"Mally" Mega Silk Way Shopping Mall Mobile Application

Dias Rysbekov[1*], Alisher Omarbekov[1*], Dias Daurenuly[1*], Anuar Kengesbek[1*] and Dimitrios Zorbas[1†]

[1*]Computer Science, School of Engineering and Digital Sciences, Nazarbayev University, Kabanbay Batyr, 53, Astana, Kazakhstan.

*Corresponding author(s). E-mail(s): dias.rysbekov@nu.edu.kz; alisher.omarbekov@nu.edu.kz; dias.daurenuly@nu.edu.kz; anuar.kengesbek@nu.edu.kz;
Contributing authors: dimitrios.zorbas@nu.edu.kz;
†Project adviser

Abstract

Navigating the Mega Silk Way shopping mall is not an easy task with available applications, especially for unfamiliar visitors. Our mobile application written in Flutter and incorporating image and audio recognition machine learning techniques allows users to easily determine their location, browse through hundreds of shops within their categories, and provides the shortest possible path to their desired destination.

Keywords: Mobile, Flutter, Machine Learning, Navigation

1 Introduction

Mega Silk Way is the largest shopping mall in Astana with an area of approximately 140 thousand square meters and about 250 various shops located inside. For a first-time visitor, this poses a significant challenge in terms of navigation. Popular available map applications like "2GIS" or "Google Maps", while displaying the shops, do not offer the means to navigate inside the mall or to accurately determine the location of the user inside relative to their surroundings. Foreign visitors have even greater difficulty in navigating, as the available information about the mall in English is lackluster, and they may not be able to communicate with the local visitors and help desk personnel due to the language barrier.

Our solution for the problem was a mobile application specifically designed for Mega Silk Way. In our application, which we creatively named "Mally", users can take pictures or pick them from the gallery to find their starting location, choose the destination shops from a conveniently organized category screen, or by searching through a list of shops. Our search algorithm also incorporates an audio classification machine learning model, which was trained to identify the shop names in Mega.

This report is organized similarly to the suggested final project template, but the approach and execution parts are separated between the development of the mobile application itself, the machine learning part, navigation and pathfinding, and decision-making regarding the UX/UI design, since these parts were done somewhat separately between the group members—Dias Daurenuly worked mainly on the UX/UI and the database, Alisher Omarbekov worked on the machine learning models and their hosting, Dias Rysbekov worked on the Flutter mobile application itself and Anuar Kengesbek developed the navigation, the map and pathfinding algorithms. Nonetheless, we had to work on each others' parts as well to ensure smooth integration due to time constraints.

2 Background and Related Work

This project began as an attempt to improve on the similar project done by previous undergraduate students. Their project was based on the similar idea to allow the users to navigate through Mega Silk way by allowing them to determine the initial location by taking a picture of a nearby shop and displaying the desired path to their destination on the map. However, their shop list was already obsolete by this point, did not contain the ground floor and in the source code initially provided to us, did not have the pathfinding, the map and the shop list integrated in the mobile application. Consequently, our application was in essence done from scratch, although inheriting from the core idea of the original project.

3 Project Approach

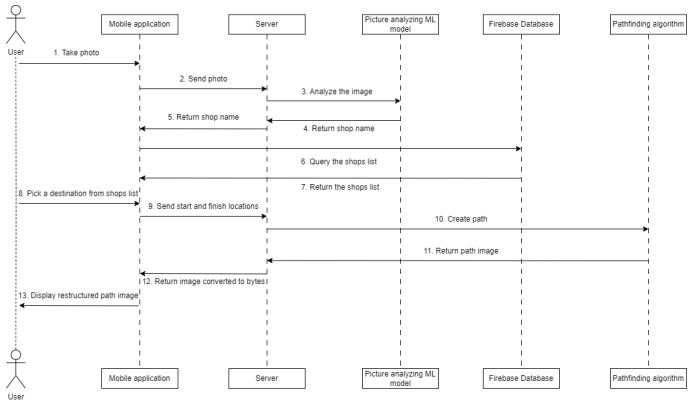

Fig. 1: Sequence diagram

As you can see in Fig. 1 our application's functionality mainly focuses on sending information and receiving results from Google App Engine and Firebase Database. When the user runs the application the Map screen is shown first with a simple map layout of the Mega Silk Way without any path on it. The user is expected to open a camera page where he can choose a picture from a gallery or take a picture using the camera of the phone. Later this picture is sent to the Google App Engine where our Picture analysing ML model is located. ML model returns the name of the shop which is shown in our application. The user then may proceed to the Categories page where he can choose the destination shop from a list of shops preloaded from the Firebase Database. On the Categories page, users also can use the search bar functionality to find the desired shop faster. Also, users can use the voice recognition ML model to enter the value in the search bar. Our model was trained specifically for Mega Silk Way shop names to eradicate the possibility of any other words appearing in the search bar if we used an already written voice search bar API. Once the shops of starting and destination locations are found, the application once again sends the shopIDs to the Google App Engine which in turn sends IDs to the Pathfinding algorithm.

The results of the algorithm are three images of the Mega Silk Way layout with the shortest path drawn on them. Later these images are converted into binary format and sent back to the application. The application restructures the binary formatted images to .png format and displays them on the Map screen of the application. Moreover, our application has profile screen functionality connected with the User Authentication system which is connected to the Firebase database, where all the related user data such as Name, Surname, email, and Profile picture is saved. Unlike the Category screen which can be displayed only after sending the picture to Google App Engine the Profile page can be accessed at any time. A profile screen is needed to save the user's personal Favourite shops and Last Accessed lists. They are also saved in Firebase DB connected to the user UID. The design layouts of our application are shown below in the Project Execution part in the UX/UI section.

Fig. 2 here shows the structures of Firebase database tables and how they are connected. Originally Firebase is a NoSQL database system and did not have a standard relational structure,

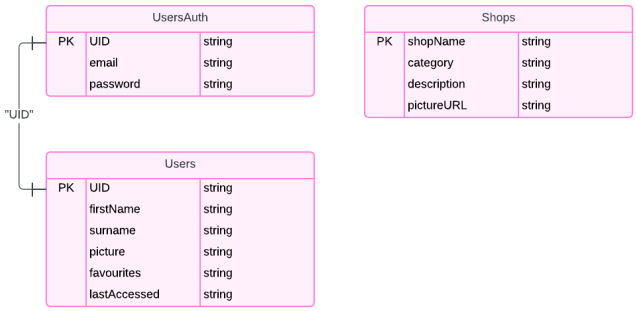

Fig. 2: Firebase Database tables

therefore there is no direct connection between tables as shown in Figure 2, this was needed to explain how the queries in the application work. The application's queries fill the three tables separately. First, the user needs to create the user account in the registration pages which automatically creates an entry in the Firebase Authentication table which is shown as the UsersAuth table in Figure 2. The User ID field is generated automatically with the Firebase services encryption system, which is later passed to the Firebase Firestore service where the Users and Shops tables are stored. The Users table is needed since Firebase Authentication tables are designed in a way that they cannot save any additional information except for email and password.

Therefore all the account-related information such as First name, surname, and profile picture are saved in a separate table connected by UID. The favorites and last accessed shop lists are stored in the Users table according to the general format of the Shops table which has 4 fields shown in Figure 2. According to our application's layout, the Favourite Shops page can be accessed from the Categories screen, while the last accessed list is displayed in the Profile screen. Both these lists can be accessed only if the user is logged in to the account, in other cases users simply will be rerouted to sign-in page of our application.

For image classification, we used an ensemble of pretrained Vision Transformer[1] (base variant) and EfficientNet-b0[2]. For audio classification, a pretrained Audio Spectrogram Transformer[3] was used. Models were hosted on Google App Engine with the help of FastAPI. Audacity was used to record the audio dataset in correct wav format.

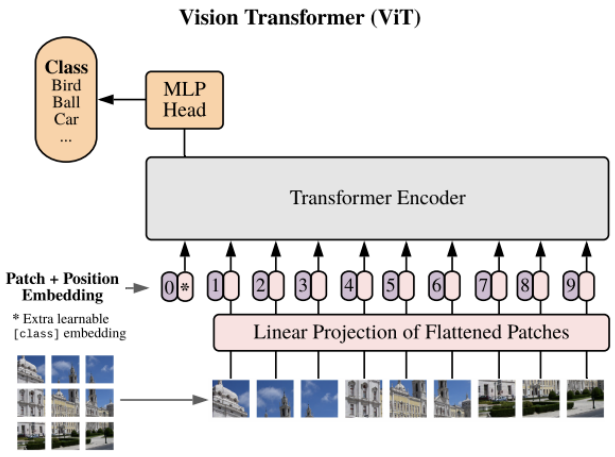

Fig. 3: Vision Transfromer architecture

The pathfinding algorithm was written in Python as it was a convenient language to use and launch on a server with the ML model. For the pathfinding algorithm, we used an undirected graph structure, as seen in Fig. 4 with no cost of movement.

Fig. 4: Graph of the mall

To build the graph we built a Windows Form application (Fig. 5) that helps us to find the coordinates and IDs of vertices that need edges. In addition, for the pathfinding algorithm, we used such libraries as OpenCV, Pillow, and Numpy. OpenCV and Pillow were used to process images and draw the path and text. Numpy was used as a way to translate the OpenCV image into Pillow and vice versa. For a pathfinding algorithm, we used the A* algorithm to find the most optimal path possible.

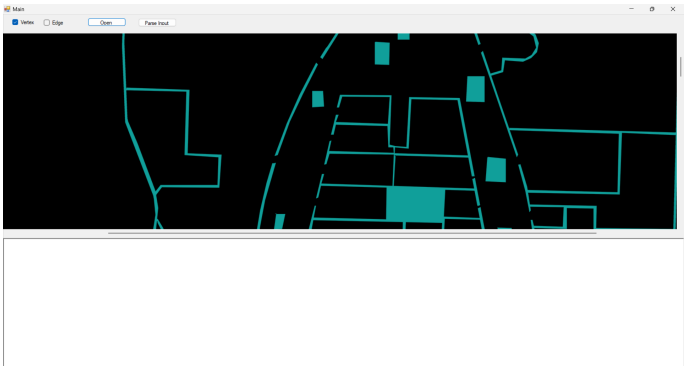

Fig. 5: Windows Application

4 Project Execution

4.1 Mobile Application

The mobile application was written in Flutter, due to the language being cross-platform, fast, and gaining popularity over the last years. The problem was that no one in our team had prior experience with mobile development. It was quite challenging to understand the concepts and the structure of apps.

Initial trials to write the application were futile since our team tried to continue the project of the previous developers, however, their codebase appeared to be in the early stages of development and therefore mostly useless. Since we were not able to continue this way, we started the application development from scratch. To gain initial pace we used Figma-based mobile application developing services, in our case dhiwise.com, to create the code template of our future application based on our initial Figma UX/UI design. However, these services were not precise and created only the basic design layout. Even though we used the deliverables of these services as our app's base, eventually changed all the page structures, added new functionalities such as user authorization, and improved the design.

The final version of the application is totally different from the initial version. Now the app has two ML models incorporated in its structure, user account registration and connection to Firebase services. However, the application passed through user testing in Mega Silk Way in order to find weak points and errors. One such problem is that our app works only with the availability of the internet and the speed of connection is crucial to the speed of the app itself. The main points slowing our app's flow were connection and sending of user pictures to the Google Cloud server and Firebase Firestore Database queries to select the shops, with all the additional information related to the shops, like Category, Shop logo, and Shop's description. In order to solve these problems we introduced the use of threads to reach concurrency in code which manages the Firebase queries and compression of all the outgoing images that are sent to the shop identifying ML model. Consequently, the execution times of responsible functions were improved by 5 and 3 times accordingly. These improvements made the application much more suited for broader use since not every user in Mega Silk Way will have a stable connection to the internet and therefore the app itself is less demanding.

Since our application is currently designed only for Mega Silk Way we decided to make changes in user account authorization for the future scalability possibilities. Now the application does not save the "frequently searched" list on the phone itself but uses a new additional table saving user data such as full name, email, list of favorite shops, and list of search history. This new table is connected to the Firebase Authentication services which are specifically designed to manage and secure user data, such as email and password. In order to save their data we connected two tables via their automatically generated userID. These changes ensure that users can use our application and no information is connected to the hardware itself, only to the account. This will be handy in future improvements if we decide to add new shopping malls.

Another problem we encountered was with the appearance of the map itself, the main focus of our application. The Pathfinder algorithm was loaded up to the Google Cloud Server similar to ML models and we needed to decide on what to return to the application once the path was created. The initial idea was that the application receives the image of the path itself which was created by the Pathfinder algorithm and overlaps it with the empty map image we used in our Flutter application. However, this implies that we would have to also receive the coordinates of the path start and path finish to precisely overlap the path and the map images. Consequently, this would add problems with the coding of the application, since Flutter does not have predefined possibilities to overlap images. Therefore we decided to just receive from the server a whole image of the map together with the path already present on it. Also since the Mega Silk Way mall has three floors the server should have been able to send up to three images. However, the server could not send several image files to the application. To solve this problem we introduced image conversation to bytes. In the current version when the server receives names of start and finish shops from the application it sends back the three image bytes variables, which are later restructured into images in the application itself.

4.2 Machine Learning

The implementation of machine learning into our application consisted of four stages: collecting the data for image and voice classification, training the image classification model, training the audio classification model, and deploying the models in the cloud to allow inference of data received from the mobile application. Data collection proved to be a cumbersome task. For the image dataset, a total of 2485 pictures across 242 shops in Mega Silk Way had to be taken. Since the time the original project was developed, many shops have been closed down and replaced, resulting in an inability to utilise most of the previously available dataset. Considering the desire for the user to take a picture of the desired shop from different angles and positions, each shop required several attempts at obtaining good-quality images from several locations. For the audio dataset, each team member recorded the pronunciation of shop names five times, totaling 4997 audio files and requiring several days just to obtain acceptable recordings.

Training the image classifier faced many challenges. At first, training Vision Transformer on our dataset took a very long time per epoch. Eventually, it was recognized that fetching and resizing the images in the Pytorch dataloader was the main bottleneck, as it required a lot of time to process our images that were in HD resolution, hence more time was spent on fetching the images from the dataloader than on the backpropagation itself. To circumvent this, we decided to resize

the images to the required 224px by 224px beforehand, so that the dataloader would not have to do it; as a result, the training times decreased several-fold. In addition, because our dataset was relatively small, we had to experiment with different data augmentation methods like rotation, crop, and contrast change to avoid overfitting. Since we did not want the inference to take longer than one second, we initially used a smaller Vision Transformer variant, but further tests on the Google App Engine demonstrated that we could also safely use the base model. After retraining with the larger model, the performance improved slightly.

However, after testing the model live in Mega, it struggled to classify the shops. Vision Transformer was seemingly limited by our small dataset. In many cases, the model returned very small probability values for every class, indicating the difficulty of differentiating between various shops. As such, we opted to utilise model ensembling. We trained a base (b0) EfficientNet, which had proven to be a relatively lightweight, highly performant convolutional neural network, on our dataset and ensembled it with the Vision Transformer. The probability outputs per class of each model were summed and averaged. This new ensembled model demonstrated better performance and thanks to the efficiency of the EfficientNet, this approach did not hinder inference times.

Next was the training of the audio classification model. At first, we tried to use Meta's Wav2Vec neural network that accepts raw WAV file inputs. Unfortunately, after several attempts, it was completely unsuccessful in classifying our data. Then, we decided to train an Audio Spectrogram Transformer, which used log-Mel spectrograms of the audio and trained in a similar fashion to the Vision Transformer. Since the Audio Spectrogram Transformer requirements had an older version of the 'timm' library, we had to retrain the image models accounting for that. The initial attempt at training the audio transformer failed to converge due to the mistake of setting the floating-point format to half-precision, resulting in the over- and underflow of gradients. This posed challenges, as compared to the image classification models, the audio classification models took much more time to train, especially considering that due to the limited computing power available even when using Google Colaboratory and Kaggle GPU assets, we were restricted in our ability to experiment with various training parameters and approaches After returning to the full-precision and applying data augmentation techniques such as frequency- and time-masking on the log-Mel spectrograms, the model was able to converge well, but the inference took about 8 seconds, which was unacceptable for our application. Taking into account that the average recording was about two seconds long, we cut the number of time frames in our model from 1024 to 256. Although changing the time frame dimension removed the ability to use the AudioSet pretrained variant (as the 256-dimensional pretrained model did not exist), just the ImageNet pretrained Audio Spectrogram Transformer performed in a similarly sufficient manner. In the end, it took about a second to perform inference on the input and performed relatively well.

After training the models, we needed to host them on a server. Google App Engine allowed a free trial for three months with powerful instances, which were ideal for inference. Wrapping the function calls for the models and pathfinding algorithm using FastAPI enabled smooth and fast deployment. However, figuring out how to properly format the incoming and outgoing data and JSON responses to and from the mobile application required many trials.

4.3 UX/UI

The goal was to make the user interface of the application extremely clear, easy to use and user-friendly. As the target audience of the application is any visitor of the shopping mall, the design of the application was created in a way that is easy to navigate in.

Firstly, the design was made in the software application called Figma, in a way that is intuitive and clear to understand. However, even if it had a good UX (user experience), the UI was not appealing. Here are the first drafts of the initial design:

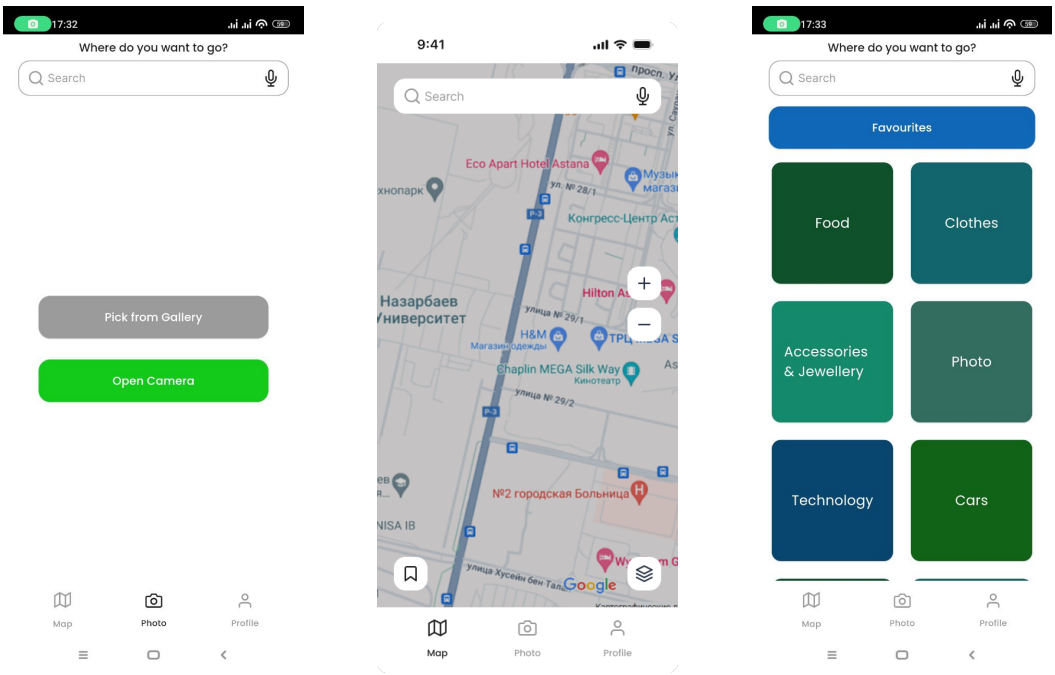

Fig. 6: The initial design of the map, photo, and categories pages.

From the illustrations above it can be clearly seen that the app lacked contrast, good colour choice and overall look. The first version of the app was tested using Guerilla testing, where we approached random people in the shopping mall to try and use the app. The reason why Guerilla testing was chosen as a replacement for Usability testing is that:

- Finding willing test participants can take time
- Arranging testing sessions in an office or usability lab is difficult to organise
- Compensating participants for their time requires a slice of the budget.
- Guerilla solves these problems, as it has these benefits:
- Directly inviting the general population to take part in usability testing
- Taking the test there and then
- Presenting a little memento to individuals as a thank you for their time

Fig. 7: Graph of the mall

From the results of this testing and reevaluating the app design by ourselves, we came to a conclusion that the app lacks visual contrast and appeal. Here is the new design that we came up with:

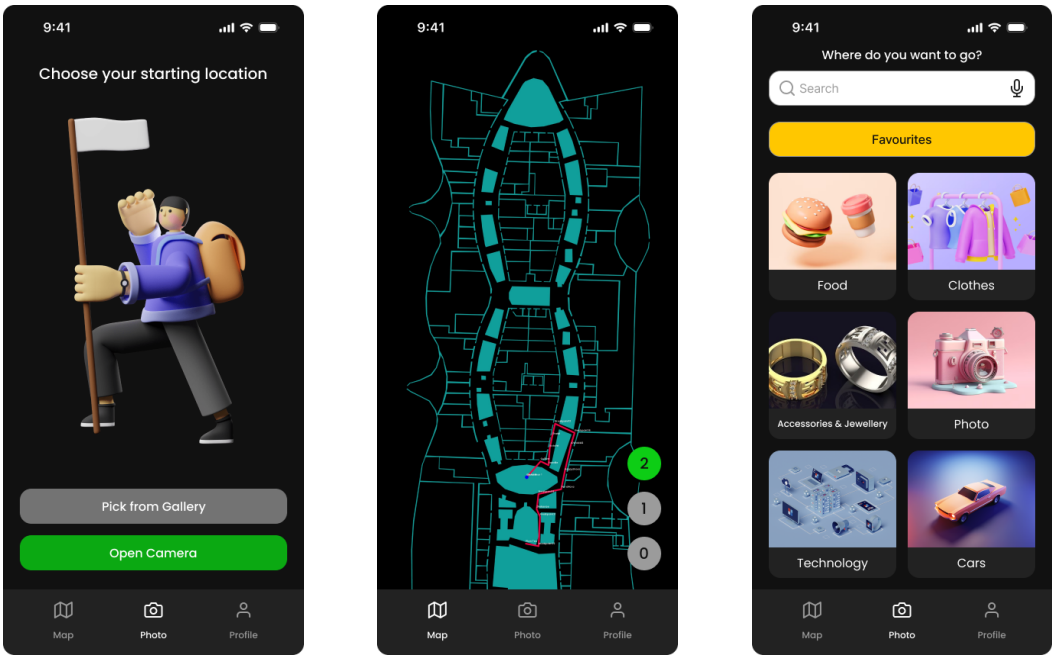

Fig. 8: The new design of the map, photo, and categories pages.

As it can be seen from the illustrations above, it is much more appealing, and there is a better colour choice and visual contrast. Let's dive deeper into the architecture of the app.

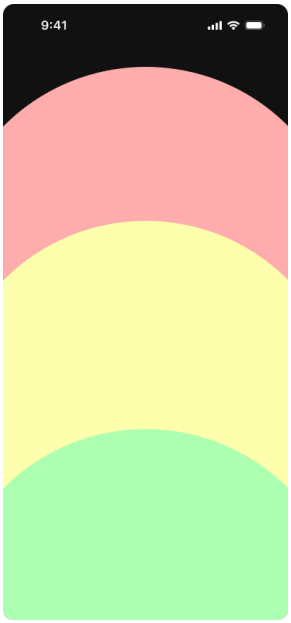

Fig. 9: Ease of access

On Fig. 9, there is an illustration of different areas indicated with different colours. The green area - is the area that is the most convenient for users to access. In this area most of the buttons and navigation should be located. That is why, here, we added a navigation bar and buttons to switch between the floors. As users' thumbs are closest to this area of the phone, the majority of main buttons are located in this area (or at least really close to it).

The second area in the illustration is the yellow area. This area is less convenient to use, but still fine. Here we added other navigation buttons that are less used.

The red area - is the most inconvenient area, so that is why we tried to decrease the number of buttons in this area to minimum.

The first thing that the user sees when opening the app is the map page (Fig. 8, first image). On this page we get the final result as the starting point, the shortest path and the destination point. We drew and painted the map by ourselves using Google Maps and 2GIS as a guide for its dimensions and size (to make it as close to reality as possible). The map was created in Adobe Photoshop CC.

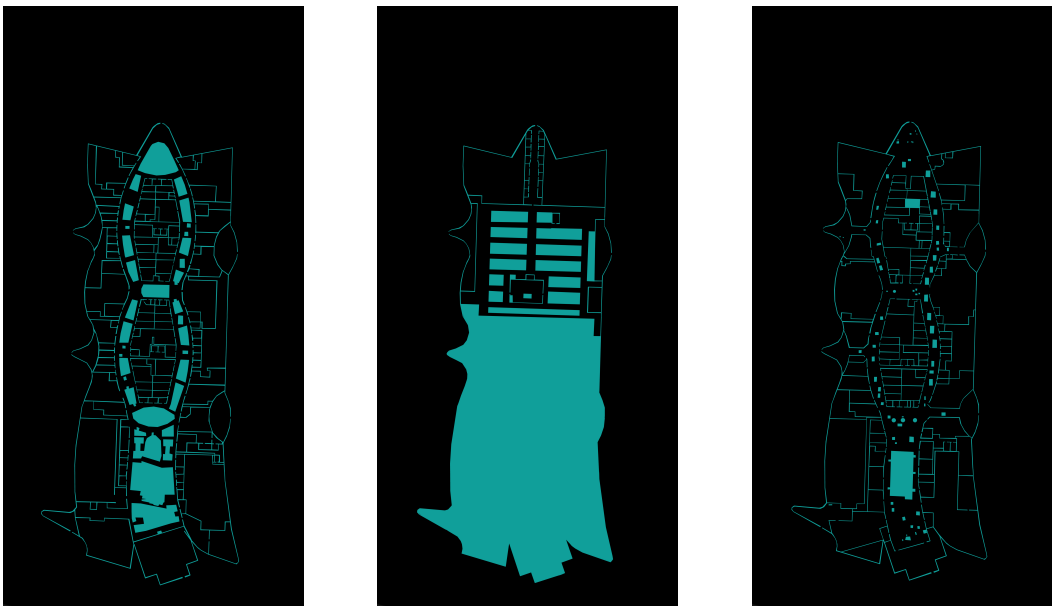

Fig. 10: Map designs of floors 0, 1, and 2

The combination of dark background (#000000) and greenish, close to cyan walls (#149F9A) made a good contrast and appealing look, so we decided to go with that. The choice for the path colour was a colour between pink and red - as this is a complementary colour for the walls, and the choice for destination colour was yellow (analogous to the path), and for the starting point it was green (analogous to the walls). This was all made to stay inside a colour scheme that works with each other.

The problem, which we encountered while creating the map and path that is understandable for users, was that we did not have access to GPS. We needed to create a solution that makes it understandable where to go from the starting point, which turns to take and in which direction to move. We decided to create checkpoints, just like in the train stations map, which shows the shops that lie on or next to the shortest path. On Figure 8, there is a visual demonstration of how we implemented this.

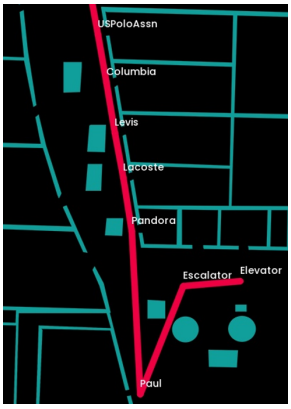

Fig. 11: Ease of access

The next page that the user proceeds to is the photo page (Fig. 8, second image). In this page the user is given a choice between choosing the photo of the shop from the gallery or taking the photo. The way buttons are arranged is in the way that the most frequently chosen button between 2 is either on the right or below(considering that people read from left to right or from up to down). This is done so that users do not need to read 3 times (e.g. 'Proceed' – 'Cancel' –'Proceed') to get to the most used button, but only 2 times (e.g. 'Cancel' –'Proceed').

After proceeding with the chosen photo, the user is navigated to the categories page(Fig. 8, third image). In the categories page, the user chooses their destination shop. Here we decided to add images on top of the names of categories in cards, so that it is easier to find the needed category. We have a search bar right below the status bar(on top of the page) which can be used to just search for the required shop destination. There is a search bar inside each of the categories too, so that people can search the shops inside categories (Fig. 12). There is a mic button (icon) on the right side of the search bar, as we implemented the ML model for speech recognition. This is added for cases where it is more convenient to pronounce the name of the shop rather than typing it. We added this to increase the accessibility of the application.

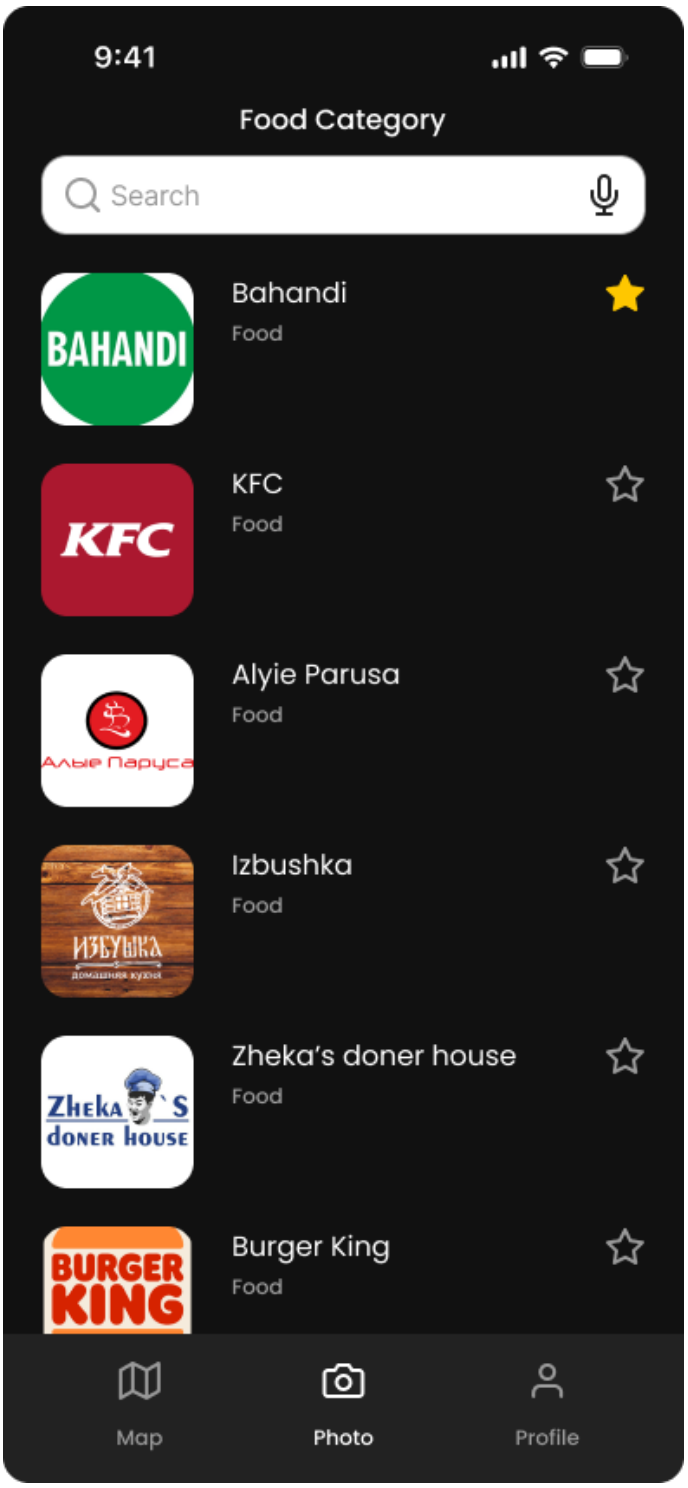

Fig. 12: Food Category page

There is also a button named 'Favourites'. However, users cannot access the content of Favourites page without logging into their accounts. Signing in and signing up can be both done in the profile page (Fig. 13).

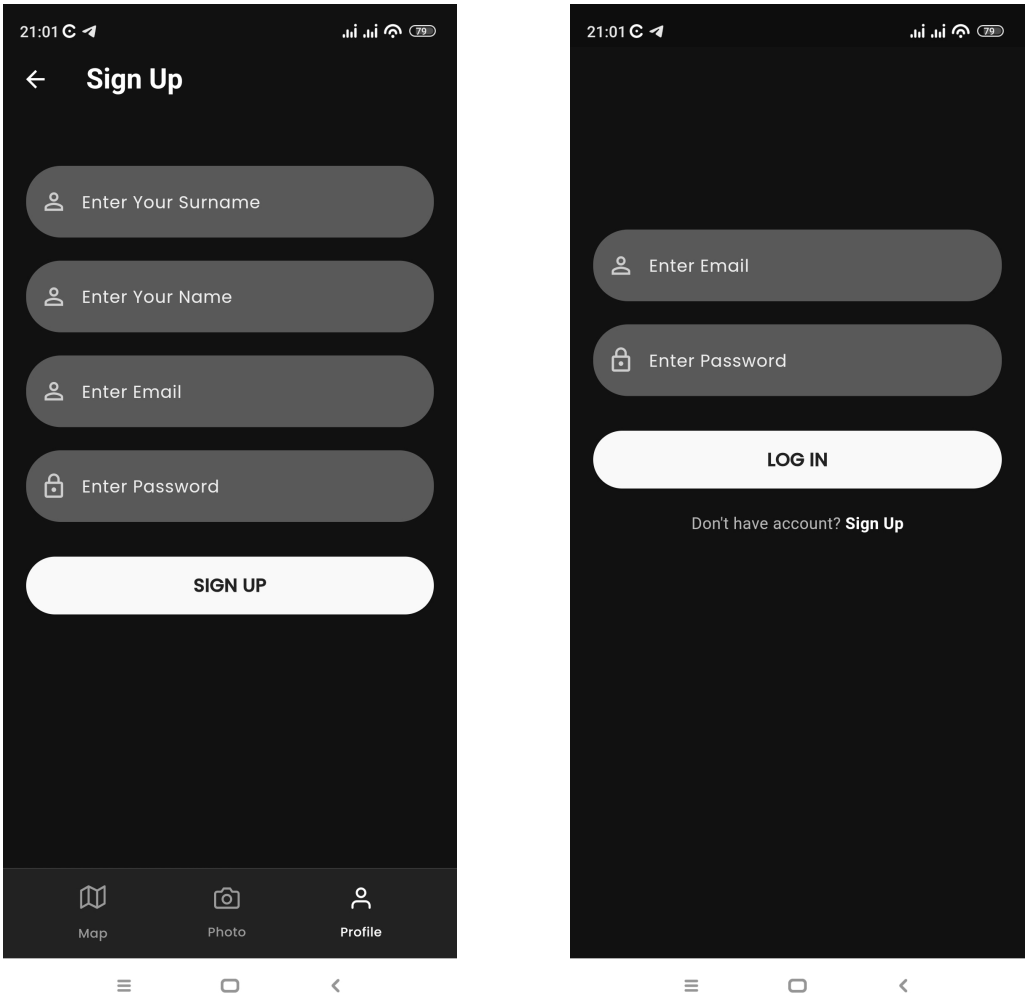

Fig. 13: Sign in and sign up pages.

Here users can log into their account and proceed to their profile page, where they are going to have access to their name and surname, option to change their profile picture. The main feature of the profile page is search history (Fig. 14). Where the last 5 shops that were chosen as the destination point are going to be shown for each user account.

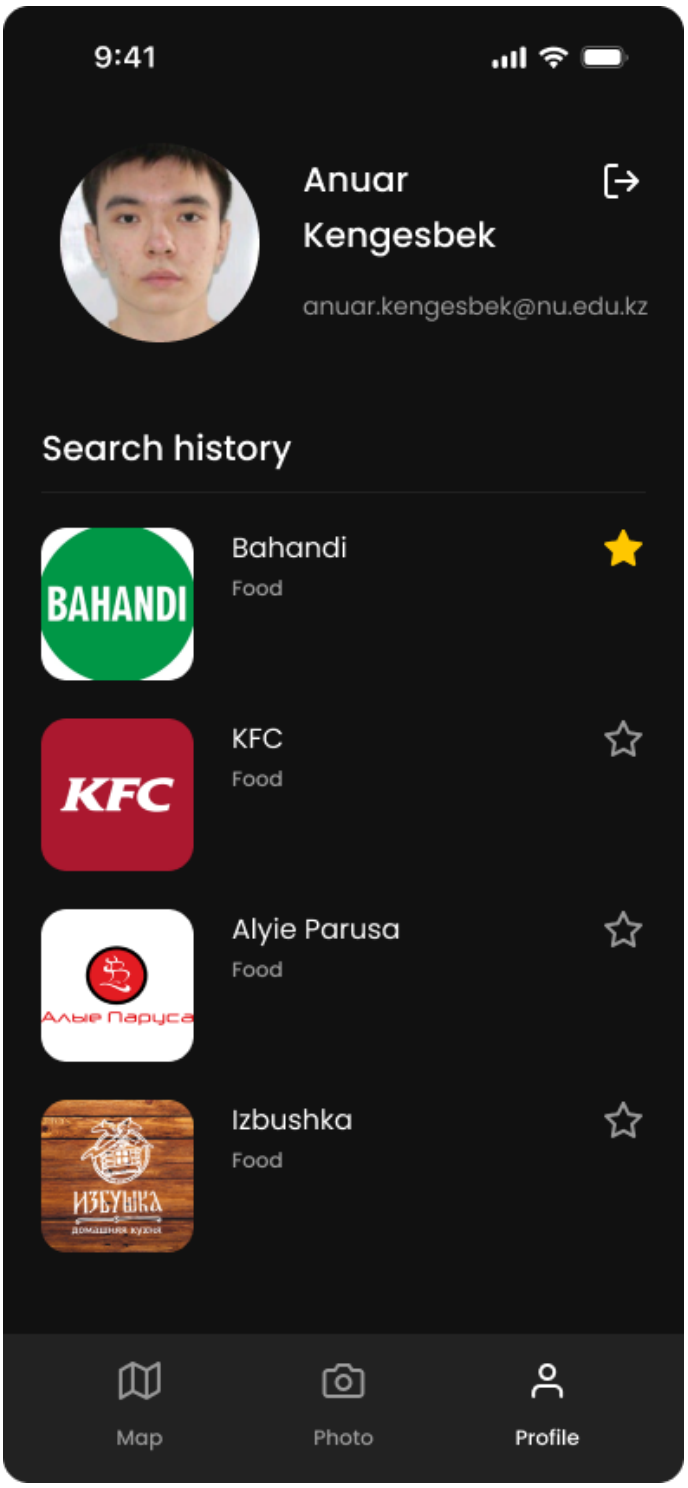

Fig. 14: Search history on the profile page.

Initially, we had an idea to put the favourites list on the profile page, with options of adding and removing favourite shops. However, we decided to implement that on the destination page, as it is the part of choosing the destination point. Instead of that, we decided to put the users' search history on the profile page, so that they have an access for the 5 last shops they ve chosen as their destination point. Here, they also have an option to add those shops to their 'Favourites' by clicking on the star icon(the same approach is on the choosing the destination page (Fig. 12).

4.4 Navigation and Pathfinding

Creating a shortest-path algorithm was a challenge in this project for several reasons. First of all, we chose Python as the language in which we will implement the algorithm. We have decided to use the A* algorithm because it is the most optimal pathfinding algorithm. Dijkstra's algorithm could be used but A* was faster and that's why we chose it.

Next up we needed the map and what to use as a map. Initially, we had several ideas including a grid, and building an undirected graph with or without cost of movement. We thought using a graph would be time-consuming as it would have over 400 vertices and even more edges. In addition, building a graph would need several checks as it could become confusing. So, we tried to implement a pathfinding algorithm on the grid that will be based on the photo of a map that our designer built. Using the OpenCV library we could transform the picture into the black and white image and from there using the same OpenCV we could build a grid that would be a Numpy array consisting of 1's and 0's. 1's would represent the walls and 0's would be a walkable area. This approach also would build a very detailed path and the shortest path possible. However, because of the size of our map algorithm needed approximately 17 seconds per path. It was disastrous and we needed to find another solution. Trying any other heuristic or reducing the size of the map did not work as other heuristics built incorrect paths and reducing the size of the map destroyed some walls and therefore some paths would go through them.

Finally, we concluded that in order for the algorithm to work we needed to build the graph. In comparison to a grid having over 6 million 'nodes' and a graph having little over 400, the algorithm would work wonderfully. Now comes the hard part of building the graph itself. We have decided that each node will have an ID and name that was in the database and a coordinate which was the pixel where that node was located. The coordinates helped us in drawing the path as we used the OpenCV library again and we just needed two coordinates to build a line between them. After building half of the first floor we spent 2-3 days properly finding the coordinates and connecting the nodes, we decided that we could build additional software that would help us to find the coordinates and connect the vertices. Some of our team members had experience in building in .NET and building Windows applications, so we created an application that would load the image and the application had two modes, vertex and edge mode. In vertex mode upon clicking on the image application prints the coordinate and in edge mode, we enter the vertice coordinates and the application builds the vertices on the uploaded image. When we click on two nodes, the application creates an edge between them and prints the IDs of the nodes that we chose. After the implementation of this application, we were able to finish the graph building rapidly.

After we finished building the graphs for different floors and connecting them via having common vertices representing elevators and escalators, we were able to properly launch the pathfinding algorithm and test it. As a result, it took around 2 seconds to find and build the path. Half of this time was taken by the graph building. We decided that this algorithm would be on a server and take inputs of start and target shops return images of maps in bytes and translate them back to images. We could build the graph on the server once and then just call the pathfinding function. Also, another reason to have the algorithm on the server was to not overload mobile phones and make all the calculations on the server. It helped us to lower the time required to build a path by half. Later, in order to make the path less confusing we decided to print out the names of the shops that lie on the path. With OpenCV we could use only default fonts so to use the desired font we added Pillow to the project and it made the map less confusing. The next problem was that the quality of the drawn path was poor and used different types of lines to make it higher in resolution; we suffered a 10-15% increase in runtime which resulted in 1.2-1.5 seconds for the response. Also, to calculate the distance that the user has to travel we calculated that one pixel of the image was approximately 0.517 meters.

As a result, we built an algorithm that worked fast and found the most optimal path. It worked properly and works on every possible and with additional software that we build we can easily rebuild the graph if there are any changes in Mega Silk Way.

5 Evaluation

Our team can agree that we have solved the project problem, and met most of our functional and non-functional requirements. Below are shown our requirements list from the updated requirements assignment written at the start of Spring semester.

- Take a photo as an input
- Recognize the store by the photo
- Calculate the shortest path to the chosen destination
- Provide with the local map of the location
- Show the calculated path on the local map
- App should have feature of voice assist
- App should track the movement of the user following the path
- App should provide feedback on successful arrival

From this list only seventh and eighth points have not been achieved due to its being difficult to achieve from the technical part. Similar paths providing applications commonly use GPS navigation systems, but this implies that we would need to use Map APIs such as Google Map API, which would be excessive for our application and would have brought additional expenses during implementation of the application. To solve this problem we had two ideas which both later discarded for having feasibility issues such as being too difficult to implement or due to the app becoming too slow to work with. The first idea was to create complementary IOT devices , which would have been placed in Mega Silk Way at certain locations, and would have sent their location information to the user phones. The application would receive the location data and on the basis of triangulation between several IOT devices would calculate the location of the user. This would put additional computational stress, brought additional expenses and made the app difficult to scale, since we would need to put these devices on every shopping Mall our app covers. Second idea was about adding GPS navigation systems without the use of Map APIs. However this is a difficult task since to achieve this we would need to change the internal structure of our Map section in our application and server data transition. Moreover, the data on how to create this functionality is hard to obtain, since this is not a common way of working with GPS.

- Response time should be around 5 seconds or less.
- UX/UI should be user-friendly and self-explanatory.
- All the data should be encrypted using SHA256 method.
- System should be compatible for any malls and buildings.

The list above is the non-functional requirements also from our assignment. We achieved all the requirements except for SHA256 encryption. The reason why it happened is that the Firebase Database systems provide their own security encryption service integrated in their structure. Firebase services encrypt data in transit using HTTPS and logically isolate customer data. In addition, several Firebase services also encrypt their data at rest: Cloud Firestore, Cloud Functions for Firebase, Cloud Storage for Firebase, Firebase Authentication (these are the ones used in our project, there are many more). Therefore addition of SHA256 would have been excessive and would have added an additional level of complexity to the application.

We used guerilla testing for both evaluation of the initial interface design(after which we changed it to the final design of the mobile application), and final evaluation, after the app was developed.

The reason why Guerilla testing was chosen as a replacement for Usability testing is that:

- Finding willing test participants can take time
- Arranging testing sessions in an office or usability lab is difficult to organise
- Compensating participants for their time requires a slice of the budget.

Guerilla solves these problems, as it has these benefits:

- Directly inviting the general population to take part in usability testing
- Taking the test there and then
- Presenting a little memento to individuals as a thank you for their time

6 Conclusion and Future Work

To conclude the work that has been done, our application allows users to navigate in the shopping mall Mega Silk Way by locating themselves via taking the photo. Image recognition model and pathfinding algorithm helps users to easily travel inside the mall. Shops are organised in categories and voice assistance makes it easier to use our application. Frequent visitors can use app more freely and have favourite shops and see last visited shops.

In the future, application can be expanded by adding more malls or buildings, just by using additional software and loading images of buildings to train the ML model. Also, the profile system can help future developers to easily control the user's data. Also, features of traceability of user movement and personalised recommendations for the shops can be added to make it more user-friendly.

References

[1] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (2021). https://doi.org/10.48550/arXiv.2010.11929

[2] Tan, M., Le, Q.V.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2020). https://doi.org/10.48550/arXiv.1905.11946

[3] Gong, Y., Chung, Y.-A., Glass, J.: AST: Audio Spectrogram Transformer (2021). https://doi.org/10.48550/arXiv.2104.01778