2: Items classification in Precision and Recall metric.

are tolerated, as long as they do not lead to classification errors [17]. This types of metrics is mainly suitable for domains, where the user's preferences in recommendations are binary. The most common classification accuracy metrics are *Precision and Recall* and *Receiver Operating Characteristic (ROC) curve-based*.

The first one (Precision and Recall) is the most popular and widely-used metric. The main idea of this using this metric is to classify all items into two groups: relevant and irrelevant. This classification can be demonstrated in Table 1.2 [17].

Precision represents the probability that a selected item is relevant. It calculates by Formula 1.7.

$$Precision = \frac{N_{rs}}{N_s} \tag{1.7}$$

where N_{rs} is number of selected items that are relevant and N_s is total number of all selected items.

Recall represents the probability that a relevant item is selected by recommender system. It calculates by Formula 1.8.

$$Recall = \frac{N_{rs}}{N_r} \tag{1.8}$$

where N_{rs} is number of selected items that are relevant and N is total number of all relevant items.

Several approaches are combining precision and recall into one metric. F1 is one of them. F1-measure is the harmonic mean between precision and recall [20]. It can be found by using Formula 1.9.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{1.9}$$

As with all classification metrics, precision and recall are less appropriate for domains with non-binary granularity of true preference. For those tasks, at any point in the ranking, we want the current item to be more relevant than all items lower in the ranking. Since precision and recall only measure binary relevance, they cannot measure the quality of the ordering among items that are selected as relevant [17].

ROC curve-based metrics is an alternative to precision and recall metric. An ROC (Receiver Operating Characteristic) curve is a two-dimensional depiction of classifier performance, on which TPR (true positive rate) is plotted on the Y-axis and FPR (false positive rate) is plotted on the X-axis [9]. The TPR is the same as the recall. So, the ROC curve plots the "good" recall against the "bad" recall. Although ROC curves are often used for evaluating recommender systems, they do not always reflect the performance from the end-user perspective.

1.6.3 Rank Accuracy Metrics

Rank accuracy metrics measure the ability of a recommendation algorithm to produce a recommended ordering of items that matches how the user would have ordered the same items. Unlike classification metrics, ranking metrics are more appropriate to evaluate algorithms that will be used to present ranked recommendation lists to the user, in domains where the user's preferences in recommendations are nonbinary [17]. Ranking metrics do not attempt to measure the ability of an algorithm to accurately predict the rating for a single item – they are not predictive accuracy metrics. If a recommender system will be displaying predicted rating values, it is important to additionally evaluate the system using a predictive accuracy metric as described above.

The *Normalized Discounted Cumulative Gain (NDCG)* is a widely used evaluation rank accuracy metric. NDCG is designed for ranking tasks with more than one relevance levels [21]. The measure includes a position dependence for results shown to the user (that gives higher ranked results more weight). NDCG is defined as follows [21]. Set of numerical relevance grades is given: $Z = \{z_1, ..., z_n\}$.

The Discounted Cumulative Gain (DCG) for a given set of search results (for a given query) can be calculate by Formula 1.10.

$$DCG = \sum_{i=1}^{n} (c_i z_i) \tag{1.10}$$

where c_i is the discount factor of the ith document in the ranked list. The most common discount factor can be found by Formula 1.11.

$$c_i = \frac{1}{log(1+i)} \tag{1.11}$$

The normalised DCG is computed by Formula 1.12.

$$NDCG = \frac{DCG}{IDCG} \tag{1.12}$$

where IDCG is ideal DCG score. It is a DCG score of the best ranked list, which can be computed by Formula 1.13.

$$IDCG = maxDCG \tag{1.13}$$

The value of NDCG ranges from 0 to 1. A higher value indicates better ranking effectiveness.

Chapter 2

Analysis

2.1 Points Of Interests recommender systems

Points Of Interests recommender systems are useful to both users and businesses. With the help of the POI RS users can find new great POIs to visit, that they will like, and in that case, business will have new visitors.

In recent years a lot of different POI RS approaches have been developed. Nevertheless, in comparison with other recommender systems domains (e.g., movies, books), POI recommendations face new challenges [22] as follows:

- *Rich contexts*: First, a user's preference is frequently based on geographical position, because users regularly visit POIs which are situated in their activity regions: near home or workplace. Second, users are visiting the same places every day (e.g., home, workplace). Third, users' preference may be dependent on time. Users like to go to different types of places at a different time: for example, going to a coffee shop in the early morning and going to the bar in the late night. Fourth, users' visiting preferences might be influenced by their social levels. Other types of context may include reviews on POIs, social posts on POIs and others.

- *Data sparsity problem*: POI recommender systems have a big problem with sparsity, even worth than other types of recommender systems. Usually, users visit a very small part of all POIs in the system. For example, the density of the data used in experimental studies for POI recommendations is usually around 0.1%, while the density of Netflix data for movie recommendations is 1.2% [22].

POI recommendation methods use different types of context information and adopt different approaches for getting user's preferences. Memory-based collaborative filtering techniques, such as user-based CF and item-based CF, are exploited for POI recommendation [23].

Some of these methods are described below.

2.1.1 Location Recommendation framework without Temporal Effects

Location Recommendation framework without Temporal Effects (LRT) [24] is a time-enhanced Matrix Factorization model. LRT is designed to suggest places to users by taking advantage of temporal patterns, so it based on the observation that user's visiting behaviour depends on the time of day/week/year.

The whole framework consists of three steps: temporal division, temporal factorization, and temporal aggregation. Firstly, the original user-location matrix C is divided into T sub-matrices according to the T temporal states, with each sub-matrix only containing check-in actions that happened at the corresponding temporal state. Secondly, each C_t is factorized into the user check-in preference U_t and the location characteristics L, while L is shared by all of U_t. Finally, the corresponding low-rank approximation $\overline{C}_t$ is constructed and aggregated into $\overline{C}$ representing the user check-in preferences of each location [24].

2.1.2 Geographical Modeling and Matrix Factorization

Geographical Modeling and Matrix Factorization (GeoMF) [25] is a geographical weighted matrix factorization model. In order to capture the spatial clustering phenomenon (i.e., POIs visited by same users are supposed to be in the same region), GeoMF integrates geographical influence by modeling users' activity regions and the influence propagation on geographical space [22]. GeoMF divides the whole geographical space into grids, each of which represents a geographical region. For each POI, its influence is propagated to surrounding regions, attracting nearby users to visit [22].

2.1.3 LORE

LORE [26] considers sequential influence, in addition to social and geographical influence.

Social influence. Friend-based Collaborative Filtering is adopted to model social influence, where social similarities between friends are computed based on the distance of their residences. In case, if residence locations of users are not available, it can be taking the most frequent check-in POIs as users' residences.

Geographical influence. For each user, LORE models a check-in probability distribution over a two-dimensional space using Kernel Density Estimation. The geographical probability of visiting a new POI is then estimated based on its location on the check-in probability distribution.

Sequential influence. LORE employs additive Markov chain to exploit sequential influence between POIs. The sequential probability of a user visiting a POI is based on the transition probability between all the user's visited POIs and the target POI.

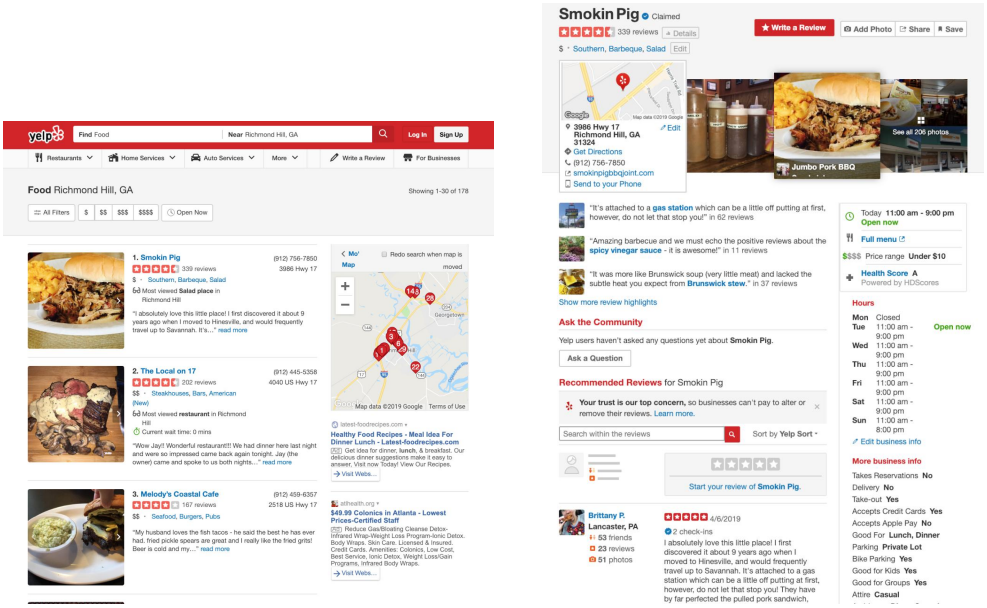
2.2 Points Of Interests recommender applications

In the last few years, the increasing interest in location-based services (LBS) has favoured the introduction of geo-referenced information in various Web 2.0 applications, as well as the rise of location-based social networks (LBSN) [27]. LBSN allows people looking for new interesting places, leave their reviews and tips about the advantages and disadvantages of visited POIs. This kind of feedback helps other users to find really great places to visit and avoid places, that they will not like. Nowadays it is a variety of different existing LBSNs, for example Yelp, Foursquare, TripAdvisor and many others.

2.2.1 Yelp

Yelp is one of the most popular LSBN. Yelp helps people find places they will like.

On the Yelp web site user can search for places within a category or location, and he will get a list of places, that will match the request. Users can get detail information about the POI they will choose. On the POI detail page, there is a description of the place, opening hours and list of properties (i.e. location, whether it takes credit cards, etc.). Also, on this page user can read individual reviews with ratings from other users, that have visited this place, and based on that information decide if they will like to go to that place or not. For leaving the review and rating (from 1-5 stars) for the visited POI,



(a) Search results page

(b) POI detail page

Figure 2.1: Yelp web site interface

the user may have a free account, which requires registration on the web site. Users may choose to submit reviews for many reasons. For example, Yelp provides direct incentives for reviewers, such as having occasional parties for people who have submitted a sufficiently large number of reviews [28].

The Yelp web site interface is described on Figure 2.1.

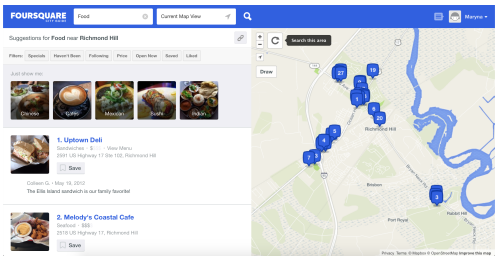
2.2.2 Foursquare

Foursquare is another widely popular LSBN. Foursquare "helps discover new places, with recommendations from a community you trust." [29].

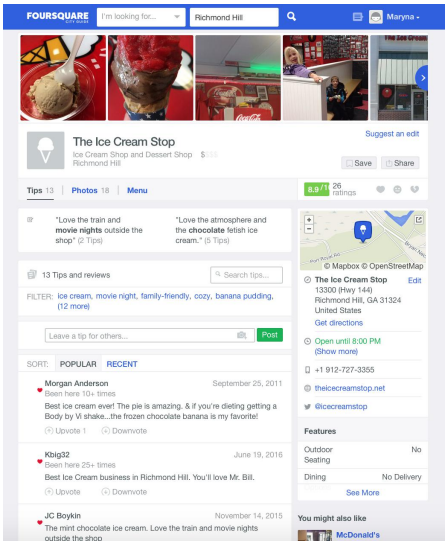
On the Foursquare web site, a user can search POIs by categories and the city. All POIs that satisfy user search request are displayed on the map, so user can see what places are situated nearby. Also, user can filter places by price, opened places or places he has been or hasn't been visited.

On POI detail page describes photo gallery and information about POI, including contact information, like web site, phone number, links to social networks. Registered users can leave some short tips about places, which they have visited. Other web site visitors can read them and use these tips while they will visit the POI.

Furthermore, on the Foursquare web site, registered users can save POIs, that they are interested in and want to visit, to the list. User can create several lists, and save there POIs by different criterions, for example by categories, cities, etc. Also, a user can add his tastes that will help to get better recommendations.

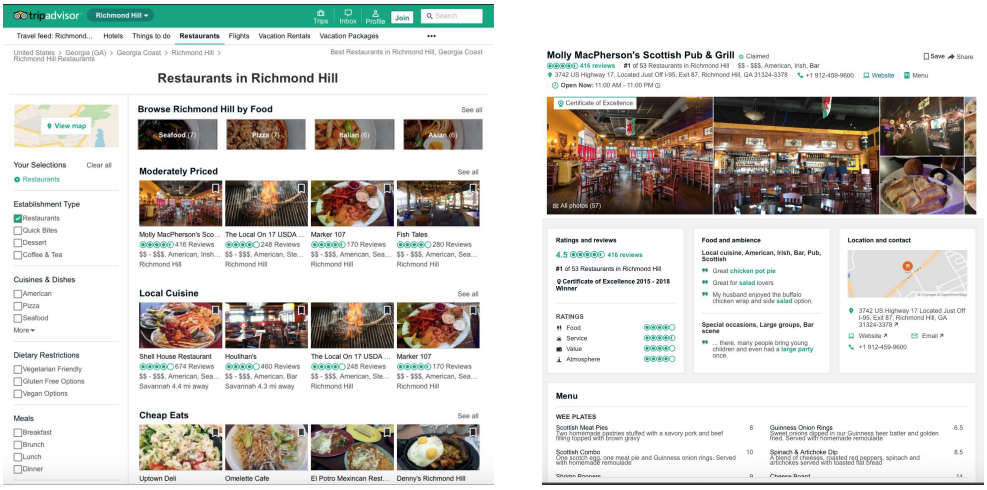


(a) Search results page



(b) POI detail page

Figure 2.2: Foursquare web site interface



(a) Search results page (b) POI detail page

Figure 2.3: TripAdvisor web site interface

The Foursquare web site interface is described on Figure 2.2.

2.2.3 TripAdvisor

TripAdvisor is an LSBN which is focused on travellers. The main idea is that travellers from all of the world can plan their own trips based on other travellers' reviews.

Users write reviews, comment and rate locations based on their impression. They can make a selection from five possible variants of rating: terrible, poor, average, very good and excellent. The rating allows TripAdvisor to create Traveller Ratings for every hotel, restaurant and visitor attraction within site. This process involves calculating a summary score based on the quality, quantity and age of the individual traveller reviews [30]. The result is represented as a five-point indicator for every place.

Furthermore, users can upload different multimedia content, such as photos or videos, that are related to the place they have visited. For doing all these actions, users may be registered in the system, so they must create a profile with basic personal information.

The TripAdvisor web site interface is described on Figure 2.3.

2.3 Application requirements

The main aim of the future developed application is to recommend to user points of interest that are matched his preferences. Based on that, were defined requirements which the developed application may satisfies. They are

divided into functional (specify a behaviour or function) and nonfunctional requirements (software and hardware requirements).

2.3.1 Functional application requirements

The functional requirements for the developing application are:

1. Providing to any user ability to create personal profile and manage it.
2. Searching POIs by cities and categories.
3. Ordering the list of POI search results depending on user roles:
   - For unauthorised users: ordering results by POIs' ratings.
   - For authorised users: ordering results based on users' recommendation scores.
4. Calculation the recommendation score for users based on their preferences and previous behaviors (visited and rated POIs).
5. Filtering the list of POI search results by categories, price range and attributes.
6. Displaying POIs' detailed information at POI's page: basic information, rating, categories, attributes, location on the map.
7. Displaying POIs' reviews and tips that were written by other users.
8. Providing to authorised users ability to write reviews and tips, and to rate places, which they have visited.
9. Providing to the administrator ability to add, edit and remove POIs from the developing application.
10. Providing to the administrator ability to add new cities to the developing application.
11. Providing to the administrator ability to manage users.

2.3.2 Nonfunctional developing application requirements

The nonfunctional requirements for application are:

1. The application will be available online with a web browser.
2. The application will be responsive and cross-browser.
3. Internet connection is required

Chapter 3

Design

3.1 Recommender algorithm

The main aim of the developing web application is to recommend POIs to users based on their previous behaviour (rating POIs and written reviews). For providing these recommendations, the application will use the hybrid recommendation algorithm that combine three approaches:

- Item-based collaborative filtering
- Singular-value decomposition method
- Content-based filtering

The mentioned methods will be combined together to produce the best suggestion to the user based on the category of POI and city he is interested in.

3.1.1 Item-based collaborative filtering

The item-based collaborative filtering approach looks into the set of items the target user has rated and computes how similar they are to the target item i and then selects k most similar items $\{i_1, i_2, ..., i_k\}$ [31]. For selecting k most similar items, the similarities $\{s_{i1}, s_{i2}, ..., s_{ik}\}$ for all item are computed. For the items with higher similarities, algorithm computes the prediction by taking a weighted average of the target user's ratings on these similar items.

The main idea of computing the similarity between the target item i and some other item j is to select users that have rated both of these items and then calculate the similarity $s_{i,j}$ with the help of one of similarity measures that were mentioned in the previous section: the cosine similarity measures, the Pearson's correlation coefficient and Jaccard coefficient. For the developing application, the similarity will be computed with the one of the most common measure: the Pearson's correlation coefficient similarity measures.

The weighted average sum is using for generating predictions. It computes the prediction on item i for a user u by computing the sum of the ratings given by the user on the items similar to i. Each ratings is weighted by the corresponding similarity $s_{i,j}$ between items i and j. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range [31]. Predictions can be computed by the Formula 3.1.

$$Predictions_{u,i} = \frac{\sum_{similar\ items,N}(s_{i,N} * R_{u,N})}{\sum_{similar\ items,N} |s_{i,N}|} \tag{3.1}$$

where $R_{u,N}$ is appropriate rating value for similar item N.

3.1.2 Singular-value decomposition method

Singular-value decomposition (SVD) is a matrix factorization technique commonly used for producing low-rank approximations. Given a matrix $A \in R^{m \times n}$ with $rank(A) = r$, the Singular-value decomposition of A is defined by Formula 3.2 [32].

$$A = USV^T \tag{3.2}$$

where $U \in R^{m \times n}$, $V \in R^{n \times n}$ and $S \in R^{m \times n}$.

The matrices U, V are orthogonal, with their columns being the eigenvectors of AA^T and A^TA, respectively. The middle matrix S is a diagonal matrix with r nonzero elements, which are the singular values of A. Therefore, the effective dimensions of these three matrices U, S and V are $m \times r$, $r \times r$ and $n \times r$, respectively. The initial diagonal r elements $(\sigma_1, \sigma_2, ..., \sigma_r)$ of S have the property that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ [32].

An important property of SVD, which is particularly useful in recommender system, is that it can provide the optimal approximation to the original matrix A using three smaller matrices multiplication. By keeping the first k largest singular values in S and the remaining smaller ones set to zero, we denote this reduced matrix by S_k. Then by deleting the corresponding columns of U and V, which are the last $r - k$ columns of U and V, we denote these two reduced matrices by U_k and V_k, respectively. The truncated SVD is represented by Formula 3.3.

$$A_k = U_k S_k V_k^T \tag{3.3}$$

where is the closest rank-k approximation to the original matrix A for any unitarily invariant norm [32].

The dimensionality reduction approach in SVD is beneficial for use in collaborative filtering. SVD provides a set of uncorrelated eigenvectors. Its corresponding eigenvector represents each user and item. The dimensionality reduction may be useful for users who rated similar items to be mapped into space spanned by the same eigenvectors.

When the $m \times n$ rating matrix R is decomposed and reduced into three SVD component matrices with k features U_k, S_k, and V_k [33], the prediction for any user u on item i can be computed with using of matrices $U_k\sqrt{S_k}$ and $\sqrt{S_k}V_k^T$. To compute the prediction, the scalar product of the u^{th} row of $U_k\sqrt{S_k}^T$ (denoted as $U_k\sqrt{S_k}(u)$) and the q^{th} column of $\sqrt{S_k}V_k^T$ (denoted as $\sqrt{S_k}V_k^T(q)$) is calculated by using Formula 3.4 [34].

$$prediction_{uq} = \overline{v}_u + \sigma_u[U_k\sqrt{S_k}(u) \cdot \sqrt{S_k}V_k^T(q)] \tag{3.4}$$

where $\overline{v}_u$ and σ are mean rating and standard deviation for user u, respectively.

3.1.3 Content-based filtering

Content-based filtering makes recommendations by matching user query terms with the index term used in the representation of the items, ignoring data from other users [35].

Let $U = \{u_1, ..., u_m\}$ be a set of users, and let $I = \{i_1, ..., i_n\}$ be a set of items. In content-based recommendation approaches ordered list of items that will be interesting for a user is computed [36] with the help of utility function described by Formula 3.5:

$$g(u_m, i_n) = sim(ContentBasedUserProfile(u_m), Content(i_n)) \tag{3.5}$$

where $ContentBasedUserProfile(u_m)$ is the content-based preferences of user u_m, for example, the item content features that describe the interests, tastes and needs of the user, and $Content(i_n)$ is the set of content features characterising item i_n. The function sim computes the similarity between a user profile and an item profile in the content feature space. For the developing application, the similarity will be computed based on the Pearson's correlation coefficient.

The easiest way to define the user profile is as a vector, where each component represents the number of times the user has rated with good points items with the appropriate attribute. Similarly, the item profile can be defined as a vector, where each component binary represents if the item has an appropriate attribute or not.

Items in the list of recommendations are ordered by computed similarity from the highest to lowest.

3.1.4 Hybrid algorithm

As it was mentioned before in the developing web application will be used the hybrid algorithm, that will calculate recommendations for users. This algorithm combines three methods that were described in previous sections.

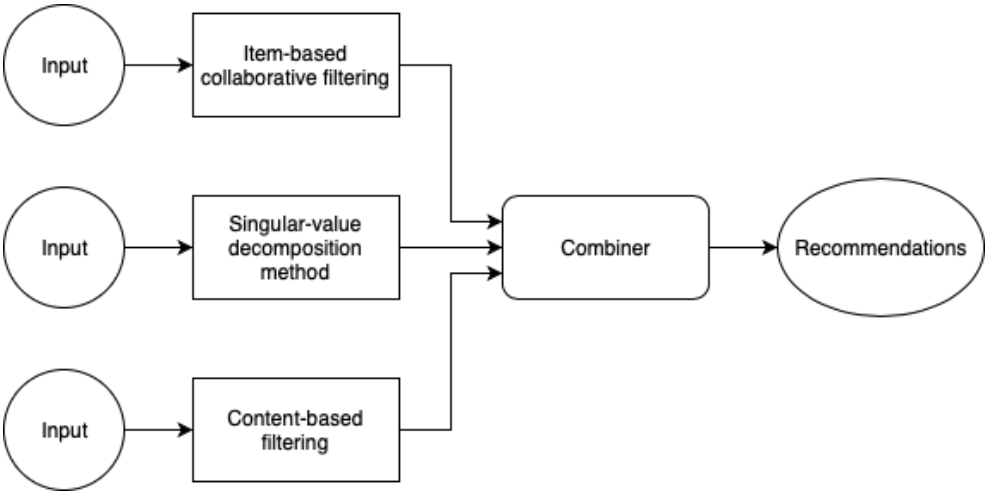


Figure 3.1: Hybrid algorithm scheme

Figure 3.1 represents the scheme of how these approaches will work together and provide recommendations for the user. It is an ensemble hybrid algorithm, that shows only the final combined result.

As the input for the hybrid algorithm will be used the following data:

- name of the category and the city which user uses for searching POIs request
- user's ratings that were left for the POIs that were visited before by the user
- categories of POIs, that user has visited and rated

Each of the algorithms will provide its computations and return the ordered list of recommended POIs. Then these ordered lists will be combined together at the "Combiner" step.

For combining provided lists to the one final recommendation list, for each POI will be calculated the average value of its rank positions in every computed list. This value will be computed by Formula 3.6.

$$averageValue_i = \frac{p_{cfi} + p_{svdi} + p_{cbi}}{3} \tag{3.6}$$

where p_{cfi} is the ith POI position in the item-based collaborative filtering result recommendation list, p_{svdi} is the ith POI position in the result recommendation list of the singular-value decomposition method and p_{cbi} is the ith POI position in the content-based filtering recommendation list.

Then all POIs will be ordered by this average value to the final list with recommendations from the lowest to the highest.

Combining these three methods together to the one algorithm will help to deal with the typical recommender systems problems, such as sparsity of the user rating matrix and ”cold-start” problem.

To avoid the ”cold-start” problem, when user will search for the POI located in the city, which user has never visited before, the algorithm will use only the content-based filtering approach. It will recommend POIs based on the categories of user preference.

If it will be a new user, who has no reviews and rated POIs, the application will recommend him POIs list ordered by the global rating.

3.2 Application architecture

The developing application will have a client-server architecture, which is illustrated on the Figure 3.2.

The client and the server will communicate with each other using REST API. The client will send JSON data to the server. The server will communicate with the database for processing requests due to the business logic. And then the server will return the response to the client in JSON format.

In the application also will be used third-party API – Google maps API. The client will send request to the API, and gets necessary JSON data.

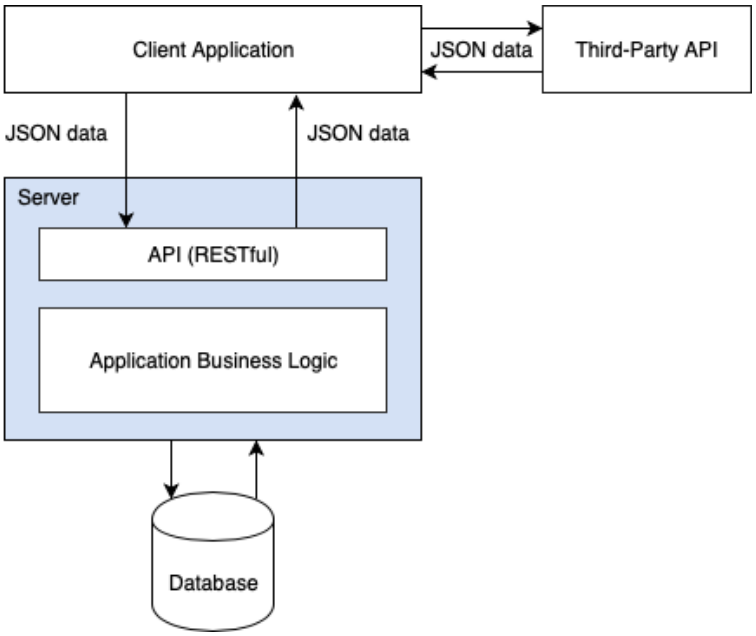


Figure 3.2: Application architecture

3.3 Database structure

For the developing application will be used data from the Yelp open dataset. It's a subset of Yelp's businesses, reviews, and user data. This dataset contains data about ten metropolitan areas in the USA and Canada.

Based on the described dataset, it was created the application entity relationship diagram, which is illustrated in Figure 3.3. The application database will be built on this diagram. All entities from the diagram will be described below.

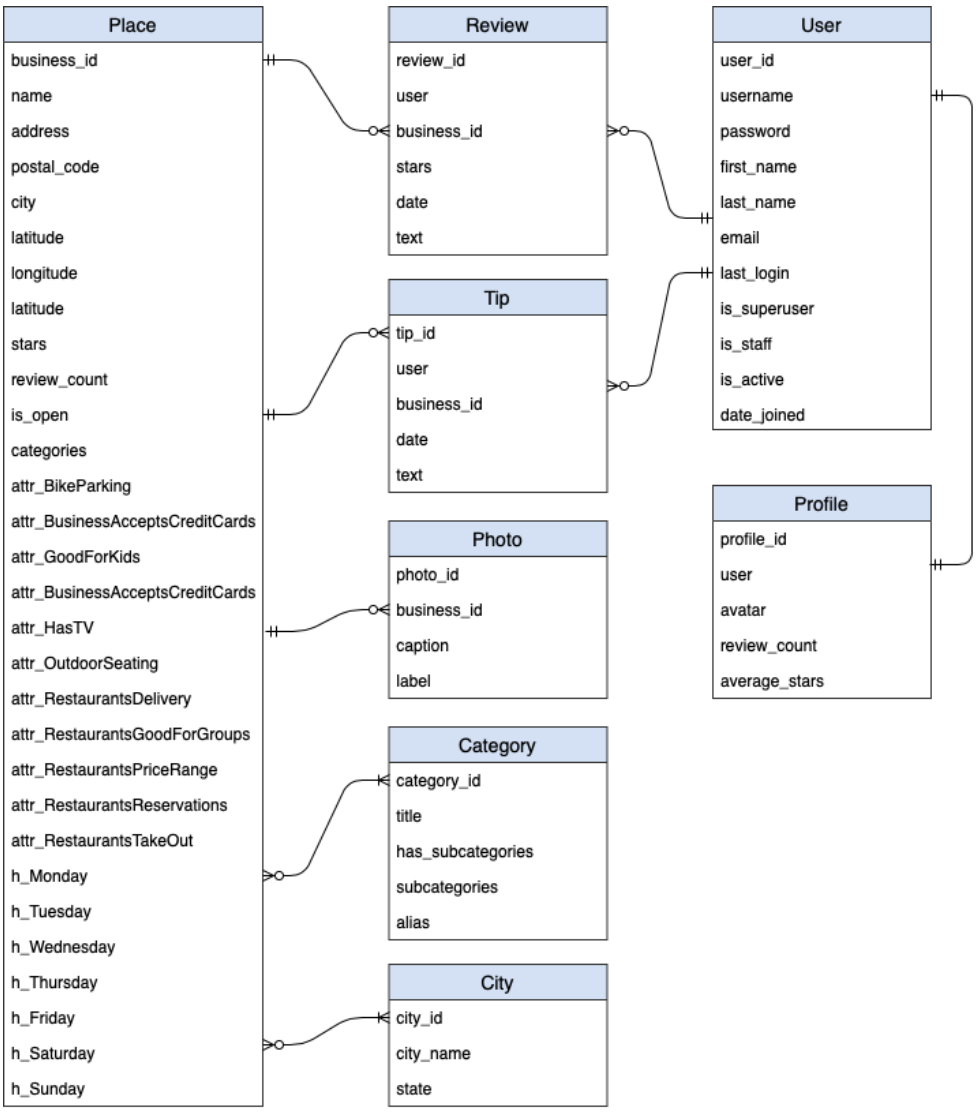


Figure 3.3: Application entity relationship diagram

Title	Data type	Description
business_id	varchar(120)	POI id. Primary key.
name	varchar(255)	POI name.
address	varchar(255)	POI address.
postal_code	varchar(15)	Postal code of POI location.
city	varchar(255)	City of POI location. Foreign key.
latitude	double	POI location latitude.
longitude	double	POI location longitude.
stars	double	POI average rating.
review_count	int	Amount of POI's reviews.
is_open	boolean	Describes if POI is open.
categories	text	POI's categories. Foreign key.
attr_BikeParking	boolean	POI attribute, describes if POI has bike parking.
attr_BusinessAccepts-CreditCards	boolean	POI attribute, describes if POI accepts credit cards.
attr_GoodForKids	boolean	POI attribute, describes if POI is good for kids.
attr_HasTV	boolean	POI attribute, describes if POI has TV.
attr_OutdoorSeating	boolean	POI attribute, describes if POI has outdoor seating.
attr_Restaurants-Delivery	boolean	POI attribute, describes if POI is a restaurant and has delivery .
attr_Restaurants-GoodForGroups	boolean	POI attribute, describes if POI is a restaurant and it's good for groups of people.
attr_Restaurants-Reservations	boolean	POI attribute, describes if POI is a restaurant and it allows reservations.
attr_Restaurants-TakeOut	boolean	POI attribute, describes if POI is a restaurant and it allows to take out food.
attr_Restaurants-PriceRange	int	POI attribute, describes average price range.
h_Monday	varchar(30)	POI Monday working hours.
h_Tuesday	varchar(30)	POI Tuesday working hours.
h_Wednesday	varchar(30)	POI Wednesday working hours.
h_Thursday	varchar(30)	POI Thursday working hours.
h_Friday	varchar(30)	POI Friday working hours.
h_Saturday	varchar(30)	POI Saturday working hours.
h_Sunday	varchar(30)	POI Sunday working hours.

Table 3.1: Place entity structure.

Place entity is representing POI, which users can visit. Its structure is described in the Table 3.1.

Review entity is representing review with rating, which users writes about POIs they have visited. It is a text with rating, where user describes why he likes or dislikes the POI, he has visited. Review entity structure is described in the Table 3.2.

Title	Data type	Description
review_id	varchar(120)	Review id. Primary key.
user	varchar(120)	User, leaved a review. Foreign key.
business_id	varchar(120)	POI, review was written about. Foreign key.
stars	int	User's rating for reviewed POI.
date	timestamp	Date when review was written.
text	text	Text of review.

Table 3.2: Review entity structure.

Tip entity is representing tips, which users write about POIs they have visited. It is usually small text where users advise something. Its structure is described in the Table 3.3.

Title	Data type	Description
tip_id	varchar(120)	Tip id. Primary key.
user	varchar(120)	User, leaved a tip. Foreign key.
business_id	varchar(120)	POI, tip was written about. Foreign key.
date	timestamp	Date when tip was written.
text	text	Text of tip.

Table 3.3: Tip entity structure.

Photo entity is representing photos connecting to POIs. It can be different views of POI or provides services. Photo entity structure is described in the Table 3.4.

Title	Data type	Description
photo_id	varchar(120)	Photo id. Primary key.
business_id	varchar(120)	POI, photo is referenced to. Foreign key.
caption	text	Photo caption.
label	varchar(255)	Photo label.

Table 3.4: Photo entity structure.

Category entity is representing categories to which POIs can be classified. Each category can include some subcategories. Category entity structure is described in the Table 3.5.

Title	Data type	Description
category_id	varchar(120)	Category id. Primary key.
title	varchar(255)	Category title.
has_subcategories	boolean	If category has subcategories.
subcategories	text	List of subcategories.
alias	varchar(255)	Formatted category title.

Table 3.5: Category entity structure.

City entity is representing cities where POI is located. Its structure is described in the Table 3.6.

Title	Data type	Description
city_id	varchar(120)	City id. Primary key.
city_name	varchar(255)	Name of the city.
state	varchar(15)	State where city is situated.

Table 3.6: City entity structure.

User entity is representing basic user's information that is necessary for registration and logging in. Its structure is described in the Table 3.7.

Title	Data type	Description
user_id	varchar(120)	User id. Primary key.
username	varchar(150)	User's username.
password	varchar(120)	User's password.
first_name	varchar(150)	User's first name.
last_name	varchar(150)	User's last name.
email	varchar(255)	User's email address.
last_login	timestamp	Date and time when user login last time.
is_superuser	boolean	If user has superuser rights.
is_staff	boolean	If user has staff rights.
is_active	boolean	If user account is still active.
date_joined	timestamp	Date and time when user was registered in the application.

Table 3.7: User entity structure.

Profile entity is representing user's profile. It is an additional information about user. Its structure is described in the Table 3.8.

Title	Data type	Description
profile_id	varchar(120)	Profile id. Primary key.
user_id	varchar(120)	User id. Foreign key.
avatar	varchar(255)	User's photo.
review_count	int	Amount of review, user has written.
average_stars	double	Average rating, user has left for POIs.

Table 3.8: Profile entity structure.

3.4 Server-side technologies

In this section will be discussed all technologies that will be used on a server-side of application.

The server side of the application will be developed in Python language using the Django framework.

3.4.1 Database

PostgreSQL will be used as a database for the developing application. It is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads [37].

The main advantage of PostgreSQL is that it provides enterprise-class performance and functions among current Open Source DBMS with no end of development possibilities. A fully open-source project, PostgreSQL's source code is developed by a large and devoted community. Similarly, the Postgres community maintains and contributes to numerous online resources that describe how to work with the DBMS, including the official documentation, the PostgreSQL wiki, and various online forums.

PostgreSQL supports languages similar to PL/SQL in Oracle such as PL/pgSQL, PL/Python, PL/Perl, C/C++, and PL/R.

3.4.2 Python

Python is one of the most popular programming languages in the world. It is very powerful and has clear and understandable syntax. Python is an interpreted, interactive, object-oriented programming language, that provides high-level data structures such as list and associative arrays (called dictionaries), modules, classes, exceptions, etc. [38]. Python transforms into a high-level language suited for scientific and engineering code that's often fast enough to be immediately useful but also flexible enough to be sped up with additional extensions [39].

This language was picked for developing the application because it has a lot of built-in general-purpose libraries and also it can be extended by plenty of different modules and libraries that are used for scientific and mathematical purposes, that will be good for implementation of the recommender algorithm.

3.4.3 Django

Django is a modern high-level framework that was built using Python language. It is simple, robust, flexible, and allows to design solutions without much overhead [40]. Django provides high-level abstractions of common Web-development patterns, shortcuts for frequent programming tasks, and clear conventions on how to solve problems [41], so it's no need in reinventing the wheel, just using prepared solutions. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks – right out of the box. Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords [42].

The application will be developed with Django framework because it allows building complex and effective web applications in a short period. Moreover, as it was mentioned, Django has many solutions to handle common web development tasks, and at the same time, it lets working outside the scope of the framework as needed.

3.4.4 REST

REpresentational State Transfer (REST) is an architectural style for building large-scale distributed hypermedia systems. It provides some standards between computer systems on the web, making it easier for systems to communicate with each other. Web services that were developing according to the REST architectural style are identified as RESTful Web services (RWS).

The REST architectural style is based on the following constrains [43]:

- *Client-server architecture.* The main principle behind this constraint is the separation of concerns. It allows for the separation of front-end code (representation and possible UI-related processing of the information) from the server side code, which should take care of storage and server-side processing of the data.

  This constraint allows for the independent evolution of both components, offering a great deal of flexibility by letting client applications improve without affecting the server code and vice-versa.

- *Stateless.* Communication between client and server must be stateless, meaning that each request done from the client must have all the infor-

mation required for the server to understand it, without taking advantage of any stored data.

- *Cacheable.* It proposes that every response to a request must be explicitly or implicitly set as cacheable.

- *Uniform Interface.* It is one of REST's main characteristics. By keeping a uniform interface between components, it simplifies the job of the client when it comes to interacting with the system. Another major winning point here is that the client's implementation is independent, so by defining a standard and uniform interface for all of the services, it is effectively simplified the implementation of independent clients by giving them a clear set of rules to follow.

- *Layered System.* By separating components into layers, and allowing each layer to only use the one below and to communicate its output to the one above, it simplifies the system's overall complexity and keeps component coupling in check. This is a great benefit in all type of systems, especially when the complexity of such a system is ever-growing (e.g., systems with massive amounts of clients, systems that are currently evolving, etc.)

- *Code-on-Demand.* It is the only optional constraint imposed by REST. With this constraint, the client can download and execute code provided by the server (such as Java applets, JavaScript scripts, etc.). In the case of REST APIs, this constraint seems unnecessary, because the normal thing for an API client to do is just get information from an endpoint, and then process it however needed; but for other uses of REST, like web servers, a client (i.e., a browser) will probably benefit from this constraint.

The developing application will use the REST API for communication between client and server sides of application because it is very flexible and allows the independent implementation of the client and server.

3.5 Client-side technologies

In this section will be discussed all technologies that will be used on a client-side of application.

On the client-side, application will be developed as JavaScript single-page application (SPA). It will be built created with the React framework, and it will use the Bootstrap framework for styling a user interface.

3.5.1 Single-page application

A single page application (SPA) is a web application that uses only one HTML web page as a shell for all the application's web pages and whose end-user interactions are implemented by using JavaScript, HTML, and CSS [44]. Most of the SPA development is performed on the front-end in contrast to traditional web applications. The SPA fully loads all of the resources in the initial request, and then due to user interactions, an application replaces and updates page components. It is no need to reload new web pages whenever navigation occurs.

SPA allows a more flexible and elegant way of dealing with data. Refreshing particular part or a section of a page without refreshing an entire page is the primary goal that SPA is serving, but all this flexibility requires a more interactive interface, and this leads to the better user experience [45].

SPAs are also very responsive in terms of server interaction. All the operations that go to the server are performed using Ajax and therefore the user interface can still receive events and won't be stuck.

3.5.2 React

React is a full-scale Javascript framework. React was built to deal with displaying data in a user interface.

React does not set out to solve every problem that will appears in user interface design and front-end development. React solves a specific set of problems, and in general, a single problem. React builds large-scale user interfaces with data that changes over time [46].

React is in its simplest form, just the view of Model-View-Controller frameworks. React is a way to describe the user interface of an application and a mechanism to change that over time as data changes. React is made with declarative components that describe an interface. React uses no observable data binding when building an application. React is also easy to manipulate because the developer can take the components he creates and combines them to make custom components that work as he expects every time because it can scale. React can scale better than other frameworks because of the principles that drove it from its creation. When creating React interfaces, developer structures them in such a way that they are built out of multiple components.

3.5.3 Bootstrap

Bootstrap is a well-known open-source frontend framework in the world. It includes HTML, CSS and JavaScript components.

Bootstrap uses a 12-column responsive grid system and features, dozens of components, styles and JavaScript plugins. In addition, it has a basic global display, typography and link styles. Moreover, by using the customizer Bootstrap can be suited for a specific web project by adjusting variables,

components, JavaScript plugins and more. It is possible to expand Bootstrap by using a wealth of resources, including themes and interfaces and building tools. Moreover, Bootstrap is used for building mobile first responsive websites of all size and complexity. It has ready to use responsive themes and templates as a starting point for any web project [47]. All of this allows frontend web development to be catapulted forward, building on a stable foundation of forward-looking design and development [48].

This will be used for application development because it allows saving time on the styling of user interface components and in that case paying more attention to the application functionality.

3.6 User Interface

The main aim of a user interface is to provide easy and effective communication between a user and an application. A successful user interface should be

- intuitive: not require training to operate
- efficient: not create additional or unnecessary friction
- user-friendly: be enjoyable to use

Before designing the low-fi prototype for the web application, it is crucial to define tasks that application might deal with and relationships between them.

3.6.1 Task List and Task Graph

Task list is using for determining the actions that will application perform to meet defined functional requirements.

We will focus only on the user's tasks. The administrator's tasks do not need to develop special interface because they will be used the standard django CMS admin page, which allows the administrator to make all the necessary content manipulations, for example adding new POI to the application, updating information about POI, deleting POI, etc.

All the user's tasks that the application should perform are listed below. These tasks were divided into four groups.

The first group contains all tasks referred to the basic user operations for managing and displaying user profile. The needed tasks are mentioned below:

- Registration of a new user in the application
- Log in to the application
- Log out from the application
- Show user profile:

  - Display user name, avatar, username, email, date of the registration in the application, average rating for visited POIs, amount of written reviews.

- Edit user profile:
  - Change user avatar and email address.

The second group includes tasks that will be performed from the first minute of working with the application. These tasks are related to the home page of the application. All tasks are listed below:

- Show home page
- Get city of user location
- Show top POI categories of the user location city
- Show top POIs per each of top categories of the user location city ranged by rating
- Search POIs by categories and cities

The third group has all tasks that will deal with the primary function of the application – recommending POIs. It covers searching the best POIs for users and ordering them by rating (for unauthorized users) or calculated recommended score (for authorized users). The tasks are the following:

- Show searched POIs ordered by rating or recommended score
- Choose another category for searching POIs
- Filter POIs by price range or POI attribute
- Show all/filtered POIs on the map
- Choose POI to see details

To the fourth group brings together tasks that are responsible for the displaying POI details and allow users to impact on this information, by leaving the feedback. The tasks are listed below:

- Show POI basic details
  - Display POI name, address, categories, opening hours, attributes
- Show POI photo
- Show POI average rating
- Show amount of written reviews for POI

- Show POI on a map
- Show also recommended POIs from the same category and city
- Show POI reviews and ratings left by other users ordered by the date
- Show POI tips left by other users ordered by the date
- Show info about user who left review or tip
  - Display user name, avatar, amount of written reviews
- Write reviews and rate visited POI
- Write tip for visited POI

Next important step for designing user interface is making a task graph. Task graph describes the application structure and actions which application may perform.

Figure 3.4 represents the task graph of the developing application. It contains mentioned tasks and divides them into groups that were discussed before: user profile group, home page group, search result group and POI details group. Each of this group will represent the web page in the application.

This graph represents the application from an authorized user view because the only difference in actions between authorized and unauthorized is that an authorized user can write tips, reviews and rate visited POIs while an unauthorized user can only read others users' feedback.

The work with the application starts from the "Landing" point. Here user can register to the application, or if he already has an account, he can log in. Logged in user can watch his profile and add or change some information there.

On the home page application will detect the city of user location and suggest him the best POIs ordered by rating from top categories in this discovered city. So user can choose one POI of the recommended list, or he can search some POIs by category in a particular city.

Then a user will get his search result – a list of POIs ordered by rating or recommending score (for authorized users). All founded POIs will be displayed on the map. User can change category or filter POIs by price range or its attributes. After finding attractive POI in the list, a user will have an opportunity to see details about selected POI.

When a user selects POI, he can read information about it: address, categories, price range, opening hours, characteristics. POI location will be displayed on map. Also, it will provide a photo of this POI, its average rating and amount of written reviews. User can read all reviews and tips written by other users, and if he is authorized, he can leave his own opinion and rating for the POI he had already visited.

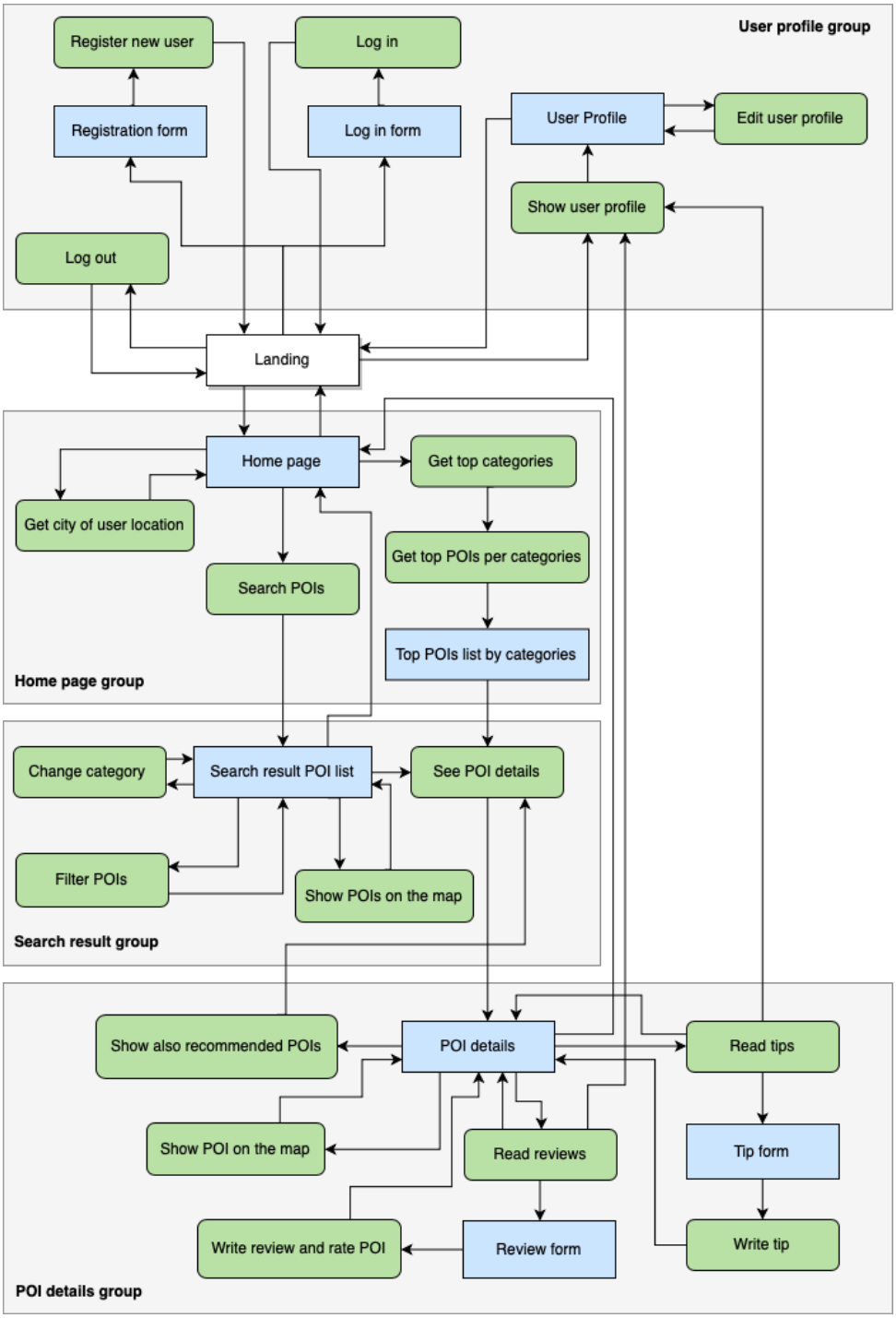


Figure 3.4: Application task graph

If a user does not like this POI, he can choose another from the ”also recommended POIs” list and read information about other selected POI.

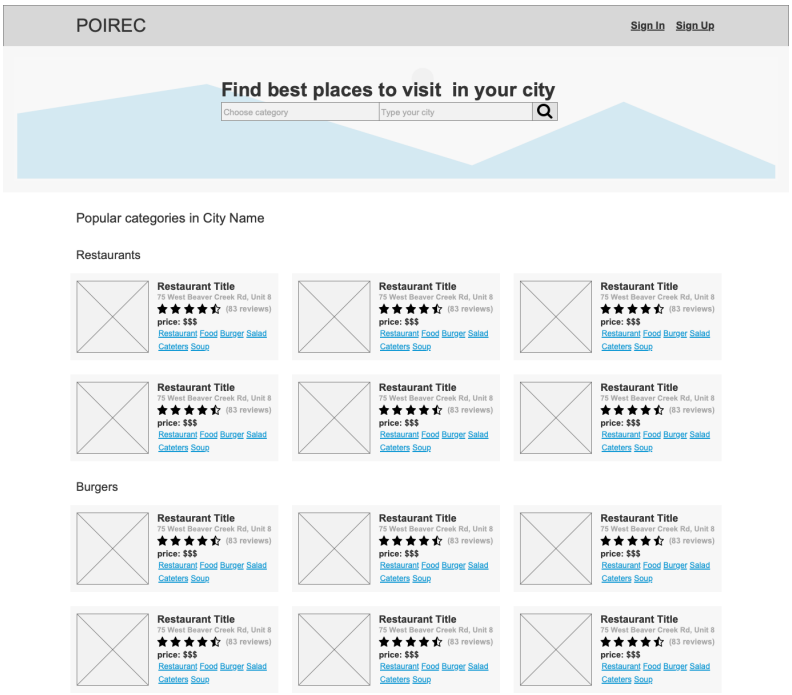


Figure 3.5: Home page prototype

3.6.2 Low-fi prototype

Based on the determined task list and task graph was designed the low-fi prototype for the developing application.

The application will contain the following pages:

- home page
- user profile page
- page with the list of POIs that are recommended for visiting
- page with the POI's details

The low-fi prototype of the application pages was designed in the Axure PR 8.

3.6.2.1 Home page

Figure 3.5 represents the prototype of the *home page* of the developing web application.

The navigation bar will be displayed on all application pages. It will be sticky and will be situated at the top of the page. The logo of the application with the link to the home page will be represented on this navigation bar. Unauthorized user will see two buttons on the navigation bar for registration and logging in. After clicking on one of these buttons will open the modal with the form for registration or logging in. If a user is authorized instead of the described buttons, he will see user icon after clicking on it will appear a dropdown menu with a link to profile setting and link for log out action.

On the home page will be located section for searching POIs. It will provide two inputs with the search button. In one input user will choose the category, which he is looking for, from the suggested list, and in another user – city. The home page also will have the section with the suggested lists of POIs by categories. There will be represented five of the most popular categories in the city of user location. Each of these categories will contain the list of the best POIs. Every element of a list will be represented as "POI card". This card will include basic POI information: name, rating, amount of reviews and categories it belongs, and it will be a link to the certain POI detail page.

3.6.2.2 User profile page

The prototype on the Figure 3.6 represents the *user profile page*.

This page will display information about a user: name, username, avatar, email address, date of registration in the application, average rating and amount of written reviews. User can see his own profile page or profile page of any other registered user. If it is user's own profile page, he will see a button for editing the profile information. If he clicks on the button, it will appear editing section, which is described on the Figure 3.7. This section has a form for changing avatar by uploading a new file and changing email address. Also, there is a link to go back to the user information.

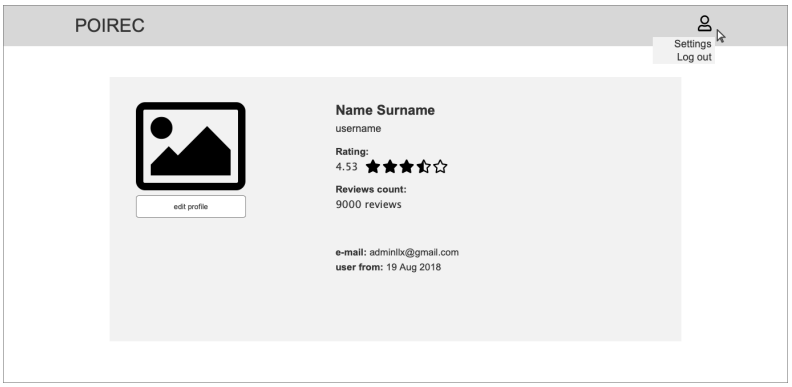


Figure 3.6: User profile page prototype

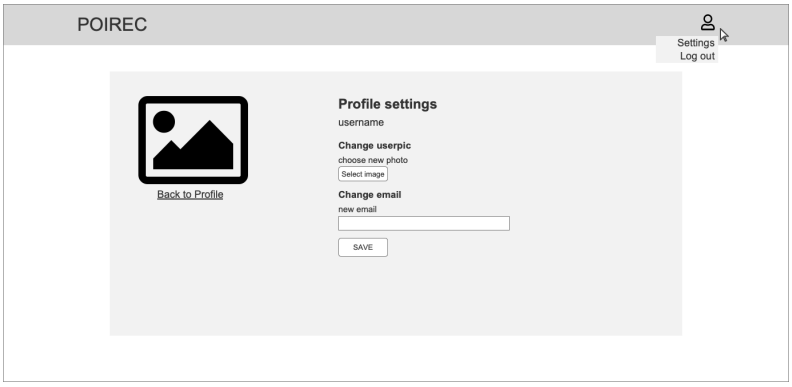


Figure 3.7: Editing user profile page prototype

3.6.2.3 Search result page

Figure 3.8 illustrates *search result page*.

This page is divided into three columns. The centre column contains a list of POIs with "POI cards". The right column displays a map with the markers of each POI that is listed in the searching result. And the left column represents filters for filtering POI list. Filters are classified into three groups: price range, categories and attributes. User can pick only one option from the price range and categories filter, but from the attributes filter, he can choose

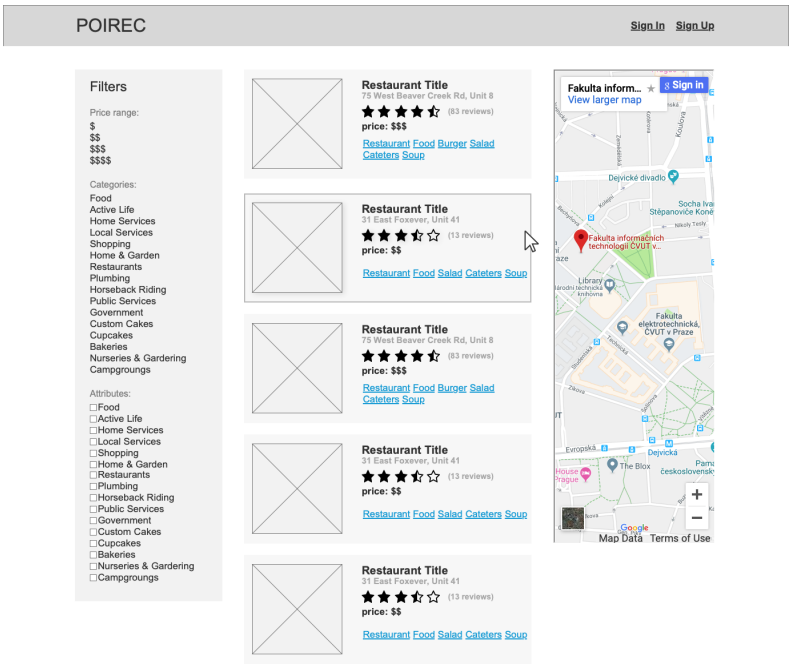


Figure 3.8: Search result page

any amount of options.

3.6.2.4 POI detail page

The last of the mentioned pages – *POI detail page* is represented on the Figure 3.9.

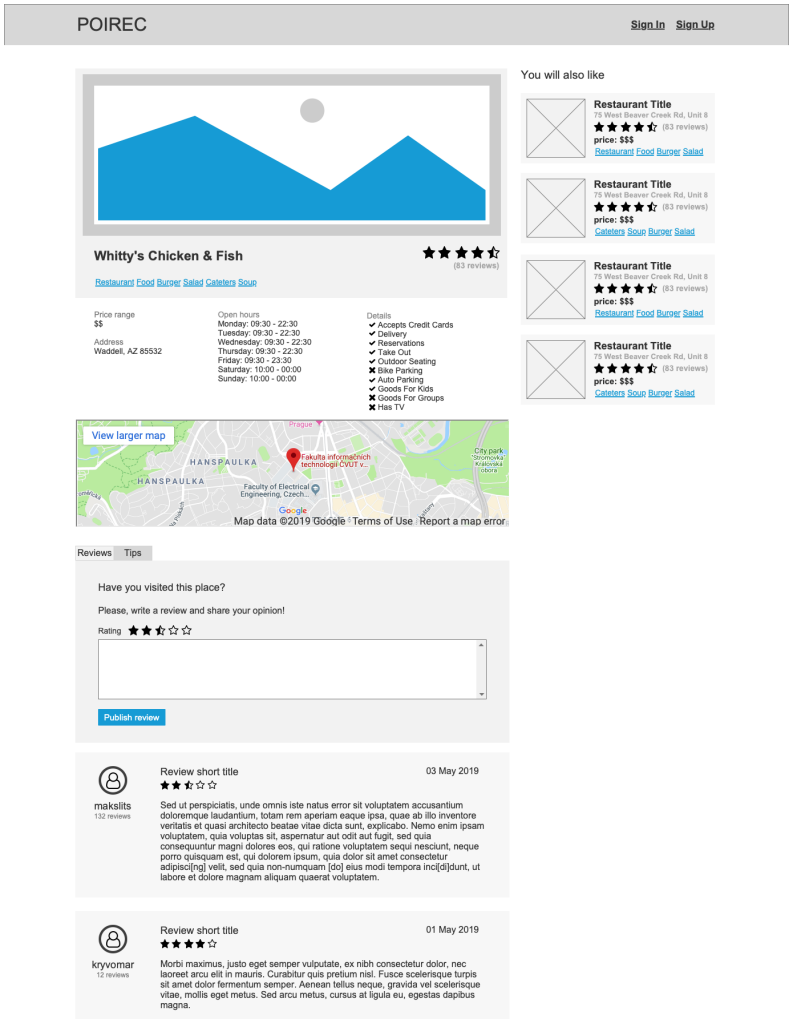


Figure 3.9: POI detail page

This page displays all information about selected POI: name, address, categories, opening hours, etc. Also, there is a section with the big photo of the POI and a map section, where the marker displays the POI location on the map. On this page is situated reviews and tips section. User can switch tabs for reading reviews or tips about POI. Reviews and tips have the same view with one difference: reviews also have ratings from the users. Each review/tip has a text of review/tip, date when it was posted and information about the

user who wrote it: avatar, name and amount of written reviews. Name of the author of review/tip is a link to his profile.

If a user is authorized, he can see review and tip form. After filling the form, he can post a new review/tip. Additionally, there is an "also recommended POIs" section in the right part of the page. There displays a list of POIs that can likewise be interested for the user.

Chapter 4

Implementation

Implementation of the application was divided into two parts: backend and frontend implementation. Each of them will be described in the following sections.

4.1 Backend implementation

In this section will be discussed the implementation of the backend part of the application. As it was mentioned before in the design chapter, the backend part was developed with Django framework, so the implementation was coming out of opportunities of this framework.

4.1.1 Models

Django framework has an object-relational mapper in which the developer describes the database layout in Python code. A model is the single, definitive source of information about the data. It contains the essential fields and behaviours of the storing data. Generally, each model maps to a single database table. Each model is a Python class that subclasses `django.db.models.Model`. Each attribute of the model represents a single database field. With all of this, Django provides an automatically-generated database-access API [42].

Models that are used in the application were created based on the database structure defined in the design chapter. Each table that was described has its own appropriate model with the same structure. Models have the same fields and fields types that were listed in the tables above. All created models are related to each other due to figure 3.3 which illustrates the database entity relationships in the previous chapter. Each model is responsible for the relevant database table structure and data that it stores. Models are synchronised with the database tables.

4.1.2 Data uploading

As it was mention in the design chapter, the application uses Yelp dataset. It is an open dataset that was downloaded from the yelp.com website. It contains data about POIs and users that are written in the JSON format. For loading this data to the database were used Django fixtures.

A fixture is a collection of data that django knows how to import into a database. Fixtures can be written as JSON, XML or YAML documents [42].

For the transforming Yelp dataset to the fixtures was created a script. This script generates a fixture in the JSON format for each model that is used in the application. Each of the created fixture contains data only for one model and represent an array of objects, that describes one object in the database table. Each fixture object includes model name, primary key and fields values.

4.1.3 Administration page

There was no need to develop separate pages for administrative purpose because Django provides it. When the models are defined, Django can automatically create a professional, production-ready administrative interface – a website for managing content [42]. It lets authenticated users add, change and delete objects.

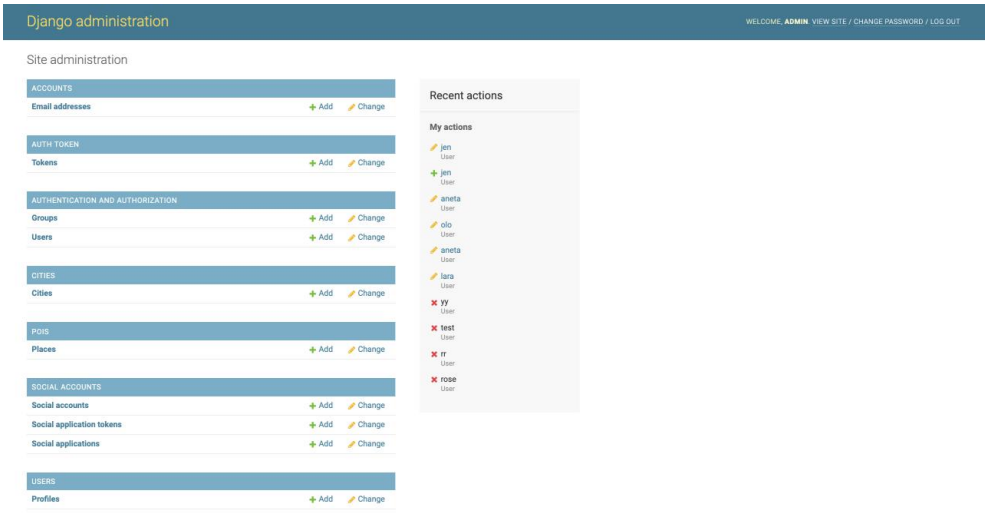


Figure 4.1: Administration page

The user who has administrator permissions can log in to an administration page, that is described on the Figure 4.1, and work with the stored content. Models that are connected to the administration page are ”Users”, ”Cities” and ”POIs”. So an administrator can add new POIs to the application, or update the information about existing POIs, or if POI doesn't exist anymore, he can easily delete it from the application. If the administrator needs to

add the POI from the city, which hasn't been in the application yet, he can add a new city and connected created POI to this city. Furthermore, the administrator can manage users. He can change their roles from regular user to the administrator of the application.

In the application was created one user with the administrator permissions. There are two ways of creating administrators in the application. The first one, the existed administrator can add administrator role to any of the registered users. And the second one, it can be done by using the command from the terminal, that can be found in the installation guide in Appendix C.

4.1.4 Recommender algorithm

For the implementation of the algorithm, that makes POIs recommendations for a user, based on his preferences, were used external libraries and packages. They are NumPy, pandas and Surprise.

4.1.4.1 NumPy

NumPy is the fundamental package needed for scientific computing with Python. This package contains [49] a powerful N-dimensional array object, sophisticated (broadcasting) functions, basic linear algebra functions, sophisticated random number capabilities and many other things.

In the developed web application, NumPy is used in the implementation of two parts of the recommender algorithm: the item-based collaborative filtering part and singular-value decomposition method part. In these parts, NumPy is mainly used for storing data as an N-dimensional array object, because the chosen methods need to work with the data stored in a matrix and to perform operations on this matrix, and NumPy package provides these functionality.

4.1.4.2 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. Pandas is well suited for many different kinds of data: tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet, ordered and unordered (not necessarily fixed-frequency) time series data, arbitrary matrix data with row and column labels, any other form of observational/statistical data sets. The two primary data structures of pandas, `Series` (1-dimensional) and `DataFrame` (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. Pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries [50].

Pandas package is used in the implementation of the content-based filtering part of the designed recommender algorithm. In this method, it is suitable

to use labelled columns for storing and working with data. `DataFrame` stores user's reviews and using this `DataFrame`, program creates new `DataFrames` for user profile and POIs feature vectors. Pandas provides many functions on its `DataFrame` which are useful and save much time in developing.

4.1.4.3 Surprise

Surprise is a Python scikit building and analyzing recommender systems. Surprise gives users perfect control over their experiments, provides various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others, also, various similarity measures (cosine, pearson...) are built-in [51].

Surprise is used in the implementation of the item-based collaborative filtering part and singular-value decomposition method part. It has a built-in collaborative filtering algorithm, that is called `KNNBasic()`, and can be set by parameters. In our case, it has set parameters to perform item-based collaborative filtering and use Pearson correlation similarity measure. Also, Surprise has a built-in singular-value decomposition method – `SVD()`. So the advantage of using Suprise that it is no need to develope the algorithms from the beginning. Surprise provides optimized and ready to use algorithms, so the developer can focus only on using and interpretating the results.

4.1.5 User authorization

For the registration of new users and their authorization, in the application is used django-allauth package. It allows for both local and social authentication and supports multiple authentication schemes (e.g. login by user name, or by e-mail), as well as multiple strategies for account verification (ranging from none to e-mail verification) [52].

The application uses only local authentication. As the authentication scheme is used login by username. For the registration of new user it requires next user data: username, email, password, first name and last name.

4.1.6 REST API

For developing the application REST API was used Django REST framework. It is a powerful and flexible toolkit for building Web APIs [42].

For the application was developed REST API. There are described only that API endpoints which are used by application on frontend.

- **GET cities/**

  Method that returns information about all cities stored in the application.

  Returns codes: 200 OK, 404 Not Found

- **GET cities/{city}/pois/**

  Method that returns list of all POIs that are located in the chosen city {city}.

  Returns codes: 200 OK, 404 Not Found

- **GET cities/{city}/categories/**

  Method that returns a list of objects for categories to which the POIs that situated in the selected city {city} belong.

  Returns codes: 200 OK, 404 Not Found

- **GET cities/{city}/categories/{category}/pois/**

  Method that returns information about all POIs that are located in the city {city} and belong to the picked category {category}.

  Returns codes: 200 OK, 404 Not Found

- **GET categories/**

  Method that returns information about all categories that POIs can belong.

  Returns codes: 200 OK, 404 Not Found

- **GET categories/{category}**

  Method that returns detailed information about category {category}: name and its subcategories.

  Returns codes: 200 OK, 404 Not Found

- **GET pois/{poiId}**

  Method that returns detailed information about selected POI that has defined id {poiId}.

  Returns codes: 200 OK, 404 Not Found

- **GET pois/{poiId}/photos/**

  Method that returns photos of selected POI that has defined id {poiId}.

  Returns codes: 200 OK, 404 Not Found

- **GET pois/{poiId}/reviews/**

  Method that returns all reviews that were written by users about selected POI that has defined id {poiId}.

  Returns codes: 200 OK, 404 Not Found

- **POST pois/{poiId}/reviews/**

  Method that creates new review object for selected POI with id {poiId} and adds it to database. Body of request is a JSON object with data that contains user username, text of review, rating stars and POI id, and response with the created review object.

  Returns codes: 201 Created, 400 Bad Request

- **GET pois/{poiId}/tips/**

  Method that returns all tips that were written by users about selected POI that has defined id {poiId}.

  Returns codes: 200 OK, 404 Not Found

- **POST pois/{poiId}/tips/**

  Method that creates new tip object for selected POI with id {poiId} and adds it to database. Body of request is a JSON object with data that contains user username, text of review and POI id, and response with the created tip object.

  Returns codes: 201 Created, 400 Bad Request

- **GET user-profiles/{username}**

  Method that returns information about registered user who has username {username}.

  Returns codes: 200 OK, 404 Not Found

- **PUT user-profiles/{username}**

  Method that returns information about registered user who has username {username}. Body of request is a JSON object with data that contains user username, updated avatar and new email address.

  Returns codes: 202 Accepted, 400 Bad Request

- **GET user-profiles/{username}/cities/{city}/categories/**

  /{category}/pois/

  Method that returns list of POIs, that are recommended to the user with the username {username} based on his preferences. All of POIs are located in the city {city} and belong to the category {category}.

  Returns codes: 200 OK, 404 Not Found

4.2 Frontend implementation

In this section will be described the implementation of the frontend part of the application. As it was mentioned before in the design chapter, the frontend part was developed with React framework, so the implementation was coming out of opportunities of this framework.

4.2.1 Bootstrap

For building styled UI components was used Bootstrap framework as it was described before. For implementing Bootstrap was used a package called React Bootstrap. It is a version of the most popular front-end framework [53] that is rebuilt for React. React Bootstrap replaces the Bootstrap javascript. Each component has been built from scratch as true React components, without unneeded dependencies like jQuery.

Also, the application uses a free theme for Bootstrap, which was downloaded from Bootswatch [54] and called Yeti. It is a CSS file with rules that describes views of the components for the entire application in one style. This file connects to the main file with the application styles. The used theme was a little bit customized by changing the colour scheme.

The most significant benefit of using Bootstrap is that it has a great grid system. It allows to position UI elements in a fast and easy way. Furthermore, it takes care of responsive views of the page, so there were no need to waste a lot of time on writing special media queries for every breakpoint. So for the making application responsible, it was written only some extra media queries, the main part of them were included in Bootstrap libraries.

4.2.2 Routing

On the UI design stage was defined that application will have four primary pages: home page, user profile page, search result page and POI detail page. The developed application is a SPA, and for navigation between created pages, it uses the React Router package [55]. There are three types of components in React Router: router components, route matching components, and navigation components. So route matching is done by comparing a <Route>'s path prop to the current location's pathname. When a <Route> matches, it will render its content.

The web application has four routes:

- **/**

  When this route matches, it renders the component, that represents the main page of the application.

- **/users/:username**

When this route matches, it renders the component, that represents the user profile page of the application, and it is filled with information about the user with username `:username`. For opening the editing section on a page, it just changes rendering options of components inside of the page component.

- **/:city/categories/:category/places**

  When this route matches, it renders the component, that represents the search result page of the application. The components on this page represent information about POI in the selected city `:city` and chosen category `:category`.

- **/places/:placeId**

  When this route matches, it renders the component, that represents the POI detail page of the application. This component displays information about the selected POI that has an id `:placeId`.

4.2.3 State management

Application React components might have a lot of states and share it. Managing the state of each component is painful. So the best idea is to handle all the states from one place. For these purposes, the application uses the Redux library.

Redux [56] is a predictable state container for JavaScript apps. It helps to write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

Due to Redux principles [56] the state of the whole application is stored in a single store. For changing the state are used actions – objects describing what happened. For specifying how does the state change are used reducers. Reducers are just pure functions that take the previous state and an action and return the next state.

In the application, the main reducer splits into three small reducers: `auth`, `pois` and `cities`.

The `auth` reducer stores users authentication data – username, token, and an error if authentication fails.

The `pois` reducer stores data about POIs which are registered at application. It stores POIs list that is fetched from the backend, filtered POIs list and an error if it occurs during fetching data from the backend.

The `cities` reducer stores data about cities existed in the application. It stores cities list fetched from the backend, and an error if it occurs during fetching data from the backend.

Any of the developed components can access and update all this stored data at any time it needs.

4.2.4 Communication with server

All client-server communications in the developed application were build on Ajax. For providing communications were used `axios` [57] library. It is the promise based HTTP client for the browser and node.js. The library has predefined methods that allow making requests and getting responses easier and quicker. Also, it provides automatic transforms for JSON data and client-side support for protecting against Cross-site Request Forgery (CSRF).

4.2.5 Third-party API

One of the functional requirement was displaying POIs on the map, for this purpose was used Google Maps API [58]. It allows to add an interactive map to the web application and customize it with content and imagery. To facilitate the work with Google Maps API in React components was used google-maps-react package. It provides the declarative Google Map React component that can be customized via properties and components it wraps.

Another application requirement was detecting the city of user location. For this purpose also was used Google Maps API. The application sends a request to the API that contains user's geolocation coordinates, API key and language of received data, and it responses with the name of the city which corresponds to the sent geolocation coordinates. Coordinates are taken from the browser navigator object.

4.2.6 User interface

The user interface of the application was developed based on the low-fi prototype, that was created and described in the design chapter. As it was discussed before, the application contains four main screens: home page, user profile page, search result page and POI detail page. The developed user interface is represented on the following Figures 4.2 - 4.5.

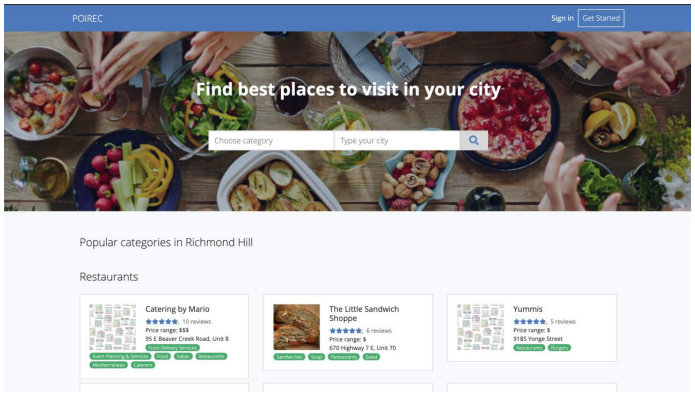


Figure 4.2: Home page

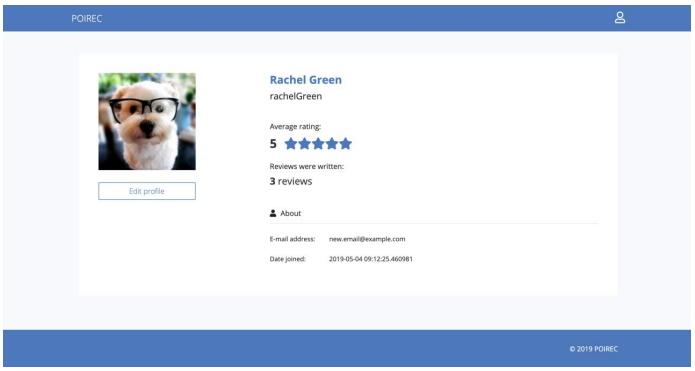


Figure 4.3: User profile page

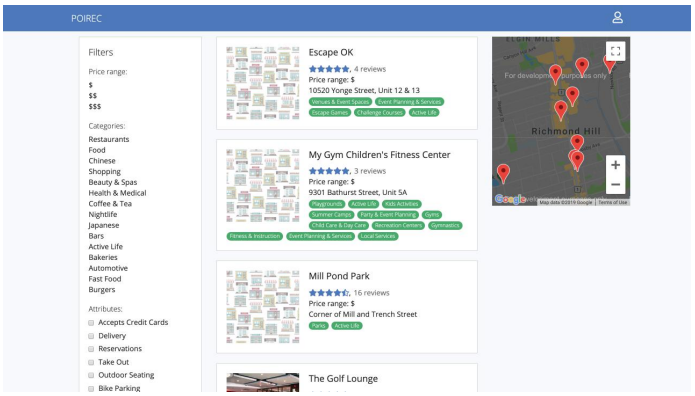


Figure 4.4: Search result page

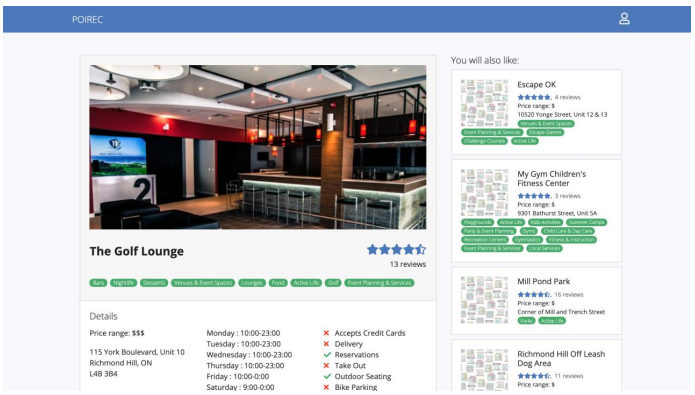


Figure 4.5: POI detail page

Chapter 5

Testing

Testing is one of the most critical parts of application development. It will point out the errors that occur during the development phases. Testing makes sure that the application performs all required functions and users can work with the application without any problems.

Testing of the developed application consists of two parts: recommender algorithm evaluation and user testing. Each of these parts is described in details below.

5.1 Recommender algorithm evaluation

The designed hybrid algorithm that is used in the application from making recommendations for a user was evaluated with two types of metrics: classification accuracy metrics and rank accuracy metrics. The evaluation with predictive accuracy metrics is impossible because the algorithm returns only the rank list of POIs and doesn't predict the rating of POI.

For performing the algorithm evaluation were selected three different users from the existing dataset. The first one has the highest number of posted reviews, and another two have posted only some reviews about POIs they visited. For each of these users were created user's profile that allows logging in the application via username and password.

Written reviews by the selected users were divided into two equal parts. The first part was used for training the algorithm and the second part – for testing. The algorithm was tested for the city where users have the highest number of written reviews, and for the city where they have posted only some reviews.

The hybrid algorithm evaluation results also were compared with the evaluation results of individual methods of which the final algorithm consists and with the list of POIs that is ordered by rating.

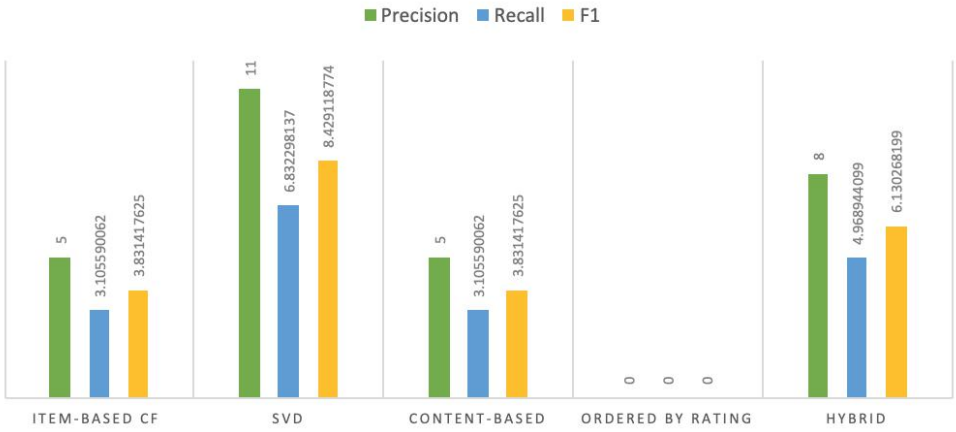


Figure 5.1: Evaluation results for the fist user and Toronto city

5.1.1 Evaluation with classification accuracy metrics

For the evaluation with classification accuracy metrics was used *Precision and Recall* metric.

As it was mentioned before, the algorithm was tested on the data from three users. For the measuring were used first 100 items from the recommendation list.

The first user has written the highest amount of reviews – 4130 reviews. For him were selected two cities: Toronto and Thornhill. For POIs that located in the Toronto city, he has left the highest amount of reviews – 1716 reviews, and for POIs located in Thornhill – 110 reviews.

The diagram on the Figure 5.1 displays the results of the used algorithm evaluation with Precision and Recall metric for the Toronto city and evaluation

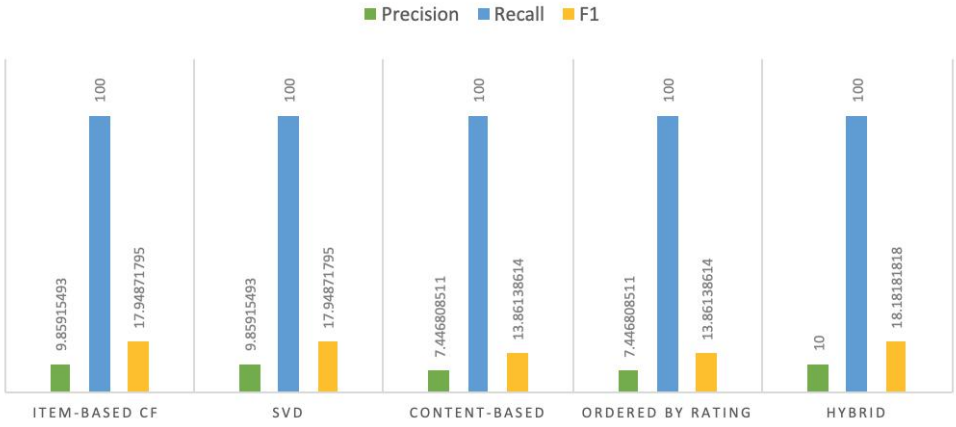


Figure 5.2: Evaluation results for the first user and Thornhill city

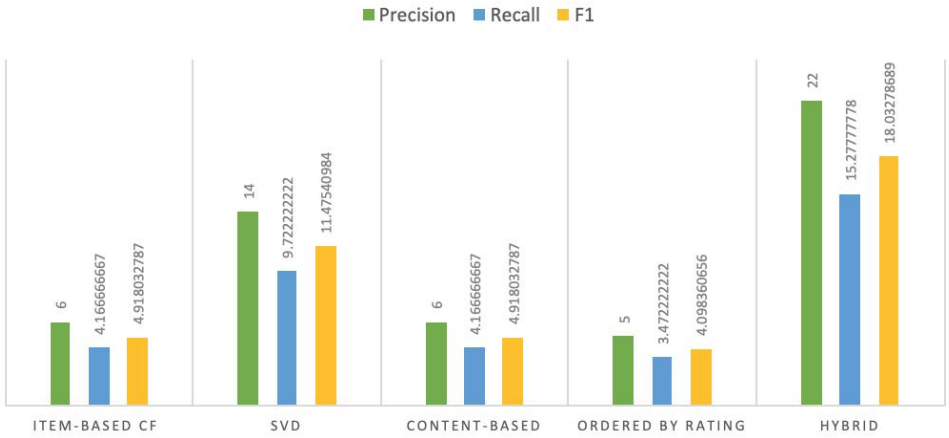


Figure 5.3: Evaluation results for the second user and Las Vegas city

results presented by other algorithms.

The diagram on the Figure 5.2 displays the evaluation results for the Thornhill city.

The second selected user has written 1559 reviews. For him were selected two cities: Las Vegas and Scottsdale. The highest total number of reviews he has posted for POIs located in Las Vegas – 1070 reviews and in the Scottsdale he has rated only 64 POIs.

The Figure 5.3 and the Figure 5.4 represent the diagrams with the evaluation of algorithms for the second user for the Las Vegas city and Scottsdale city respectively.

The third selected user has posted only 499 reviews. Scottsdale city and

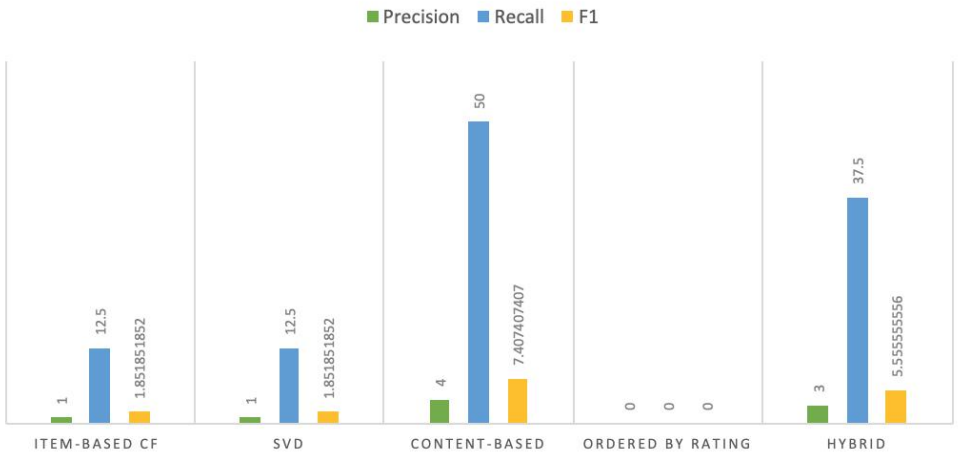


Figure 5.4: Evaluation results for the second user and Scottsdale city

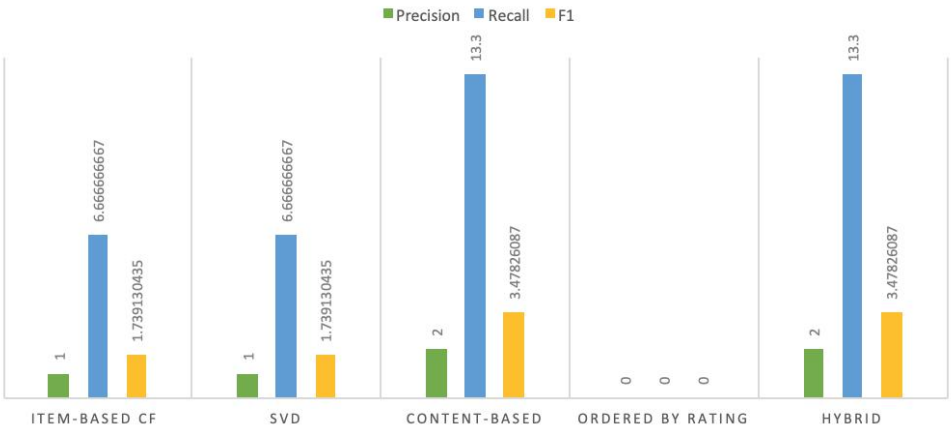


Figure 5.5: Evaluation results for the third user and Scottsdale city

Tempe city were selected for testing of the algorithms. In the Scottsdale, he has rated 232 POIs, and in Tempe city, he has visited 39 POIs.

The Figure 5.5 and the Figure 5.6 describe the diagrams with the evaluation of algorithms for the third selected user and cities Scottsdale and Tempe respectively.

To sum up, based on the received results from the evaluation of the algorithms with the *Precision and Recall* metric, the implemented hybrid algorithm works pretty well in case when the user has rated more POIs from one city. In case when the user has rated only a few POIs in the web application, the algorithm will give user a list with a higher amount of relevant POIs than recommendation list ordered by rating, but results will be close to the content-based filtering method.

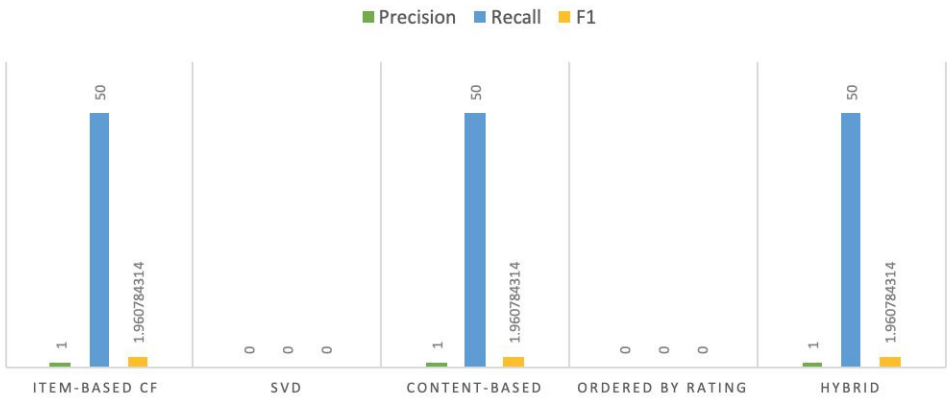


Figure 5.6: Evaluation results for the third user and Tempe city

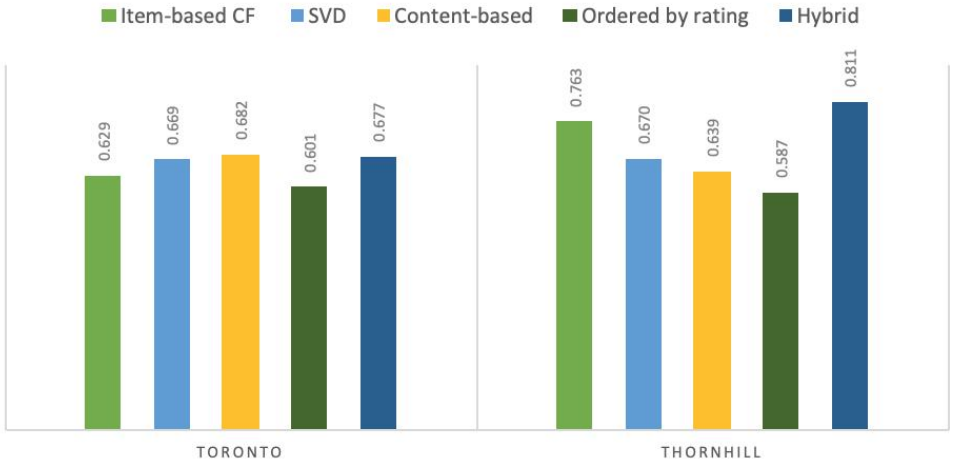


Figure 5.7: Evaluation results for the first user

5.1.2 Evaluation with rank accuracy metrics

For the evaluation with rank accuracy metrics was used *NDCG* metric.

The evaluation of the algorithms with the NDCG metric was made on the same data that was used and described in the previous section. Furthermore, the evaluation results from the implemented hybrid algorithms were compared with the result from the other methods: item-based collaborative filtering, singular-value decomposition method, content-based filtering and recommendations by rating.

Results of the evaluation of the algorithms on data from the first selected user are displayed on Figure 5.7. It represents NDCG values for every tested

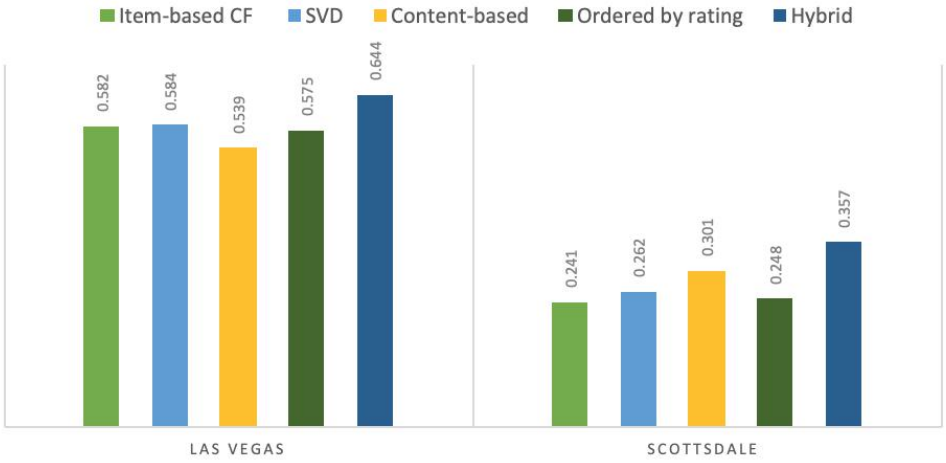


Figure 5.8: Evaluation results for the second user

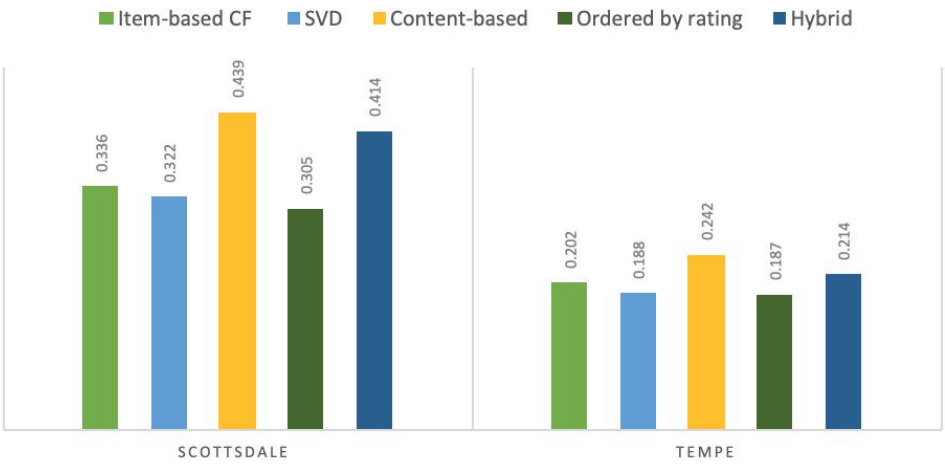


Figure 5.9: Evaluation results for the third user

algorithm in two cities: Toronto and Thornhill.

The Figure 5.8 represents the diagrams with the results of the computing NDCG values form data about the second user on cities: Las Vegas and Scottsdale.

The results from the evaluation of algorithms using the NDCG metric on the data from the last third user is illustrated on Figure 5.9. It displays results for two cities: Scottsdale and Tempe.

So, in conclusion, the implemented hybrid algorithm gives good results. It works better when the user has a great user profile (has rated a lot of POIs); in this case, it works better than other algorithms that were tested. But in the situation when it is a new user who just starts to use an application and has rated only a few POIs, the algorithm can rank the recommendations better than item-based collaborative filtering, singular-value decomposition method and recommendations by rating, but content-based filtering method ranks recommendations a little bit better.

5.1.3 Execution time

For each evaluation of the algorithm was measured the execution time. The measurement was made on laptop with a 2.7 GHz Intel Core i5 processor.

Execution time was measured by using the `time` built-in Python module. Results of time measurement are described in Figure 5.10. It displays time of running the computations for each selected user and cities that were chosen for measurement before. The time was measured in seconds.

The execution time of the used algorithm is increasing due to the size of a user profile. It is a problem, in case that users will not want to wait long for getting the results. There are some ideas that will help to reduce

Figure 5.10: Execution time

the algorithm execution time. The first one is to run the algorithm on more powerful hardware. The second one is to make the algorithm run in parallel. And the last one is to use query results caching on the database layer.

5.2 User testing

User testing is one of the primary methods of functionality and interface testing. The advantage of this testing is that you have feedback from potential users about how the developed application is convenient and intuitive for the user who works with it for the first time. Furthermore, a people who test it can suggest some improvements. The disadvantage is that this type of testing is very time-consuming.

5.2.1 Target audience

People from the target audience of the application always participate in user testing. The target audience for the developed application has no restrictions on age and gender. A potential application user is a person who is computer literate and uses the Internet for his own purposes (for example using applications, reading news, watching videos, etc.). The application focuses on people who love travelling or visiting and exploring new places in their home city. At this stage of the development, the application will be useful for people who live or planning to visit the USA or Canada because it stores POIs from these countries.

5.2.2 Testing process

Testing of the application is divided into three parts. At first, person, who takes part in application testing might answer the following pre-test questionnaire:

- How do you discover new places which you want to visit?
- Do you use some web sites/application that recommends you where to go?
- Have you ever participated in any usability testing?

After answering the questions, a person can go to the next part of the testing. He needs to complete tasks, that are described in the testing scenario, step by step. The tasks are:

1. Register in the application. Use your own information.
2. Log in the application with the data you provide on previous step.
3. Change information in your profile. Upload new avatar, use file avatar.png on the desktop. Change e-mail address.
4. Go back to your profile information and check if avatar and e-mail are changed.
5. Go to the main page.
6. Imagine that next week you are going to the city Richmond Hill. Find some good places to eat in this city.
7. Find expensive restaurants.
8. Choose one that you would like to visit.
9. Read information about this place. Read some reviews and tips.
10. Imagine that you have visited this place, write a review with your opinion.
11. Log out from the application.

Finally, after finishing performing tasks from the testing scenario, a person who tests the application might leave feedback about the application in the form of answering the following questions:

- What is your overall impression of the application?
- Was the test scenario clear?

- What did you like in the application?
- What didn't you like in the application?
- Do you have any improvement suggestions?

5.2.3 Testing results

The application was tested by five people. All their feedback is provided in Appendix B. Also, screen casts from the user testing are available on the attached CD disk.

Based on testers feedback, the application testing results are the following. The developed application has an intuitive and user-friendly interface. It is easy to use and find all necessary things and information that user need to know about POI he wants to visit. The application has basic functionality, but even so, it allows a user to specify his request by using filters.

Some shortcomings were detected during application testing. All of them will be fixed. The critical ones are

- The weird behaviour of the registration form. Sometimes it doesn't create a new user. There are no successful or error messages, so the user doesn't know if his registration was successful and he can log in to the application.

  The solution to this problem is adding successful/error messages and fixing the form validation.

- The name of the "registration" button might confuse users. Some of the testers at first click to the "log in" button instead of the "registration" button. One of them said that he was confused, and could not find the right one from the first sight.

  The solution is to rename the button from "Get started" to the "Registration".

- Search results return too many POIs, especially on the map. It causes difficulties for a user to explore POIs throw the map.

  The solution is to add pagination for the search results, and it will reduce the number of POIs described on the map at the same time.

Also, testers have suggested some improvements, that will be applied in the future. For example, adding the ability of searching POIs by several categories at the same time and adding photo gallery for POI, where users will be able to share their photos, which were made at the POI.

Conclusion

The main goal of this work was a design, implementation and evaluation of a proof of the concept web application acting as the Points Of Interests recommender system.

In the work were considered various types of recommender systems and investigated different methods, which can be used for making recommendations. It was designed and described the hybrid method which lies at the basis of the developed web application. Designed algorithm combined three different approaches: item-based collaborative filtering, singular-value decomposition method and content-based filtering. Combining of approaches allows to deal with shortcomings of each of these methods, especially it helps to avoid the cold-start problem.

Based on the designed hybrid algorithm was designed and implemented web application, that acts as the Points Of Interests recommender system.

The application was tested and had good feedback about the overall impression of the user interface, but it also was discovered some shortcomings, that are described in the previous chapter "Testing", all of them will be fixed.

Furthermore, the used hybrid algorithm was evaluated with two types of metrics: classification accuracy metrics and rank accuracy metrics. For this purpose were used three users with different user profile size. The results of the evaluation was that for the big user profile the designed algorithm returns better ranked recommendations than each of the algorithms it contains. For the case, when user is looking for the POIs in the city, where he hasn't rate any POI, the content-based filtering method works better. This information was taken into account and for this cases the developed application uses the content-based filtering method.

The developed web application that acts as the Points Of Interests recommender system satisfies all the requirements that was defined in the "Analysis" chapter. It uses the hybrid recommender algorithm that makes pretty good recommendations. So users of the web application can find new amazing places, that they will like, quickly and easily.

Bibliography

[1] Mahmood, T.; Ricci, F. Improving recommender systems with adaptive conversational strategies. *C. Cattuto, G. Ruffo, F. Menczer (eds.) Hypertext*, 2009: pp. 73–82.

[2] Cvetkoviıc, B.; Kaluž, B.; et al. Hybrid recommender system for personalized poi selection. *International Multiconference Information Society*, 2013.

[3] Resnick, P.; Varian, H. Recommender systems. *Communications of the ACM*, 1997: pp. 56–58.

[4] Bobadilla, J.; Ortega, F.; et al. Recommender systems survey. *Knowledge-Based Systems*, 2013.

[5] Aggarwal, C. C. *Recommender Systems: The Textbook*. IBM T.J. Watson Research Center Yorktown Heights, NY, USA, 2016, ISBN 9783319296579.

[6] Burke, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 2002.

[7] Ricci, F.; Rokach, L.; et al. *Recommender Systems Handbook*. Springer New York Dordrecht Heidelberg London, 2011, ISBN 9780387858197.

[8] Breese, J. S.; Heckerman, D.; et al. Emperical Analysis of Predictive Algorithms for Collaborative Filtering. *Microsoft Research*.

[9] Su, X.; Khoshgoftaar, T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009.

[10] Sarwar, B. M.; Karypis, G.; et al. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*, 2001.

[11] Rajaraman, A.; Leskovec, J.; et al. *Mining of Massive Datasets*. Cambridge University Press, 2011, ISBN 1107015359.

[12] Brusilovsky, P.; Kobsa, A.; et al. *The Adaptive Web. Methods and Strategies of Web Personalization*. Springer-Verlag Berlin Heidelberg, 2007, ISBN 0302-9743.

[13] Falk, P. *Practical Recommender Systems*. Manning Publications Co, 2019, ISBN 9781617292705.

[14] Felfernig, A.; Burke., R. Constraint-based recommender systems: technologies and research issues. *International conference on Electronic Commerce*, 2008.

[15] Bridge, D.; Goker, M.; et al. Case-based recommender systems. *The Knowledge Engineering Review*, 2005: pp. 315–320.

[16] Burke, R. Knowledge-based recommender systems. *Encyclopedia of library and information systems*, 2000: pp. 175–186.

[17] Herlocker, J. L.; Konstan, J. A.; et al. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 2004: pp. 5–53.

[18] Goldberg, K.; Roeder, T.; et al. Eigentaste: a constant time collaborative filtering algorithm. *Information Retrieval*, 2001: pp. 133–151.

[19] McNee, S. M.; Riedl, J.; et al. Accurate is not always good: how accuracy metrics have hurt recommender systems. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*, 2006.

[20] Carrer-Neto, W.; Maria Luisa Hernandez-Alcaraz, R. V.-G.; et al. Social knowledge-based recommender system. Application to the movies domain. *Expert Systems with Applications*, 2012.

[21] Robert Busa-Fekete, T. E. B. K., Gyorgy Szarvas. An apple-to-apple comparison of Learning-to-rank algorithms in terms of Normalized Discounted Cumulative Gain. *ECAI 2012 - 20th European Conference on Artificial Intelligence: Preference Learning: Problems and Applications in AI Workshop*, Aug 2012.

[22] Liu1, Y.; Pham, T.-A. N.; et al. An Experimental Evaluation of Point of interest Recommendation in Locationbased Social Networks. *Proceedings of the VLDB Endowment*, 2017: pp. 1010–1021.

[23] Li, X.; Cong, G.; et al. Rank-geofm: A ranking based geographical factorization method for point of interest recommendation. *SIGIR*, 2015: pp. 433–442.

[24] Gao, H.; Tang, J.; et al. Exploring Temporal Effects for Location Recommendation on Location-Based Social Networks. *RecSys*, 2013: pp. 93–100.

[25] Lian, D.; Zhao, C.; et al. Geomf: Joint geographical modeling and matrix factorization for point-of-interest recommendation. *KDD*, 2014: pp. 831–840.

[26] Zhang, J.-D.; Chow, C.-Y.; et al. Exploiting sequential influence for location recommendations. *SIGSPATIAL*, 2014: pp. 103–112.

[27] Pontes, T.; Vasconcelos, M.; et al. We Know Where You Live: Privacy Characterization of Foursquare Behavior. *UbiComp*, 2012: pp. 898–905.

[28] Luca, M. *Reviews, Reputation, and Revenue: The Case of Yelp.com.* Harvard Business School, 2016.

[29] Foursquare. About foursquare. 2011. Available from: `http://foursquare.com/about`

[30] Jeacle, I.; Carter, C. In TripAdvisor we trust: Rankings, calculative regimes and abstract systems. *Accounting, Organizations and Society*, 2011: pp. 293–309.

[31] Sarwar, B.; Karypis, G.; et al. Item-based Collaborative Filtering Recommendation Algorithms. *WWW10*, 2001.

[32] Zhou, X.; He, J.; et al. SVD-based incremental approaches for recommender systems. *Journal of Computer and System Sciences*, June 2015: pp. 717–733.

[33] W., B. M.; T., D. S.; et al. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 1995.

[34] Polat, H.; Du, W. SVD-based Collaborative Filtering with Privacy. *ACM Symposium on Applied Computing*, 2005.

[35] Martínez, L.; Pérez, L. G.; et al. A multigranular linguistic content-based recommendation model. *International Journal of Intelligent Systems*, 2007: pp. 419–434.

[36] Ivan Cantador, D. V., Alejandro Bellogin. Content-based recommendation in social tagging systems. *Proceedings of the 2010 ACM Conference on Recommender Systems*, 2010.

[37] Martinez, L.; Perez, L. G.; et al. What is PostgreSQL? *International Journal of Intelligent Systems*, 2007: pp. 419–434.

[38] Sanner, M. F. Python: a programming language for software integration and development. *J Mol Graph Model*, 1999.

[39] Oliphant, T. E. Python for Scientific Computing. *Computing in Science Engineering*, 2007: pp. 10–20.

[40] Forcier, J.; Bissex, P.; et al. *Python Web Development With Django*. Addison-Wesley Professional, 2008, ISBN 9780132701815.

[41] Forcier, J.; Bissex, P.; et al. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2009, ISBN 9780132701815.

[42] Django. Django documentation. Available from: `https://docs.djangoproject.com/en/1.8/`

[43] Doglio, F. *Pro REST API Development with Node.js*. Apress, 2015, ISBN 9781484209172.

[44] Fink, G.; Flatow, I.; et al. *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. Apress, 2014, ISBN 9781430266747.

[45] Jadhav, M. A.; Sawant, B. R.; et al. Single Page Application using AngularJS. *International Journal of Computer Science and Information Technologies*, 2007: pp. 10–20.

[46] Gackenheimer, C. *Introduction to React*. Apress, 2015, ISBN 9781484212455.

[47] Cochran, D. Twitter Bootstrap Web Development How-To. *Birmingham : Packt Publishing Ltd*, 2012.

[48] Spurlock, J. *Bootstrap*. O'Reilly Media, 2013, ISBN 9781449343910.

[49] NumPy. About NumPy. Available from: `https://www.numpy.org`

[50] Pandas. Pandas documentation. Available from: `http://pandas.pydata.org/pandas-docs/stable/`

[51] surprise. Surprise documentation. Available from: `https://surprise.readthedocs.io/en/stable/`

[52] django-allauth documentation. Available from: `https://django-allauth.readthedocs.io/en/latest/index.html`

[53] React Bootstrap documentation. Available from: `https://react-bootstrap.github.io/getting-started/introduction`

[54] Bootswatch: Free themes for Bootstrap. Available from: `https://bootswatch.com`

[55] React Router documentation. Available from: `https://reacttraining.com/react-router/`

[56] Redux documentation. Available from: https://redux.js.org

[57] Axios documentation. Available from: https://github.com/axios/axios

[58] Google Maps Platform Documentation. Available from: https://developers.google.com/maps/documentation/

Appendix A

Acronyms

API Application programming interface

CB Content-based Filtering

CF Collaborative Filtering

CSRF Cross-site Request Forgery

DCG Discounted Cumulative Gain

FPR False positive rate

JSON JavaScript Object Notation

LBS Location-Based Service

LBSN Location-Based Social Network

MAE Mean Absolute Error

NDCG Normalized Discounted Cumulative Gain

NMAE Normalized Mean Absolute Error

POI Point Of Interest

REST REpresentational State Transfer

RMSE Root Mean Squared Error

ROC Receiver Operating Characteristic

RS Recommender system

RWS RESTful Web services

SPA Single-page application

SQL Structured Query Language

SVD Singular-value decomposition

TPR True positive rate

Appendix B

Testing survey

B.1 Tester 1

Maksym, 24 years old, student.

B.1.1 Pre-test questionnaire

1. As usual I use travel guides, local guides, youtube travel series.
2. Yes, I use navigation apps, like google maps that allows to search places that are located nearly.
3. Yes.

B.1.2 Post-test questionnaire

1. Application looks nice.
2. Yes, it was clear.
3. I like the map on the search result page, where you can find places near you favourite location. Filtering by attributes looks very useful, especially if you in a situation when you haven't any cash, you can find place where you can pay by card, or order a delivery.
4. I don't like that all results are shown on the one page, especially on the map. Also I don't like that after changing user picture, updated information doesn't show even the successful notification message is shown. It confuses the user.
5. I suggest to add successful notification messages after creating a profile, add pagination for search result and reviews and add photo gallery for places cover image.

B.2 Tester 2

Kostantyn, 21 years old, student

B.2.1 Pre-test questionnaire

1. I use advises from friends, foursquare or advertising on bus stops.

2. Yes, as I have said, I use foursquare and sometimes web sites that aggregate all events in my city.

3. No.

B.2.2 Post-test questionnaire

1. Application looks fine, but it has small functionality and sometimes it doesn't work as I was expected.

2. Yes.

3. I like that the interface was very intuitive.

4. I don't like that categories are not mixed together, so you can choose only one, also I don't like that selected category aren't highlighted.

5. I think that good improvement will be allowing user to choose more categories in one time, also I can suggest to add some arrows for switching in the search fields, because if user fill wrong field he can easily switch, it is no need to retype his searching parameters, and also I advise to fix some weird application behavior especially with registration form.

B.3 Tester 3

Sergey, 20 years old, student

B.3.1 Pre-test questionnaire

1. Mainly I use the youtube.

2. Sometimes I use foursquare.

3. No.

B.3.2 Post-test questionnaire

1. Application is good and easy to use.
2. Clear.
3. I like that application is user-friendly and intuitive.
4. I don't like that it no allowed to search only by city name.
5. I advise to add searching only by city name, maybe to do it by the most popular category.

B.4 Tester 4

Ludmila, 21 years old, student

B.4.1 Pre-test questionnaire

1. I just google everything what I need.
2. No, I don't use any of them.
3. No.

B.4.2 Post-test questionnaire

1. It is pretty easy to use.
2. Yes.
3. I like that application is simple and as I have said it easy to use.
4. I don't like the submenu under the user icon, that you need to click to the icon and than choose Profile option, I was expected that I'll be navigated to profile page after clicking the user icon in the navigation panel.
5. I don't know, maybe fix that thing with user profile navigation.

B.5 Tester 5

Vlad, 22 years old, student

B.5.0.1 Pre-test questionnaire

1. I listen my friends and their advises.
2. No.
3. Yes.

B.5.1 Post-test questionnaire

1. Regular application, I found all that I was need.
2. Yes, it was ok.
3. I like that I can find all that I need.
4. I don't like that searching fields don't have dropdown options, and you need to start typing.
5. I advise to change name of registration button to something more understandable, that will not confuse the user.

Appendix C

Installation guide

The following steps need to be performed for the application installation.

Before the application installation it might be installed node.js, python, PostgreSQL.

For the backend part installation:

1. Go to the directory `application/backend`
2. Create virtual environment by the command from the terminal:

   ```
   virtualenv poienv
   ```

3. Activate created virtual environment:

   ```
   source poienv/bin/activate
   ```

4. From the terminal run the command:

   ```
   pip install -r requirements.txt
   ```

5. In the file `application/backend/poi/poi/settings.py` find the `DATABASES` section and set it up according to your database.
6. Go to the directory `application/backend/poi`
7. Run the command:

   ```
   python manage.py makemigrations
   ```

8. Run the command:

   ```
   python manage.py migrate
   ```

9. Load data to the database:

   ```
   psql -h hostname -d databasename -U username -f application/poi.sql
   ```

For the frontend part installation:

1. Go to the directory `application/frontend/gui`
2. From the terminal run the command:

   ```
   npm install
   ```

To run the application:

1. Go to the directory `application/backend/poi`
2. Run the command:

   ```
   python manage.py runserver
   ```

3. Go to the directory `application/frontend/gui`
4. Run the command:

   ```
   npm start
   ```

The application is running on `http://localhost:8080`. The administration page is available on `http://localhost:8000/admin`. Administrator login data is the following:
username: admin
password: admin
To create new administrator user:

1. Go to the directory `application/backend/poi`
2. Run the command:

   ```
   python manage.py createsuperuser
   ```

Appendix D

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `application` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
- `screen casts` ..... the directory of user testing screen casts