BEERIFY

Made by:

Császár Ákos, Informatics III

Molnár Hunor, Informatics III

Tasnádi Norbert, Informatics III

Váncza Tibor, Informatics III

Leading Teacher:

Szántó Zoltán

Table of Contents

1. Introduction

The aim of the project is to organize a database of unique beers for the client and also give possibility to the user to add a new beer to the database via an android/ iOS application. There are currently around ~800 different beer cans and these beer cans need to be organized so the client can decide in the future if a beer can is already in his inventory or not.

2. Target

We would like to develop a Flutter application for the client and also for the all the potential users. The main goal is to organize all the data about the beers; thus, the heart of the application is the database. The application would also use a Python script to determine the six most similar beers to the user's input image. These would be the main targets for our application:

- Database search (data)
- Database search (image search, using image similarity script)
- Nice looking UI
- Comfortable UX
- System flexibility (Android / iOS)

3. Requirement specification

3.1 Functional requirements

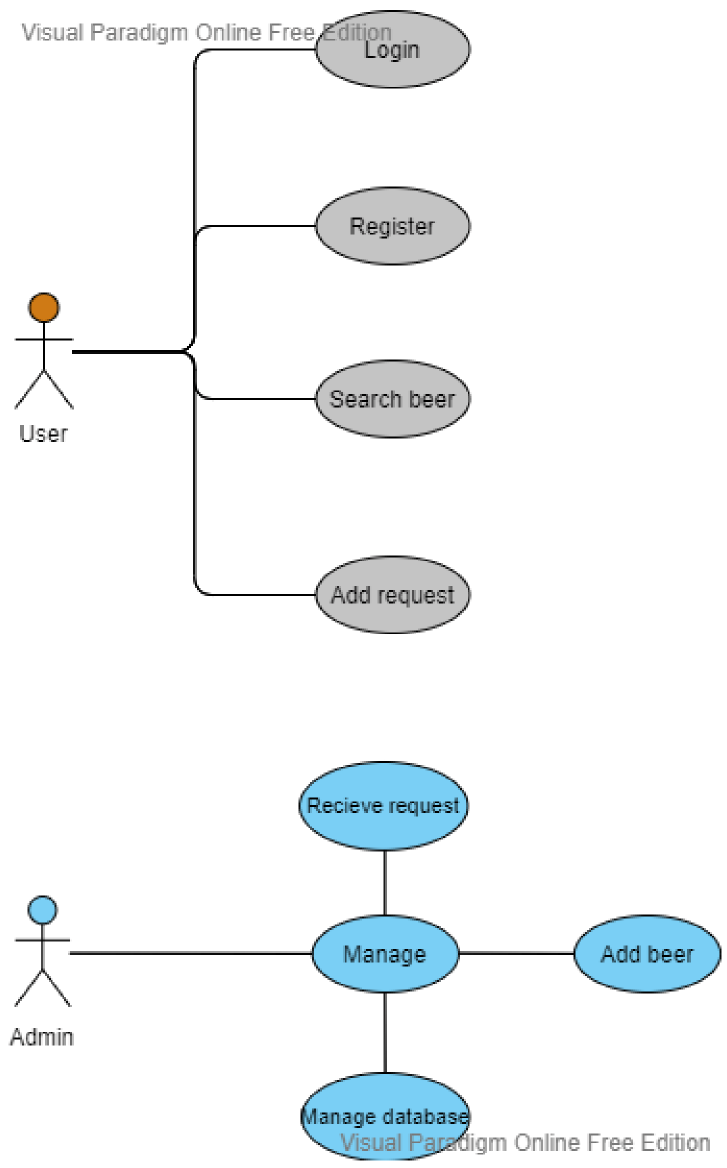


Figure 1. Functional requirements

User:

- **Register/Login:** The user must register and log in in order to use the other functions of the application
- **Search beer:** The user can search in the database using different filters. Possible filters:
    - Beer name, alcohol %, etc.
    - Image search

- **Add request:** The user can add a request for adding a beer. The user gives a few relevant data and sends a request to the admin.

Admin:

- **Receive request:** The pending requests are stored and can be evaluated later. (This option is not yet developed)
- **Manage database:** The admin manages the database by adding beers and correcting data.
- **Add beer:** If the beer is not in the database the user can send a request to the admin the admin of the application then can decide to add a beer. To add a beer the client must send the beer can to the owner and after that the admin can add the beer to the database. The beer is currently can be added by the admin only[1].

3.2 Non-functional requirements

The most important non-functional requirements for our project are the following:

- **Data integrity**: the data about the beers should stay accurate and consistent, so the user and the client can also access beer
- **Efficiency**: the search in the database in any way should be fast and accurate
- **Maintainability**: The database should be easily maintained

3.3 Software requirements

- Android OS / iOS
    - SDK env: 2.12.0 to 3.0.0
- Internet Access
- Camera Access

The project revolves around a set of beer data which means that there is a necessity to create a database for the project. The database

[1] Possible future development plan

The frontend of the project is created in Flutter, so it is ready for publishing for both Android and iOS systems. The Flutter application uses a Python image-similarity script for System Architecture

4. System Architecture

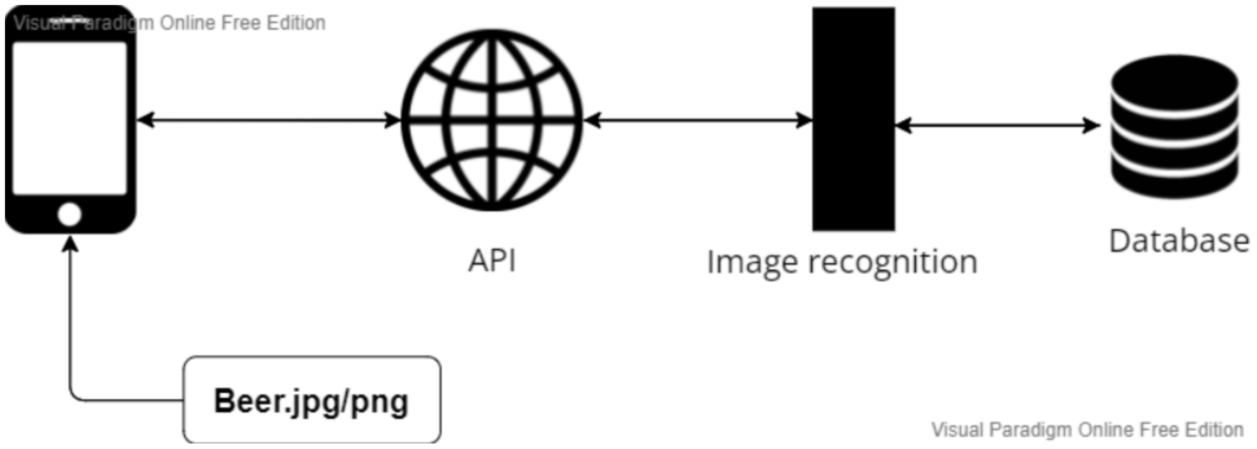

Visual Paradigm Online Free Edition
API
Image recognition
Database
Beer.jpg/png
Visual Paradigm Online Free Edition

Figure 2. System Architecture

5. Project Management

The development phases were performed and documented on *Trello* [2] and *GitHub*[3]. Trello was used for task management which helped our project greatly. On GitHub we uploaded the actual source code and used it mainly for version control. We planned the main functionalities, project structure, wireframes, UML diagrams and assigned every team member his TODO list.

6. Project Planning

The project's GitHub repository and Trello workspaces were managed by **Váncza Tibor.** During the first period of the development, we decided the roles of each people:

- **Database** – Tasnádi István-Norbert
- **Flutter** – Molnár Hunor
- **UI/UX Design** – Császár Ákos
- **Image-processing** – Váncza Tibor

[2] https://trello.com/b/Say7ocQD/szofter-tervez%C3%A9s-projekt
[3] https://github.com/tibiv111/Beerify

For version control we used GitHub. The main project was divided into the 4 main branches which later was merged as the final project. The main branches are the same as the roles we mentioned before.

During the planning phase we encountered multiple difficulties. These difficulties could be described in multiple ways we decided to use a Q/A form of storytelling:

Q: **What database should we use for our project?** We need to store images too and want it to be cheap.

A: MongoDB Atlas. Free storage until 512 MB and infinite amount of query. It is also cloud based and compatible with Flutter.

Q: **If we only have 512 MB of cloud storage, how are we going to store all the Images?**

A: The original images are compressed to a ~50kb form which means that in case of 2000 images we would only use up ~100-110MB storage space.

Q: **Is Flutter the best choice?**

A: We want to ensure that our users can use the application on both iOS and Android, so the fastest and best solution was the Flutter.

Q: **How do we solve image-search in Flutter?**

A: We decided we should use an image-similarity Python code for our project, which determines the first six best most similar beer cans that matches the input image of the user.

Q: **If we use a Python script to determine the most similar images, how does the Flutter's Dart language utilize this script?**

A: We'd send the input image via REST API to an online host *Heroku*[4] which would execute our python script with the input image and returns the similarity percent

[4] Heroku

7. Design and UI

The design for our application was envisioned in Figma, made by Ákos Császár.

7.1 Splash screen

This is the first screen that the user sees. This screen contains the application logo which is quite important for an application.

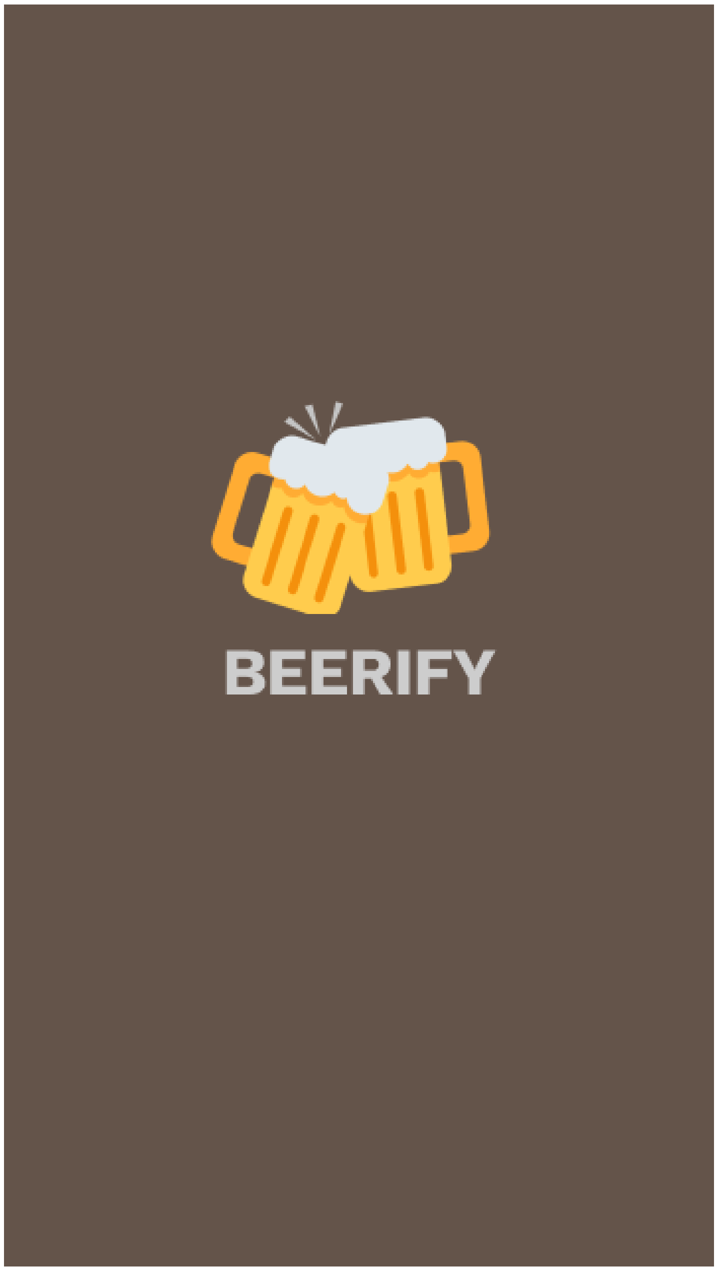


Figure 3. Splash screen

7.2 Login screen

The login screen where the user can log in to the system.

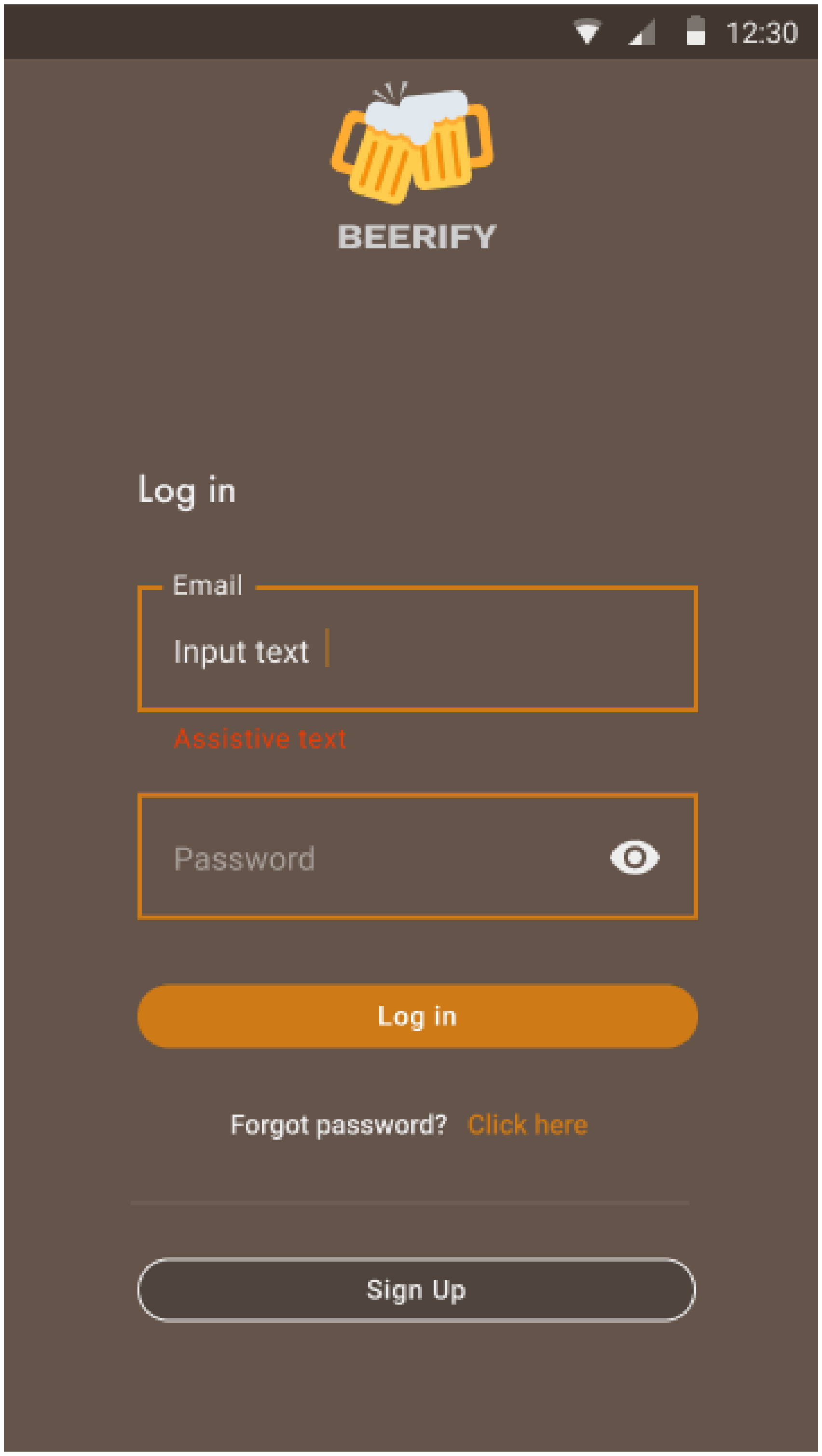


Figure 4. Login screen

7.3 Register screen

The screen where the user can register to the system.

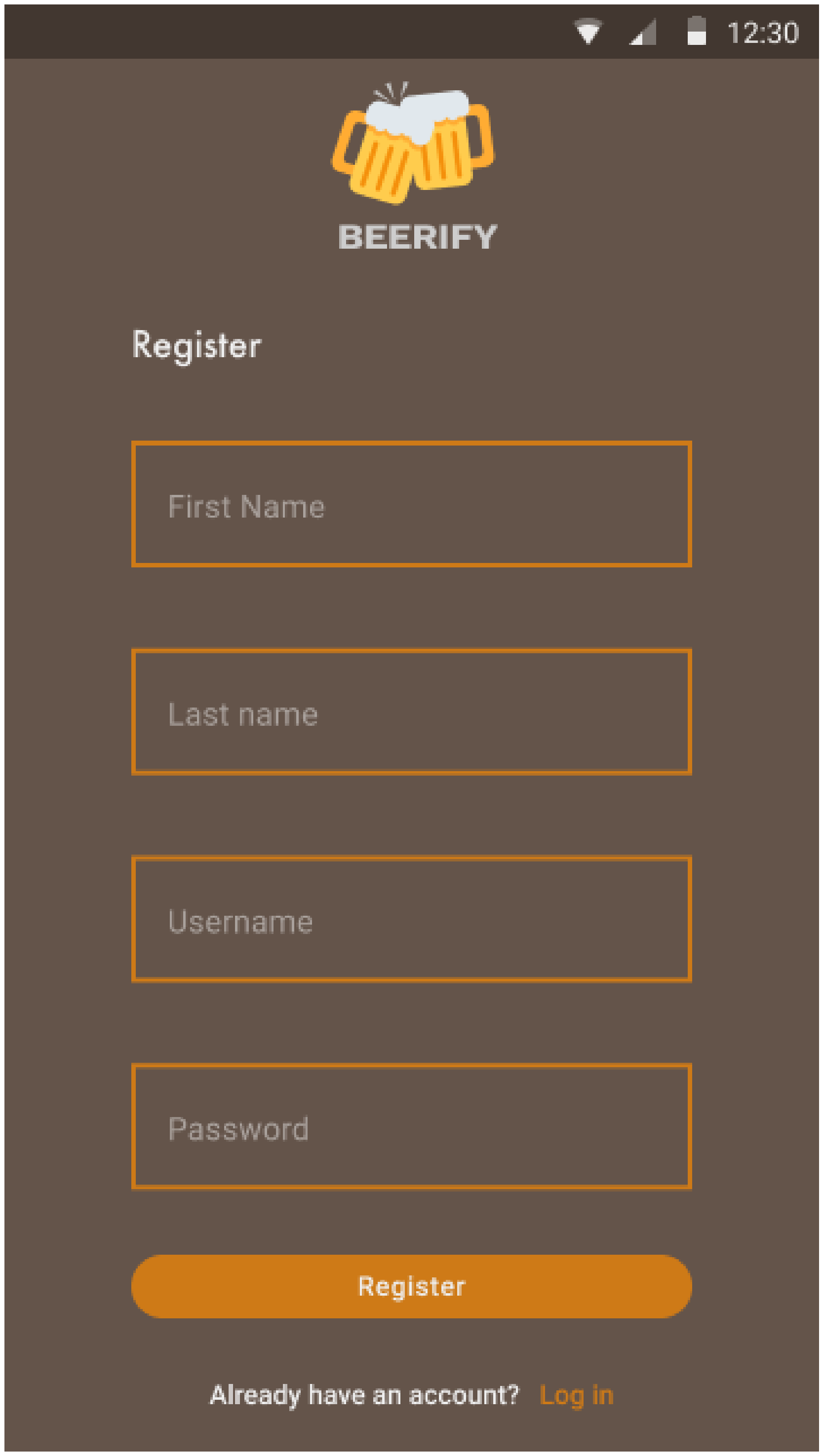


Figure 5. Register screen

7.4 List of beers screen

This is the main screen. Here you can search for existing beers in the database. This screen also helps the user to check if a beer is already in the database

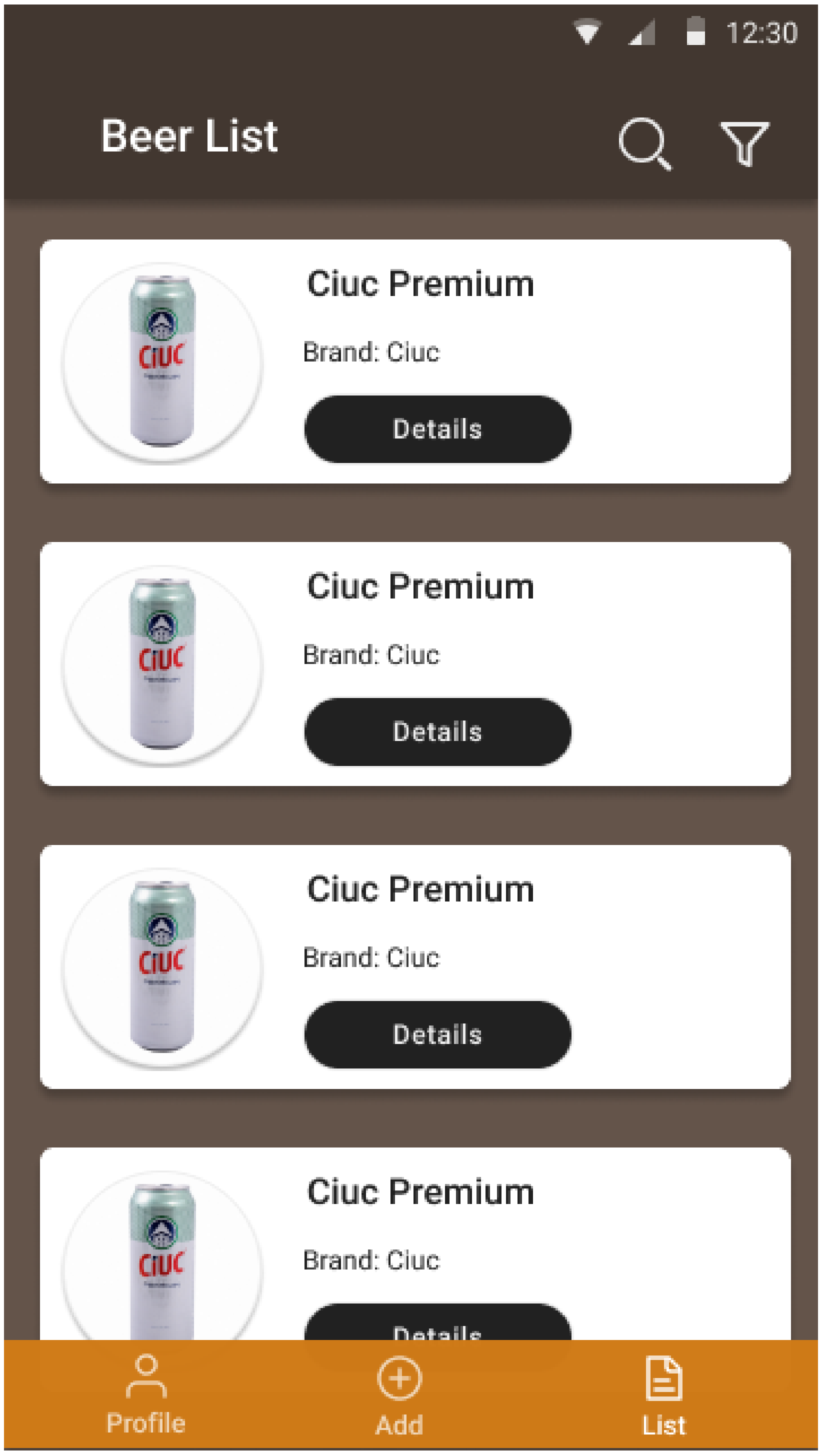


Figure 6. List of beers screen

7.5 Profile screen

In this screen the user can check the profile data

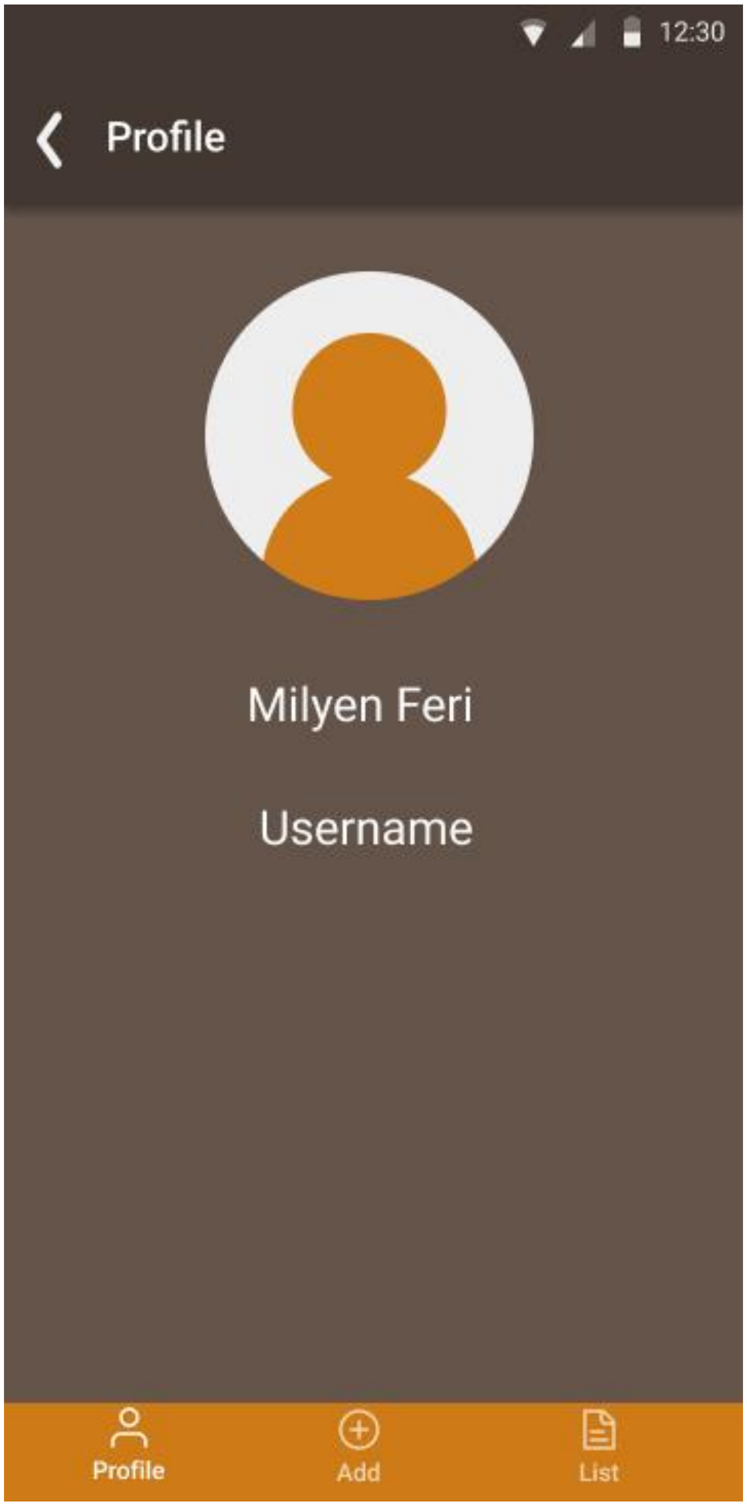
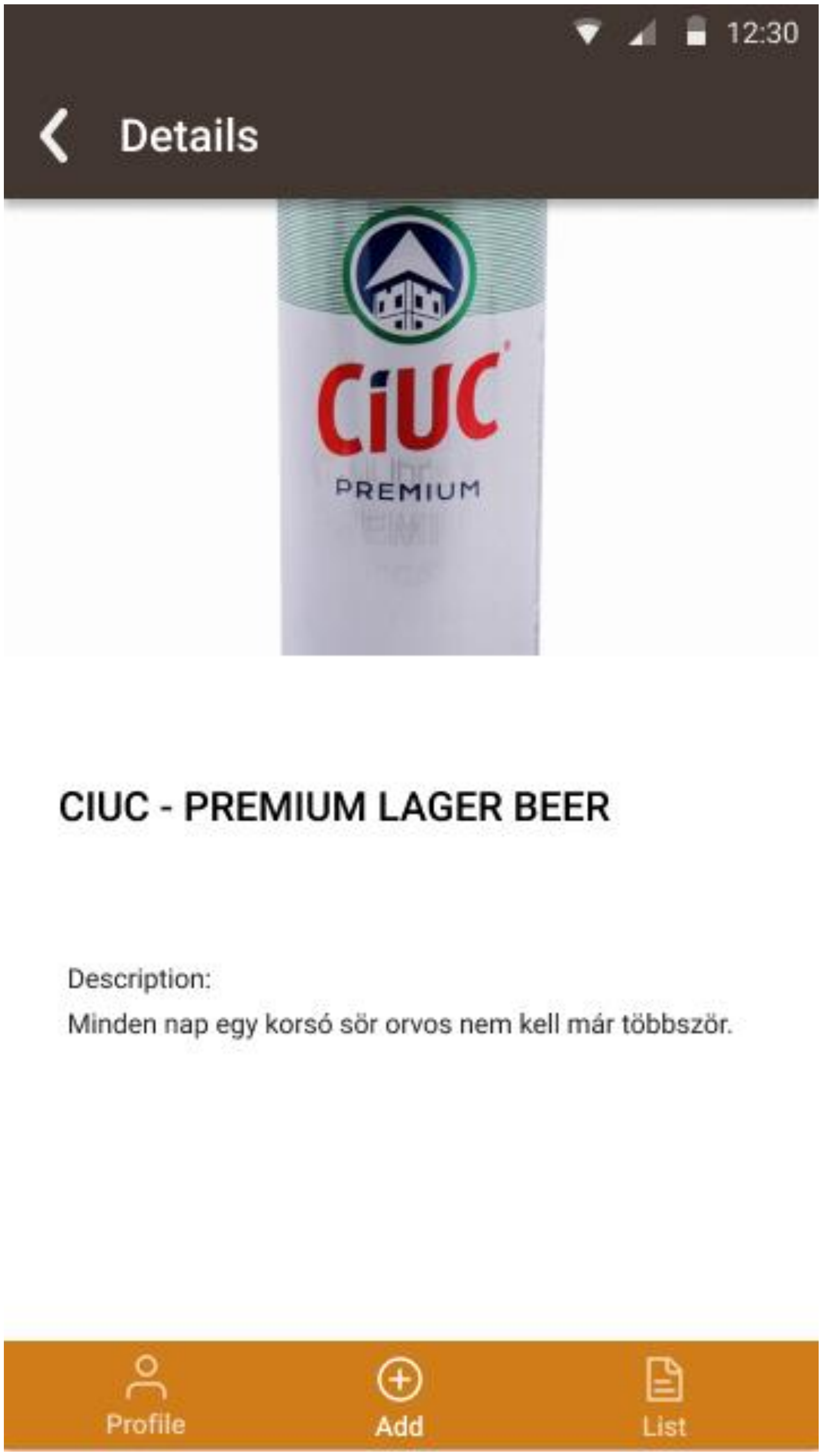


Figure 7. Profile screen

7.6 Beer detail screen

This screen has the details about a beer. Everything relevant will be found on this screen.



CIUC - PREMIUM LAGER BEER

Description:
Minden nap egy korsó sör orvos nem kell már többször.

Figure 8. Detail screen

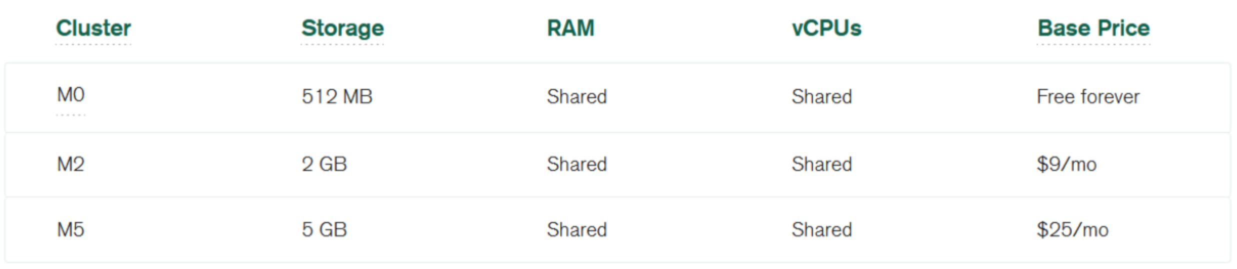
8. Detailed Project Structure

8.1 Technologies used

- Python 3.8
- Flutter 2.8.0
- MongoDB Atlas

8.2 Database

The database was created in MongoDB Atlas using the Shared M0 Cluster which has:



Cluster	Storage	RAM	vCPUs	Base Price
M0	512 MB	Shared	Shared	Free forever
M2	2 GB	Shared	Shared	$9/mo
M5	5 GB	Shared	Shared	$25/mo

Figure 9. Database prices

There are only two tables in our database:

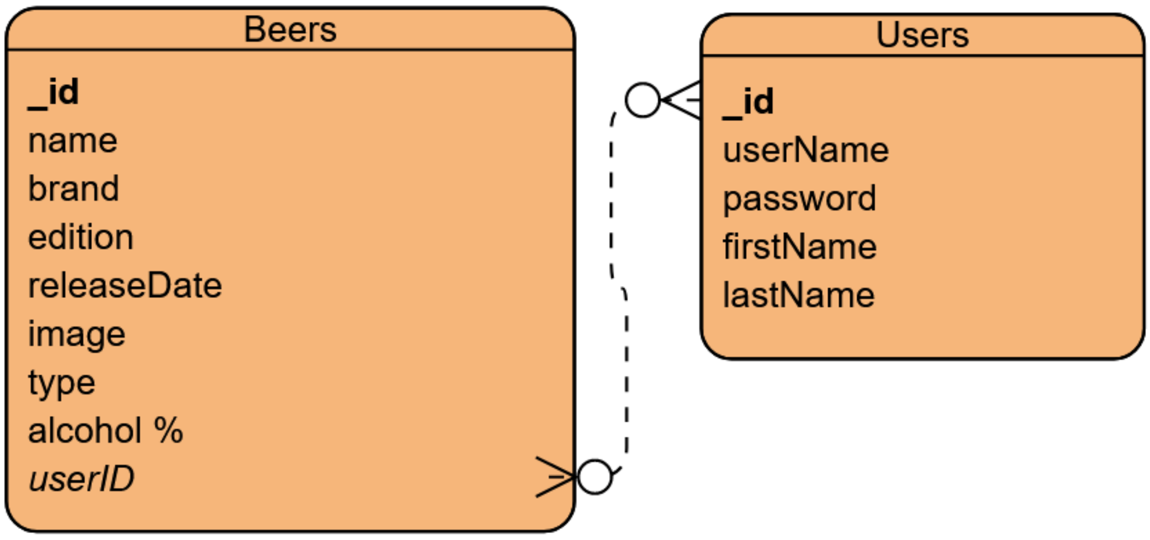


Figure 10. Database tables

The most important aspect of our project is the database of beers. We store all the relevant data regarding a beer. The users table is important since the user is required to log in, so we can know who wants to add a new beer to the database.

8.3 Image Processing

The image processing has a main function that controls the process. The image processing contains three major parts and runs in this order:

- Image compression and resize
- Image background removal
- Image similarity

8.3.1 Image compression and resize

Images must be compressed to the minimum size without losing visible quality. This was achieved with a Python script that uses mostly Pillow and NumPy. The script itself receives an input image, resizes it to 720x1280 resolution (The original size is around ~4000x5500px). We chose this resolution because the image similarity program works much faster on smaller images and also keeps the accuracy. The script then compresses the image to a thumbnail image (~40KB-120KB) and if the image is larger than 100 KB then it lowers the original quality (75) by 5 (scale between: 0-100).

8.3.2 Image background removal

Removing the images is an important aspect of the program since the image similarity compares the pixels of images. We open the images as PNG files so during the comparison the algorithm does not compare the transparent elements. We remove the background of the image via a Python script using Pillow.

8.3.3 Image similarity

The image similarity Python script has a path to an image for parameter. This input image will be then compared to all the other beer pictures in the database. The comparison itself is made with the NumPy and Pillow libraries of Python. The comparison part of the program takes the pixel tuples from both images, calculates their norm and returns the dot product of a triple division of the original image vector and the two norms. The return value is a list of dictionaries (pair of data in Python) which contains the probabilities and the name of the pictures. The list is presented in a decreasing order, so the first element of the list is the most similar image to the input image. The script then saves the found images.

The Flutter app would send the picture the user made with the phone's camera to the backend. In the backend the image-similarity script would run then send back the result to the Flutter app via REST API.

8.4 Flutter

The Flutter Application consists of two main parts, being the UI and the connection to the backend.

8.4.1 Connection

In Flutter connecting to a MongoDB database is not just a few lines of code. Everything regarding the database works with asynchronous functions and Future type objects. Future type represents the result of an asynchronous operation.

8.4.2 App router component

App router is a navigation component for our application. The main purpose of it is make the screen changing easier. The navigation component is created once, and all the screens only call the function to change the actual screen to the desired one.

8.4.3 Login and Register

The user connects to the database by logging in. When the login is finished the program sends a REST API request with the username and password.

Register uses the same REST API request method to send the necessary data to the database. Once the registration is successful the user receives a Snackbar message and can now log in.

8.4.4 Gallery

One of the main parts of the project revolves around the images of beers in the database. The database stores the images in Bison Binary type so when the application requests the images it must be transformed to a readable state. This is achieved by turning the data into base64 form then to a readable form for Flutter.

8.4.5 Search

In the application the search in the database can be achieved in two ways:

- by filtering
- image search

The filter as the name suggests filters the data by string comparison. The image search however uses an image-similarity to find the most similar image to the input image.

9. Diagrams

9.1 Sequence Diagram

9.1.1 Login

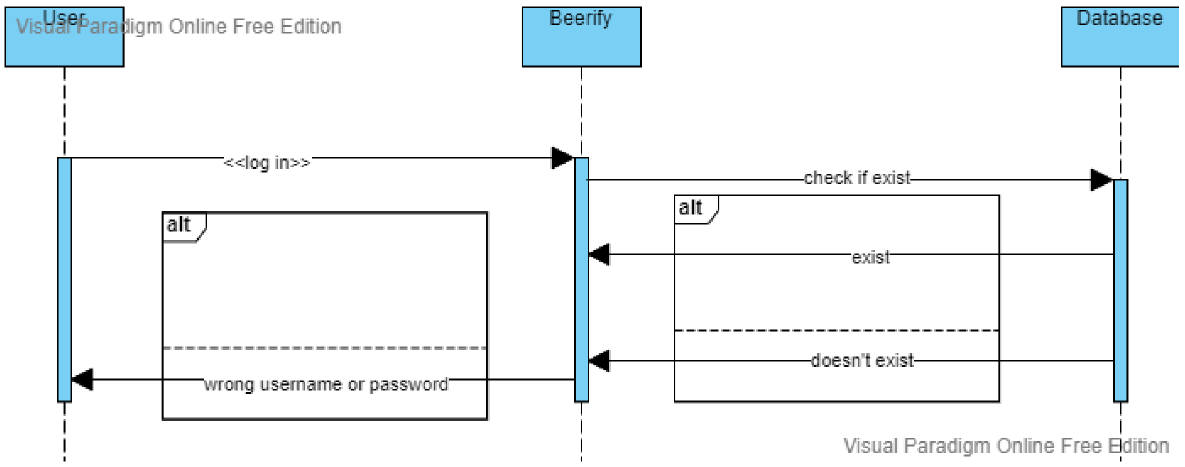

User
Visual Paradigm Online Free Edition
Beerify
Database
<<log in>>
check if exist
alt
alt
exist
doesn't exist
wrong username or password
Visual Paradigm Online Free Edition

Figure 11. Login sequence diagram

9.1.2 Register

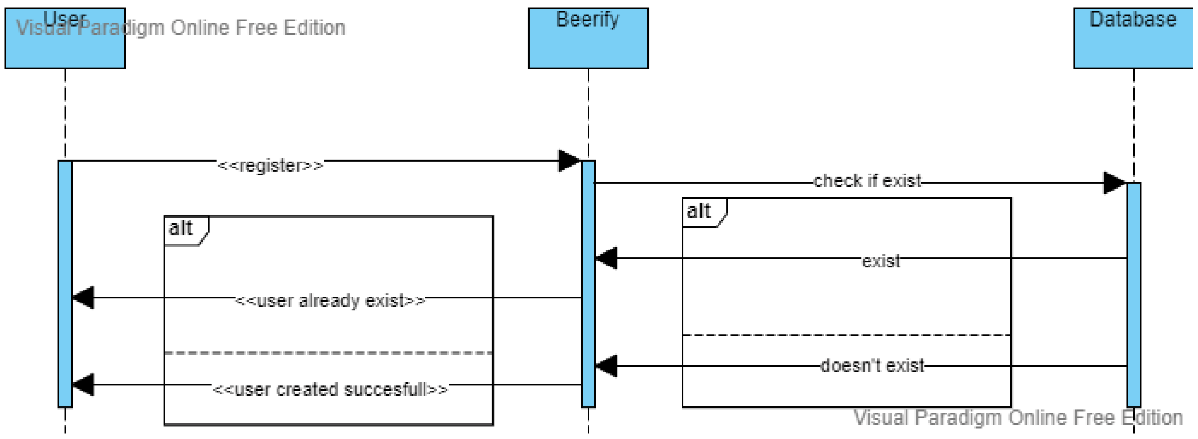

User
Visual Paradigm Online Free Edition
Beerify
Database
<<register>>
check if exist
alt
alt
exist
<<user already exist>>
doesn't exist
<<user created succesfull>>
Visual Paradigm Online Free Edition

Figure 12. Register sequence diagram

9.2 Activity Diagram

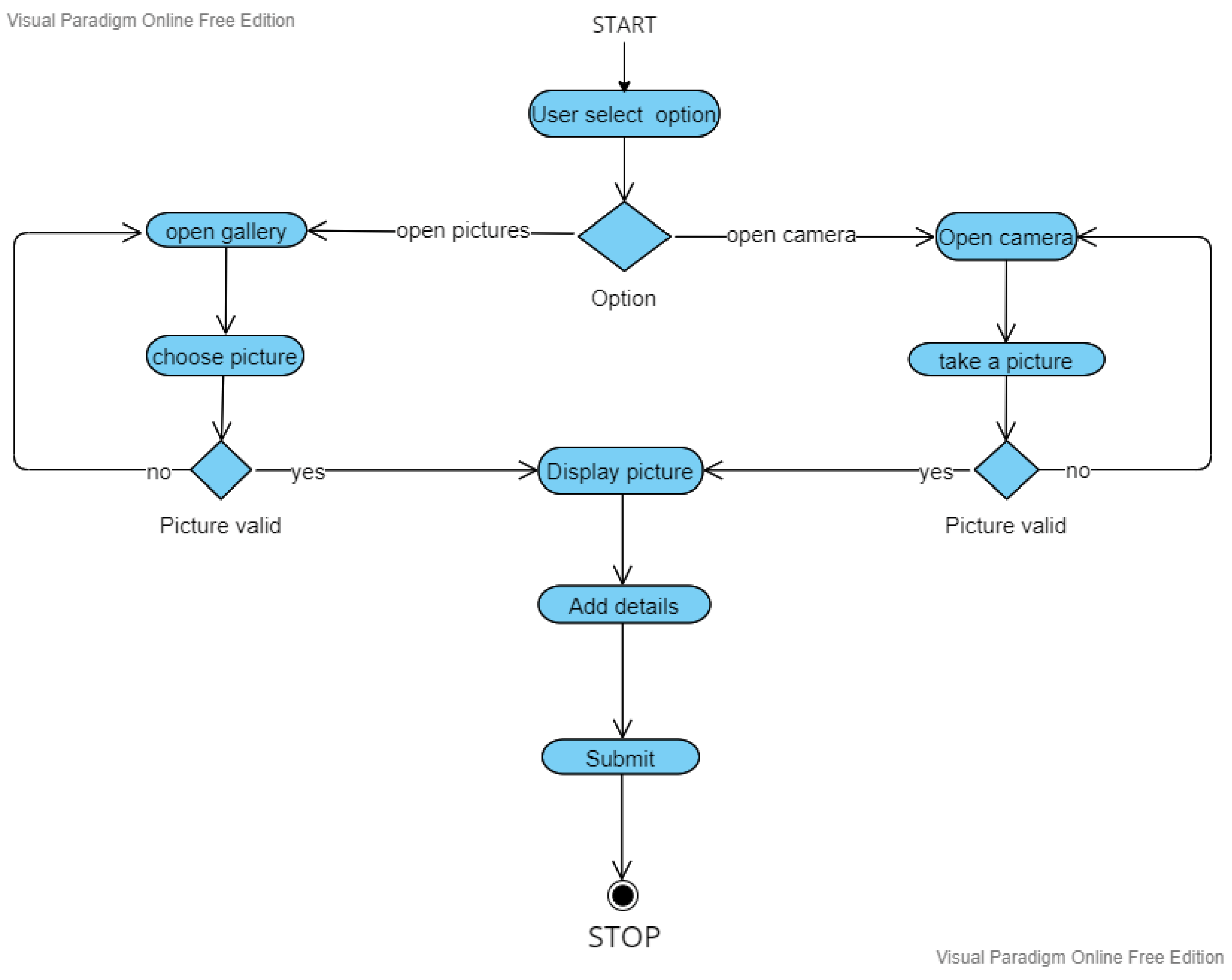


Figure 13. Activity diagram for send picture to backend

10. Further development potential

Further development potential would be to have a system where the user can add beer to the database without the help of an admin. An automatized system would do the check-up and the registration of the beer in the database.

Another important thing would be an upgraded backend and API system. The current system is free, but very slow. For long term a new upgraded backend would be perfect.

11. Summary

In conclusion, we can confirm, that we achieved most of our targets. This project was a real challenge for us, which we managed with success. Even for the smallest things, we had to collaborate, make discussions several times. The most important lesson turned out to be that every project needs to be well planned in advanced.

12. Bibliography

Cloud Application Platform | Heroku

MongoDB Atlas Database | Multi-Cloud Database Service | MongoDB

https://trello.com/b/Say7ocQD/szofter-tervez%C3%A9s-projekt

https://github.com/tibiv111/Beerify