Personalized Travel Routes Generation for Mobile Application

Maksim Khlopotov, Igor Kotciuba,
Valeriia Stromtcova, Aleksandr Kudriashov, Mikhail Galperin
ITMO University
Saint Petersburg, Russia
{115801, 148954, 224733,225238,197317}@niuitmo.ru

***Abstract*—This paper presents the personalized routes recommendation system consisting of the server app that serves data for the client mobile app. Both the development process and the theoretical explanations of decisions made by the authors are described. The proposed solution is implemented with full cycle stages in mind including data collection, formatting, configuring data models for client apps and designing an acceptable user experience level of the mobile application.**

I. INTRODUCTION

Modern tourism is a booming trend characterized by appearance of numerous new technologies that are aimed to increase the attractiveness of places of interest and to develop a personalized approach to the creation of tourist routes. This is a result of a tourism global growth, as well as tourism economic potential and its influence on the development and improvement of public services in various regions.

The key problems that are currently being solved with e-Tourism technologies include: supporting up-to-date information updates about popular tourist destinations; providing helpful and valuable touristic guides [1]; hotels selection for long-stay city tourists [2]; an opportunity to share the stories about places visited and tourist experience in general [4].

These days, the requirements related to the tourism information technology development mostly concern searching for a methodical, algorithmic, and software tools to create diverse and realistic personalized tourist routes [3]. However, it is still a challenge for tourists to create a route that combines multiple tourist destinations into one trip [5], considering financial, temporal, and other limits simultaneously. The major difficulties also arise when choosing the route length and duration for both individual tourists and groups, especially when there is a conflict of interests between group members [3]. Moreover, the travel industry is oriented on analyzing the data to classify tourists on the base of the places of interest that they select and their travelling habits [6].

The possible solutions of the aforementioned problems include: Pareto optimization heuristics with non-dominated sorting [3]; multi-criteria optimization by popularity, cost and number of attractions [5]; route locations grouping using clusterization methods for daily subtasks search and taking users ratings and feedback into consideration [1]; genetic algorithm for variable neighborhood search and memetic search in differential evolution [3]; motivation-based routes matching with existing tourism topologies [6]; k-means clustering; trip buddy with recommendations based on user web surfing behavior [7] and other methods.

The usage of intellectual systems provides opportunities to combine user preferences with effective automated methods for complex computational tasks solving and provides better understanding of tourist behavior [8] to make decisions with higher quality and speed [1]. Nevertheless, the search of new methods to solve a problem of personalized tourist routes creation has no complete solution, and research in this area remains relevant.

The development of a recommendation system that provides users with unique personalized walking routes on their mobile devices has the following goals:

- cover the majority of potential interests,
- represent cultural and historical information about Saint Petersburg in conformity with user preferences,
- achieve the advantage of being able to generate an endless number of routes containing all possible combinations of existing locations which have been picked and combined using the recommendation algorithm.

II. DATA MINING & PREPARATION

A. Data preparation task

Every travel route generation system [9] requires a huge amount of specific format data that comes from a trustworthy origin. Currently, existing relevant data oftentimes dissatisfies the condition of clear standardization and full value of the information needed. At the same time, there are sources of data which were not used to parse the information from. With an example case of collecting list of persons who have a relation to Saint Petersburg, the entire multilevel process of collecting, parsing and formatting data is analyzed.

The proposed solution includes a set of software instruments to work with Wikipedia that provides such utilities as data retrieval, data filtering, and data transformation to the format which is specified by the subject area. An introduction of scoring system is suggested to rank and rate the relevance

of all the information located in Wikipedia categories. It is implied that each Wikipedia category (e.g. Poets, Musicians) is tagged by an expert group. For each tag the scoring systems calculates an integer value that is later used to determine the overall score of each data entity. An expert group also sets all minimal acceptable values for each tag in set which allows filtering low relevant or low contentful entities from the subject area perspective.

Steps of raw data processing:

1) Narrow thematic categories exclusion.
2) Scoring (calculating integer rating for each person) based on 7 criteria (size of Wikipedia page, languages count that this page is translated to, hyperlinks amount, amount of external Wikipedia services URLs, count of the word “Saint Petersburg” and its forms mentions in the text, amount of categories Wikipedia page belongs to, amount of notes - additional notes, links and footnotes).
3) Filtering (removal) of 7-10% (on average) of persons from the list tail (frequently these were "empty" pages with a few sentences in its description, empty main body of the page and all 7 criteria had the lowest values).
4) Collecting all the information from the persons left, which includes each person credentials, birth and death dates if applicable, short description.
5) Normalization (conversion) of categories titles to the names of professions that these persons were engaged in when possible (an example of non-convertible category is "graduates of some university").

Then a term of non-convertible category is introduced separately - this is the category with the title which cannot be converted into a profession name. This category related work is necessary since it is important for the person data model to have the information about their profession as it can be used as a base filter when user requests are being processed. User may wish to walk around the places that are related to famous people of a specific profession or even to an exact person.

B. Results visualization

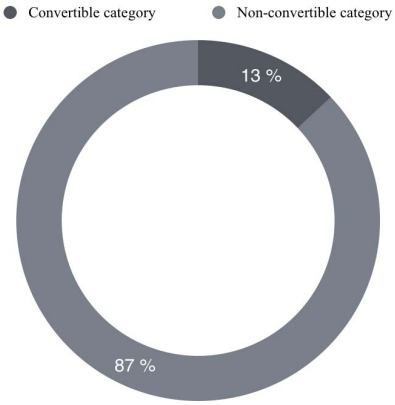

Fig. 1. Convertible to non-convertible categories ratio

An average number of persons in non-convertible categories as it is shown in Fig. 1 exceed an average number of persons in convertible categories by 6.5 times. It is explained by the fact that non-convertible categories set contains such titles as “died in Saint Petersburg”, “awarded a Leningrad defence medal”, “born in the modern Saint Petersburg area”, “Leningrad blockade related people”, “teacher of Saint Petersburg universities”. The category ‘globalization’ is noticeable to relate frequently to geographical position, life cycle step (birth or date) or relation to a certain event.

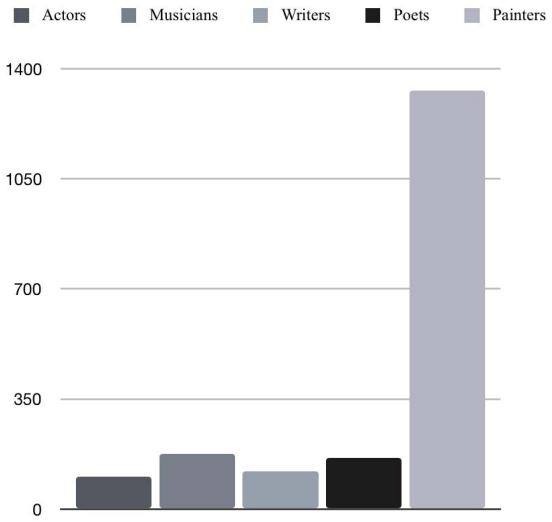

Fig. 2. Distribution of creative professions representatives

Painters take the major part (Fig. 2) because the students of Saint Petersburg Repin Institute and Art-Industrial Academy are highly recognizable. Another possible explanation of the fact that painters take the major part of the persons fetched is an availability to be visually engaged with the results of their work placed in various museums located in Saint Petersburg in comparison to musicians and poets whose works require a bit more conditional context to be consumed.

Since every data category on Wikipedia can have an endless number of subcategories, it becomes valuable to know which of them has the biggest subcategories set.

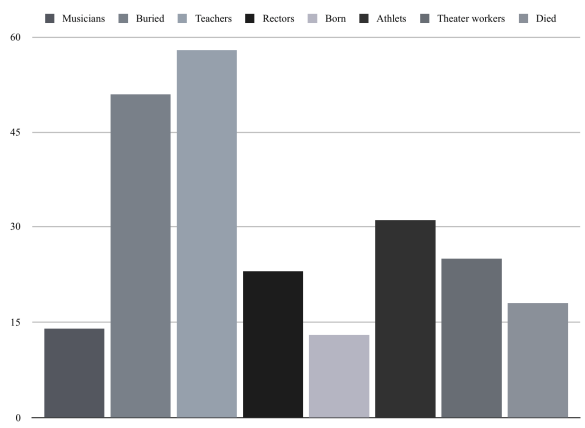

Fig. 3. Distribution of persons’ creative professions

The highest values are kept by categories “buried in Saint Petersburg” (nested level differentiation by the places of burial) and “teacher of Saint Petersburg universities” (Fig. 3).

All the rest slice of data stays inadvisable to split to more subcategories in the opposite to unite them in one list.

While solving the task of persons’ data preparation, 84k entities have been collected.

After passing through the aforementioned scoring system, only 81k entities have been left. This persons’ set was ready for being used in the recommendation system. The same data collection approach was used for “Saint Petersburg attractions” category data retrieval as well. Later the persons’ set content was used as tags for the related locations.

A combination of attractions and persons related to Saint Petersburg datasets is powerful in terms of providing user with the relevant content to their requests. To let them type persons’ names and expect routes including locations related to these persons to appear in these routes, a work of extracting relations between persons and locations has been performed. Since each Wikipedia page has a set of categories it belongs to, it became possible to organize a cloud of tags for each entity collected by parsing these categories, formatting them and attaching to the database object.

Categories names contain words of any part of speech and only nouns were left as tag content to better match the typical user query like “Saint Petersburg of Pushkin”. An array of categories names was split into separate words, and then short ones were removed (like particles and articles). After that a dictionary of each word frequency was calculated, one time appeared words were excluded, and the rest of future tags were passed through a pymorphy2 library to filter nouns. If possible, word forms were also converted to neutral representations to avoid the maternal or numeric declension.

Location to person relation was extracted from the location description text with the help of Named Entity Recognition and Natasha frameworks. Rule-based named entity recognition returned either full extracted persons’ credentials or only initials with last names which was also enough to check if this person is presented in an existing persons’ data set or is needed to be additionally fetched from Wikipedia as well. This extraction could also be contextual in case it provides the type of relationship. As a result, this led to the appearance of verifications concerning relationship extraction correction by comparing the context of the location and person by their categories. Even if categories comparison was impossible due to complete difference between compared contexts, it was still presumable that the person-location relationship was extracted correctly. To solve this kind of cases, an alternative way of checking relationship context has been made. There was a tool developed for that purpose and it had an interactive mode of verifying location context, person context and their relationship format. Thus, a human in a role of the operator could act as a verifier using that tool. This was a one-time need to finalize all works related to database content preparation.

The only data entities extracted from Wikipedia were Person and Attraction (place of interest). The designed usage of the system is intended to let the end user search both with the person they are interested in or specifying the type of the place to visit they wish to know more about. That is why collecting and storing locations to persons (and backwards) relations became such an important subtask of the data collection step.

C. Tools used

All technical tools were developed using Python 3 programming language during these processes and were covered with unit tests that allow to be sure that the system works correctly at any moment - both in general and in particular (i.e. each submodule). That also lets be adaptive to any changes on the Wikipedia side for the future similar data collection operations. Such libraries as *requests* and *re* were helpful as well since regular expressions are oftentimes a good fit for solving text-related tasks and network-related functionality was required only once. The idea of downloading all the source data to the working machine and then use it as long and variably as needed was introduced, discussed and implemented. Total amount of 15 GB of data was downloaded.

III. BACKEND SERVER

A. Business requirements

Every system that serves client applications includes a backend server. It is represented by a web resource accessible via HTTP protocol that handles business logic and provides all the formatted data. The server satisfies a list of business requirements declared by the project needs and functionality set initially invented by project idea holders. A server must be able to collect, store and modify routes, locations, and user data; process data provided by user and suggest either expert or generated routes if requested.

B. Features

1) The server should:

- Store locations, routes and user data.
- Serve routes and locations data to the client applications.
- Generate on-demand personal routes based on user data.
- Modify data in database based on user input, i.e. update location value.

2) The following criteria must be complied:

- Work robustly and consistently.
- Generate optimal routes respecting relevance of each location and duration of travel time.
- Have clear and well-documented API for consumer applications.
- Provide database flexibility.

C. Data storage

Backend stores the following data needed for route generation algorithm:

1) Tags are content labels that represent relation to person or cultural category.

2) Location Instances represent points of interest, containing coordinates, descriptive information, set of tags, and value.

3) Routes are set of locations with description that stores locations consequently in the way they are offered for the user to follow.

Data model is demonstrated in Fig. 4.

D. Tools used

The Graphene-Django Framework is used for the backend core. It provides a set of utilities to quickly organize working environment and data storage, configure database and implement API endpoints that could be used by client applications.

While a database engine was being selected the key priorities were the speed of deployment, readable and easy-to-use on a mobile side data formatting, scalability and a lack of strictly designed schema. MongoDB meets all these requirements since it stores data in JSON documents and uses inner memory, which ensures that all operations will be executed fast enough. Package Django-nonrel was used to add non-relational database support, and server engine was set to django_mongodb_engine.

The key endpoint that server has is the one that takes user parameters and returns best-fit routes back. All parameters are provided to the server in URL as GET parameters. There are data type, data value and data limits checks for the input parameters since requests can be received from not only a trustworthy origin and this is mostly about secure working with any data that comes from the user.

Client applications use GraphQL language to specify queries and data fields needed to be received.

The deployment is done via Docker container. The list of dependencies used for the codebase has a specified version for each dependency with an auto update disabled in order to work with only tested and secure versions of these frameworks. Every operation performed places a set of log lines in the logging journal that can be later analyzed in case of any bugs revealing, though no personal information is logged. For the server responses we use HTTP codes that indicate appropriate state of the request but no special format of the response text. Every new version of the back end functionality is fully covered with unit tests, and release checklist about deployment, environmental and product state is respectively performed. Since code version control is used during development, no other versions caching systems are required for the opportunity to roll any functionality back if needed.

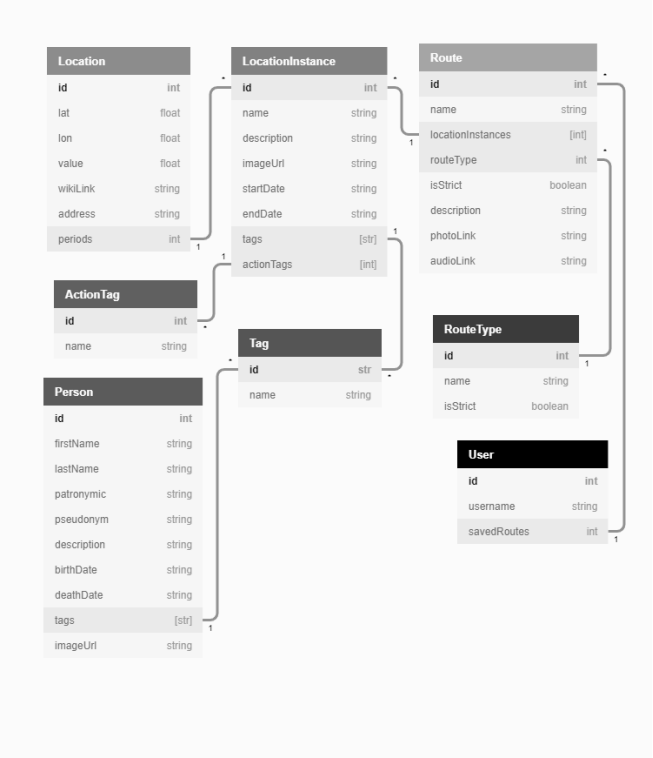

Fig. 4. Data structure used by route generation algorithm

The codebase is manually typed whenever it is written with Python 3 which supports dynamic types and strictly linted before the code is pushed to the production version of the server. It allows several developers to write a readable and understandable code for everyone.

IV. Routes Recommendation Algorithm

A routes recommendation algorithm is the core of modern travel-related applications. However, creation of such system is a challenging problem [10]. Many of the existing systems are oriented on self-drive tourism [12], while the proximity of locations in Saint Petersburg is better suited for walking tours. To address this issue, mining of user social media data [14], historical mobility records [15] and person- location relations is performed. Then resulting data is transformed into a set of tags for route personalization.

The proposed recommendation Algorithm 1 uses previously collected data from the database and user request as an input. Request contains the user coordinates, tags and duration in minutes. Combined with the information from the database (such as relations between locations and historical figures, list of possible activities in location etc.), it is possible to create virtually endless number of routes.

First, algorithm collects near expert-created routes based on their center of mass and tags specified. Then it uses the same user geolocation and tags to retrieve a list of nearest locations. The list of locations is further transformed into a list of optimal possible routes between each location; optimization is done by calling Google Maps API. Each expert route is transformed into subset of routes by excluding different locations to match duration expected.

The next step is filtering both the expert and generated routes. This operation involves the following procedures:

1) Rating calculation for each route in the subsets of routes based on tags relevance and locations value.
2) Searching for the most rated route in each subset.
3) Exclusion of all routes in each subset excepting the most rated ones.

At the end of the pipeline, the algorithm returns optimal route for a given set of criteria.

Algorithm 1 suggest_route

Input: *latitude, longitude, tags, duration*, *database*
Output: *optimal route*
0: *expert_routes* = **request list of routes made by experts from** *database* **that are near to** *latitude* **and** *longitude*
1: *nearest_tagged_locations* = **request list of locations from** *database* **containing specified** *tags* **that are near to** *latitude* **and** *longitude*
2: *distance_matrix* = **call Google Maps API to get distance matrix for** *nearest_tagged_locations*
3: *generated_routes* = **use** *distance_matrix* **and** *duration* **to generate routes for** *nearest_tagged_locations*
4: *expert_routes_subsets* = **generate subsets of routes by excluding locations to match** *duration*
5: *all_routes* = *expert_routes_subsets* + *generated_routes*
6: **for each** *route* **in** *all_routes*
7: **for each** *location* **in** *route*
8: *coefficient* = **compute intersection of user** *tags* **and** *location* **tags**
9: **add** *rating* **for** *route* **based on** *location* **value and** *coefficient*
10: **endfor**
11: **endfor**
12: **sort** *all_routes* **by decrease of rating**
13: **return first element of** *all_routes*

V. MOBILE APPLICATION

A. Business requirements

A mobile application is required to provide recommendation and navigation mechanisms to the end users. Development process of the mobile application can be started once there is data collected and formatted, and a server with live endpoints that can be accessed by the application to execute the business logic.

In the app, an authorized user should have a possibility to:

1) View the list of nearby routes.
2) Get recommended and personalized routes.
3) View the route details and all the points with their descriptions.
4) View their personal location and route chosen on the map.
5) Follow the route by going to each point.
6) Finish the route and go back to the list of routes.

From the technical perspective, the app should use a single codebase for Android and iOS and be efficient enough to provide a normal user experience.

Moreover, the application should have a user friendly and intuitively understandable user interface (UI).

B. Features

The following features were extracted from the initial business requirements:

1) The app should:

- Authorize (login/register) the user.
- Display the list of nearby routes.
- Display the list of recommended routes based on a keyword and duration entered by the user.
- Display the chosen route description.
- Show the user location and connected route points on the map.
- Navigate the user on the route chosen.

2) The app should comply with the following criteria:

- Work on both iOS and Android with a single codebase.
- Have a performance close to native (under the condition that the Internet connection on a device is stable).
- Follow the standard industrial UI design guides.

C. Design

The mockups of all application screens (Fig. 5-7) were developed according to the user journey through the app and using Google guidelines to Material Design. Android users subconsciously attribute a level of trust and security to a Material Design app because they associate the app with Google. Moreover, this concept is widely used in mobile and web apps today, and its elements become habitual to the vast majority of users. Furthermore, there is a wide range of standard components such as icons, fonts, cards that can be used when developing the user interface for an app. Thus, the concept of Material Design was chosen to develop the UI side of the mobile app.

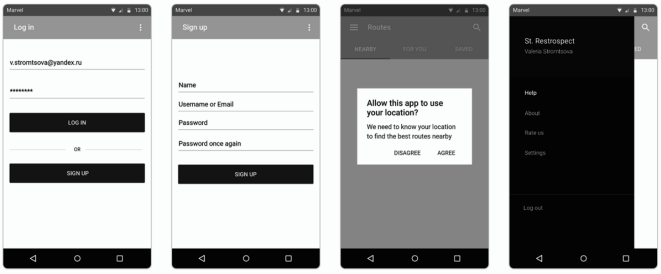

Fig. 5. General mockups

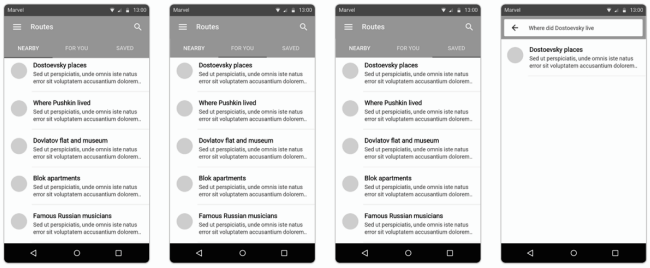

Fig. 6. Routes lists

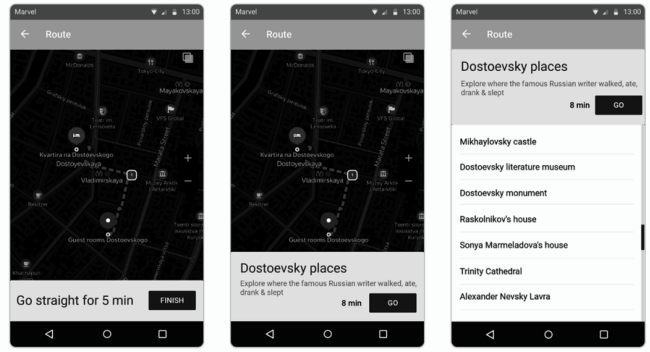

Fig. 7. Route views

The architecture of the mobile app complies with the component approach, used as a standard in React Native applications. In addition, three principles of Redux are used to build the application.

On a high level of abstraction, the application code is split into several parts:

1) App, a single code base for both iOS and Android including:

- Components, UI elements that are used to display the information to the user and react to the user actions.
- Locales which provide localization for the whole application.
- Redux (with subcomponents *actions*, *reducers*, *store*) which is used to handle the business logic.
- Routing which provides the navigation between screens.
- Screens, that are meta-components combining different components to display on a single screen.
- Services to interact with the backend via API.
- Theme provides consistent theming (colors, fonts, paddings, etc.) for the whole app.

2) Android, Android-specific settings.

3) iOS, iOS-specific settings.

4) Tests covering the App code.

The component diagram for the mobile application can be seen in Fig. 8. Mobile application consists of a set of screens user interacts with, architecture modules like redux and routings that are suitable for navigation, theme module to support easy customization of core colors of the app user interface, a bunch of services which are split into core services (database or networking related) and upper layer services (actions like adding route to favorites or leaving a review), and locales module that lets the app interface to be translated into several languages.

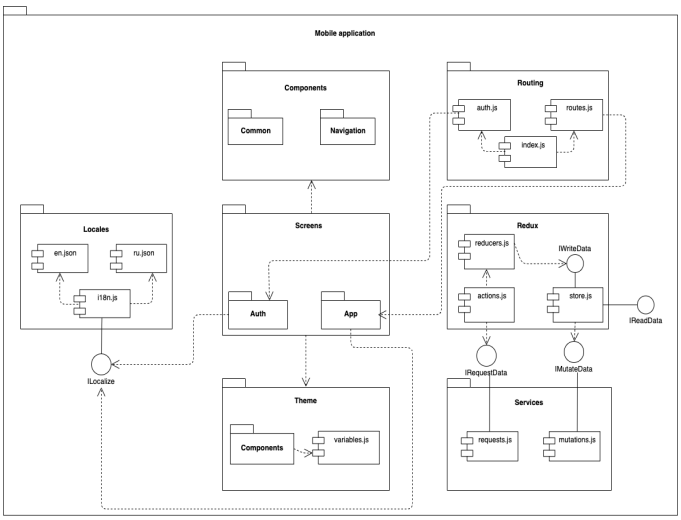

Fig. 8. Mobile application component diagram

The activity diagram for the mobile application is demonstrated in Fig. 9. User is prompted to share their timing limitations on how long the route should last. The cloud of their interest is also specified by a set of tags that describe routes, locations and persons they are related to. After that, a route is generated on the server and returned back to the application. User is now able to start their walk through this route; their location will be tracked via smartphone GPS module, and information about locations included in the route will be changed on the screen accordingly.

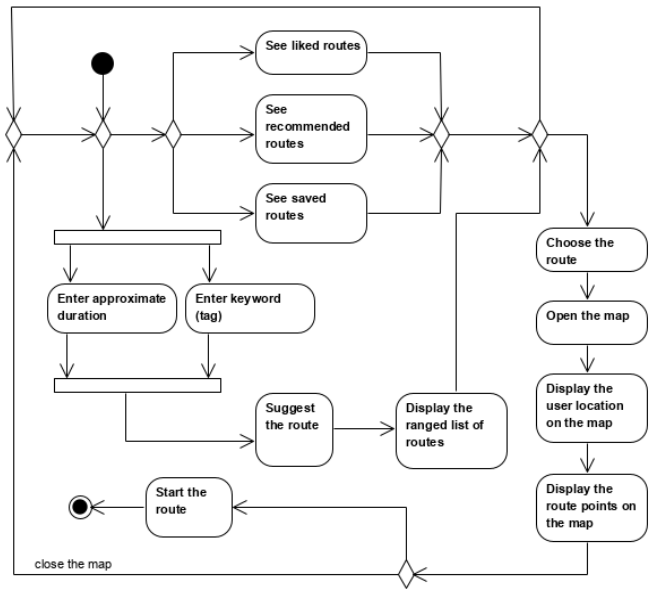

Fig. 9. Mobile application activity diagram

There are also features like adding route to the list of favorites by liking them, checking out routes recommended

and constructed by the group of experts and saving the route locally for the later use (walking them through) at any time in future.

The classes diagram for the mobile application is shown in Fig. 10.

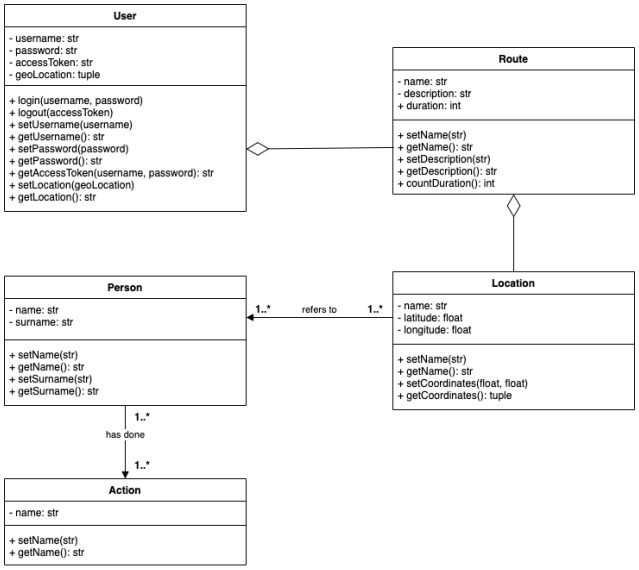

Fig. 10. Mobile application classes diagram

When designing system models, their properties and methods, it is a huge must to keep everything compliant with software engineering principles like single responsibility (which defines that every class must solve only one business logic problem), open-closed (forbidding classes to be modified but allowing them to be extended in its functionality), Liskov substitution (working with classes that inherit from other classes the same way as if they were their root classes), interface segregation (visual separation of features accessible to user by their business needs) and dependency inversion (designing everything in the way classes depend on easily replaceable abstractions rather than concrete implementations) principles.

The data flow diagram for the mobile application is demonstrated in Fig. 11.

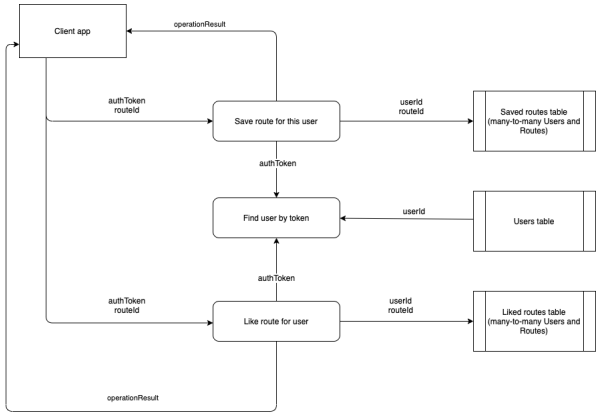

Fig. 11. Mobile application data flow diagram

Mobile application uses token-based authentication system. Once user has been authenticated, a token string is generated for their session. Next, the user passes this token in the Authorization HTTP header for each request they make. Since every business logic method on server requires authentication, the first action the server does when receiving a request with token included is finding user by this token in the database and then proceeding to the actual execution of the request.

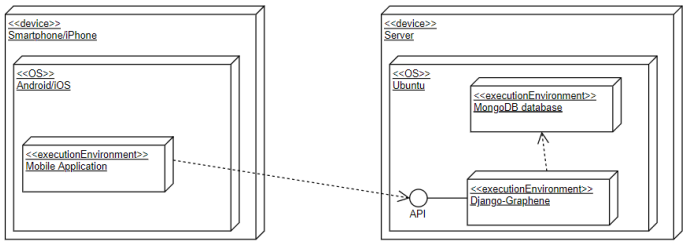

Fig. 12. Mobile application deployment diagram

The application is deployed to smartphones running Android or iOS operation systems. Each of them provides its own limits and rules on running the same tasks related even to the user location-tracking task.

D. Technology stack

While choosing the technology stack, the following considerations were taken into account:

1) A single codebase should be used for both iOS and Android operating systems (in other words, a cross-platform app should be developed).
2) Efficiency, speed and reliability are crucial when it comes to navigating the user through the city following a specific route.
3) The technologies should be relevant and supported by a large community. This is crucial for future maintenance and scalability of the app.

A comparison of frameworks that could be used for a mobile application development is shown in Table I.

TABLE I. FRAMEWORK COMPARISON

Framework / Criterion	Apache Cordova	Xamarin	React Native
Usage of platform-native programming languages	-	C, C++	Objective-C, Java
Supported platforms	Android, iOS, Windows, Blackberry, web	Android, iOS, Windows, Blackberry	Android, iOS
User interface development technologies	HTML5, CSS3, JavaScript, AngularJS, Ionic	HTML5, CSS3, JavaScript	HTML5, CSS3, JavaScript, React Native Components
Similarity to platform-native apps	Medium	High – compiled to native app	High – built as native app
Data model	WebView	WebView	Own data model

React Native was chosen as the tool to build the app. It allows using its own data model to avoid WebView [17] problems with performance and resource usage. React Native possesses a large community of developers and maintainers, a detailed documentation, a great number of tutorials, and good performance, which makes it an optimal tool to create the app.

Moreover, to implement fully the functionality of the app, the following libraries were used:

1) Apollo GraphQL provides integration with backend API using the GraphQL query and mutation language.
2) Redux is a widely used tool to manage an app global state with such concepts as *actions*, *reducers* and *store*.
3) React-native-localize is used to translate the text on user interfaces depending on the locale that is set on the operating system level.
4) Native base provides standard UI components and allows changing easily the application theme colors.
5) React-navigation is used to provide the navigation between the application screens (views).

React-native-maps, react-native-geolocation-service, react-native-maps-directions provide an API to display the map on screen, set points (locations) on the map, display the current device location and calculate and display routes between given points.

Consequently, the choice of technology stack was made based on the various criteria, such as popularity and documentation, community size and support, and technical requirements (e.g. the application should be cross-platform).

E. Development

The application was developed according to the aforementioned business requirements and features needed, as well as the architecture described above.

After installing the application, the user is prompted to sign up or log in the app by either creating a username and password or using the existing ones. After signing up or logging in the app, the user is redirected to the app main screen. At this point, the application finds the device location and passes it to the backend in order to receive the nearest routes. Then, the routes are displayed on the screen in a list. Having chosen a route from the list, the user is redirected to the route detail screen. Here, the route title, description and points list can be seen. Furthermore, the current location and the route as a whole are displayed on the map (Fig. 13-14).

Fig. 13. The main application screen

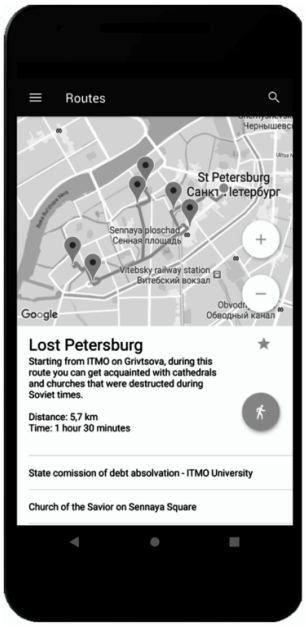

Fig. 14. Route details screen

After tapping on an action button, the user is redirected to the route navigation screen where each location details can also be viewed. The device location on the map is updated in real time, and the next location shown to the user is updated when the user walks by the locations (Fig. 15).

Fig. 15. Route navigation screen and location details screen

The user has a possibility to get a list of recommended and personalized routes by entering a text query and/or setting the desired duration of the route [11].

VI. Conclusion

The new intellectual system for walking routes generation and recommendation has been proposed and implemented. Since the development process has covered everything starting from data collection to the user experience on mobile application designing, it was manageable and continuous.

The system developed can provide an extensive set of possible routes for Saint Petersburg as it contains information about all attractions and famous places to visit and is able to generate routes based on user preferences. This system takes durational, positional and culturological aspects into consideration to offer a route to the end-user and covers the majority of sets of preferences to avoid conflicts of interests while traveling in groups.

The system aims to provide the optimal route possible by either creating completely new routes from the very beginning or modifying existing ones to fit the user requirements.

Acknowledgments

We would like to thank Wikimedia Foundation, Inc. for being a trustworthy and contentful data provider for this project purposes.

References

[1] I. Cenamor, T. de la Rosa, S. Núñez, D. Borrajo, “Planning for tourism routes using social networks”, *Expert Systems with Applications*, vol. 69, March 2017, pp. 1-9.

[2] W. Zheng, H. Ji, C. Lin, W. Wang, B. Yu, “Using a heuristic approach to design personalized urban tourism itineraries with hotel selection”, *Tourism Management*, vol. 76, Feb. 2020, pp. 103956.

[3] W. Zheng, Z. Liao, “Using a heuristic approach to design personalized tour routes for heterogeneous tourist groups”, *Tourism Management*, vol. 72, June 2019, pp. 313-325.

[4] C. Bassano, S. Barile, P. Piciocchi, J.C. Spohrer, F. Iandolo, R. Fisk, “Storytelling about places: Tourism marketing in the digital age”, *Cities*, vol. 87, Apr. 2019, pp. 10-20.

[5] M.A. Uwaisy, Z.K.A. Baizal, M.Y. Reditya, “Recommendation of Scheduling Tourism Routes using Tabu Search Method (Case Study Bandung)”, *Procedia Computer Science*, vol. 157, 2019, pp. 150-159.

[6] S. Ma, A.P. Kirilenko, S. Stepchenkova, “Special interest tourism is not so special after all: Big data evidence from the 2017 Great American Solar Eclipse”, *Tourism Management*, vol. 77, Apr. 2020, pp. 104021.

[7] M. Sumardi, R. Wongso, F.A. Luwinda, ““TripBuddy” Travel Planner with Recommendation based on User‘s Browsing Behaviour”, *Procedia Computer Science*, vol. 116, 2017, pp. 326-333.

[8] O. Boulaalam, B. Aghoutane, D. El Ouadghiri, A. Moumen, M. Malinine, “Proposal of a Big data System Based on the Recommendation and Profiling Techniques for an Intelligent Management of Moroccan Tourism”, *Procedia Computer Science*, vol. 134, 2018, pp. 346-351.

[9] C. Bin, T. Gu, Y. Sun, L. Chang, L. Sun, A Travel Route Recommendation System Based on Smart Phones and IoT Environment, Web: https://www.hindawi.com/journals/wcmc/2019/7038259/.

[10] Y. Xu, T. Hu, Y. Li, A travel route recommendation algorithm with personal preference, Web: https://ieeexplore.ieee.org/document/7603205/.

[11] G. Cui, J. Luo, X. Wang, Personalized travel route recommendation using collaborative filtering based on GPS trajectories, Web: https://www.tandfonline.com/doi/abs/10.1080/17538947.2017.1326535.

[12] L. Liu, J. Xu, S. Shaoyi Liao, H. Chen, A real-time personalized route recommendation system for self-drive tourists based on vehicle to vehicle communication, Web: https://www.researchgate.net/publication/259519060_A_real-time_personalized_route_recommendation_system_for_self-drive_tourists_based_on_vehicle_to_vehicle_communication/.

[13] P. Ashokkumar, N. Arunkumar, S. Don, Intelligent optimal route recommendation among heterogeneous objects with keywords, Web: https://www.sciencedirect.com/science/article/pii/S004579061830404X.

[14] R. B. Chandanshiv, A. Nawathe, Travel Route Recommendation by Mining Travelogues and Community Contributed Photos using Cosine Similarity, Web: http://ijsrd.com/Article.php?manuscript=IJSRDV5I41383.

[15] Y. Ting Wen, J. Yeo, W. Chih Peng, S. Hwang, Efficient Keyword-Aware Representative Travel Route Recommendation, Web: https://yonsei.pure.elsevier.com/en/publications/efficient-keyword-aware-representative-travel-route-recommendatio

[16] D. Gavalas, M. Kenteris, C. Konstantopoulos, G. Pantziou, "Web application for recommending personalised mobile tourist routes", IET Software, vol. 6, no. 4, pp. 313-322, 2012.

[17] Hu Junguo, Qi Hengnian, Dong Feng, "An improved ant colony algorithm and the path planning of tourist attractions[J]", Application Research of Computers, no. 05, pp. 1467-1420, 2012.

[18] J. Wang, N. Wu, W. Xin Zhao, F. Peng, X. Lin, Empowering A* Search Algorithms with Neural Networks for Personalized Route