Restaurant Recommender: 'Hungry Hunter'

Group 4's final project for the Spring 2023 CFG Degree: Software Development 1 Class

Team Members:

- Kathleen Kay Amora
- Federica Cimino
- Mungunzaya Ganbat
- Nia Jones

INTRODUCTION:

Our team aims to provide a user-friendly website that provides users accurate restaurant recommendations based on their choices. Additionally, the app also functions as a diary for recording dining experiences. It allows the users to write notes, assign personal ratings and keep track of their favorite and must-visit restaurants. In the end, the project provides users with a platform to curate a personal collection of their dining experiences, similar to a memory diary.

Our team set the following objectives:

➢ Leverage an existing webAPI (Yelp Fusion API) that would provide a wide range of restaurants in their database, which we could use in creating an accurate and efficient search engine.

➢ Create a working web app where the user can create their own account where they can see their favorite restaurants, add reviews and ratings and key in general information about their dining preferences.

➢ Allows users to set their own preferences (e.g. location, cuisine and rating) to tailor their search to their own needs.

☐ Roadmap of the report:

Each section of the report highlights the following key-points:

1. Background: explains the purpose and problems that the app is aiming to answer.
2. Specification and Design: illustrates the key features and application logic of our web-app, provides technical and non-technical requirements and an overview of the system design
3. Implementation and Execution: describes the overall trajectory of the project from planning, work distribution, projected timeline, agile workflow implementation to challenges and achievements.
4. Testing and Evaluation: provides details on our testing scope and strategy, results of functional and user testing and reporting system limitations.
5. Future Extensions (Appendix 1): lists down possible upgrades of the app.

BACKGROUND

Eating out has long been a popular pastime for socializing with friends, or bonding with families and loved ones while indulging in delectable cuisines from all over the world, some of which we cannot replicate at home. As the hospitality industry continues to rise due to economic development and globalisation, it has become increasingly challenging to always stay informed of new, upcoming and well-renowned restaurants that are worth experiencing. With the overwhelming abundance of choices, making decisions can be a daunting experience without organizing our preferences. Aside from this, manually keeping track of restaurants we would like to try out someday, come back to because of different reasons or rather, restaurants we definitely do not want to visit ever again could be quite taxing.

To address these problems, we have created this app to serve as a diary to keep track of our dining experiences by writing notes and adding our own personal rating, taking note of restaurants we love visiting or would like to visit as a "favorite," as well as providing an efficient search engine that caters to our preferences.

SPECIFICATIONS AND DESIGN

Key features:

★ *User Profiles:* Users will need to create a profile that tracks their favourite restaurants, food preferences and dietary restrictions.
★ *Efficient and Personalized Restaurant Search:* Users can key in specific details to suit their tastes such as preferred location, cuisine and rating to focus their search.
★ *Detailed Search Results:* Users will receive a detailed compilation of restaurant details from the Yelp Fusion API including name, city, cuisine, address, review count, enabling them to make informed decisions based on their preferences.
★ *Add Restaurant to Favourites*: Users can store the restaurant details as a favourite so they can get back to them sometime.
★ *Adding personalised comments and ratings:* Users can use our app as a personalised restaurant diary, allowing them to leave comments about their dining experiences, express their food preferences, and keep track of restaurants they want to try, all conveniently stored in their user profile for easy access.
★ *Contact page for development:* Users can report technical issues as well as suggest possible ideas for feature upgrades by sending their queries to the team's GMAIL account (cfgprojectqueries@gmail.com) via the Contact Us Page.

Application logic:

★ *User authentication:* The app verifies the user's credentials with the SQL database (restaurant_app) and creates a session token for the user. If user credential does not exist in the database, it will automatically add it to the users table in database table through connecting Python to SQL.
★ *User credentials verification*: The app also checks the user's input, whether the username, email and passwords follow the correct format.
★ *User profile creation:* The app saves the user's profile information in the SQL database.
★ *User filters and restaurant recommendation:* The app sends an HTTP GET request to the Yelp Fusion API with the required parameters based on the user's preferences and henceforth retrieves a list of recommended restaurants. User sees the result in a web page through Streamlit package.
★ *User favorites*. Users can favorite a restaurant they like or would like to visit, and the results will then be saved to the SQL DB, restaurants table with restaurant_ID as the primary key and user_ID as a foreign key from the users table.
★ *User review and rating:* Based on the favorite restaurants list, the app saves the user's review and rating in the review table in the SQL database with review_ID as the primary key. Two foreign keys (restaurant_ID, user_ID) were successfully incorporated as well.
★ *User favorites, review, rating display.* After successfully adding favorites, reviews and rating, by visiting the My Favorites, Reviews and Rating tabs, the user can see all his comments and favorites from the creation of their accounts up until the recent one. The data has been imported from the restaurants and reviews table in the SQL database, and has been fetched given that the user_ID matches the current session' username.
★ *User log-out and account deletion*: Users can either log-out of the session by clicking the log-out button to end the session or delete your account entirely, which will delete all the personal information in the users database.

Non-technical requirements:

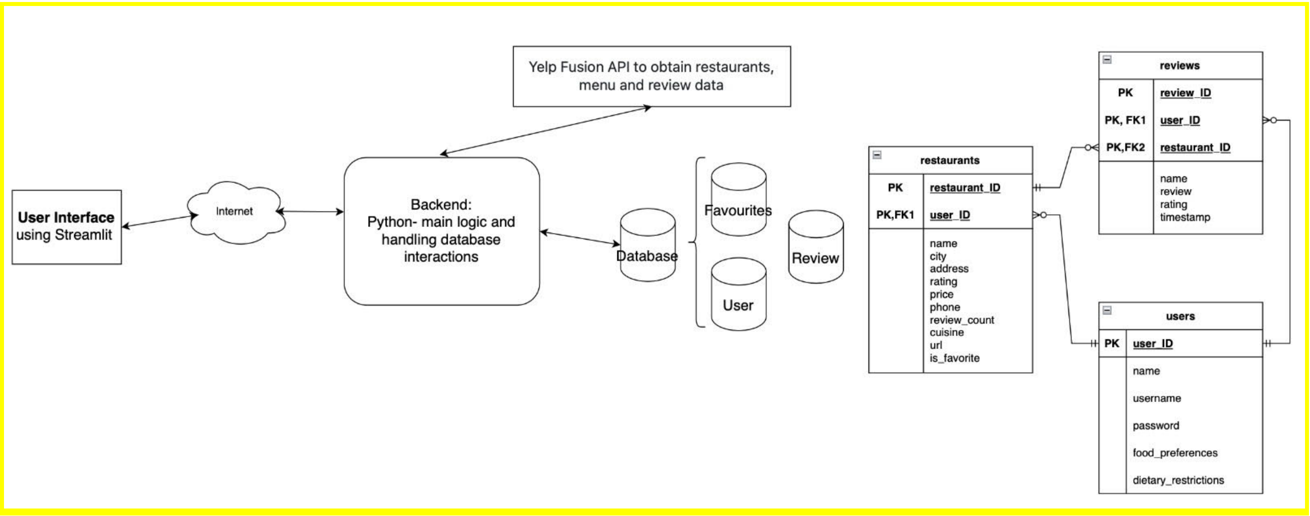
★ *Security*: The User authentication feature enables us to verify user's credentials.
★ *Compatibility:* The app's minimum requirements in Windows 11 and Mac OS environment, with Python 3.10 or later installed in the computer. Streamlit packages as well as other modules stated in the above list (e.g. pymysql, mysql-connector) should be installed.
★ *Performance:* The app is expected to load within 2 seconds
★ *Usability:* Due to the usage of Streamlit package, the user will have an ease using the app through the GUI in an web browser, providing a more user-friendly manner of navigation
★ *Documentation:* This documentation provides a detailed and comprehensive report of the software development process and the key features of our system.

Technical requirements:

When using the app, the following should be done:

★ Make sure the computer has the minimum requirement stated in the *Compatibility* section above.
★ User has to download all the folders in the project
★ User needs to ensure all the relevant libraries are installed by checking (for front-end: check environment.txt)
★ Make sure to input your own SQL credentials in the config.py file before running.
★ Users need to create the database using SQL and create the users table using database.sql file.
★ Make sure to install all packages/requirements listed above before running.
★ To run GUI, type *streamlit run home.py* in the terminal.

Design and architecture



IMPLEMENTATION AND EXECUTION

1. Development Approach and Team Member Roles

Our development approach consisted of distributing tasks based on different areas of expertise within the team. The following roles were assigned to team members:

- SQL to Python connection: Nia, Kathleen
- User Profiles GUI and Authentication: Mungunzaya, Kathleen
- YELP Fusion API and Search Engine GUI: Kathleen and Fed

Each team member worked on their respective tasks individually while maintaining communication through our WhatsApp channel or Slack group. We also collaborated as a team to provide assistance whenever needed, ensuring smooth integration between different components. To facilitate document and diagram sharing, we created a dedicated Google Drive folder.

2. Tools and Libraries

System/Environment:

- Python 3.10 - 3.11
- PyCharm 2022.3.3 (Community Edition)
- MySQL Workbench 8.0
- Git/Github

Libraries

- os: Provides functionality for operating system-dependent operations
- streamlit: A Python library for creating interactive web applications.
- requests: A library for making HTTP requests.
- datetime: Provides classes for working with dates and times
- pymysql: An open-source Python library that facilitates connecting to MySQL databases; faster than mysql-connector
- smtplib: A python module capable of sending mail to gmail accounts by creating SMTP client sessions; used for Contact Us page
- ssl: a python module to secure connection between client and server side; used for Contact Us page
- unittest - a python module for creating and running unit tests.
- Unittest.mock: is a module in the Python standard library that provides a powerful and flexible framework for mocking objects and replacing them with test doubles during unit testing

Main Frameworks:

Front-end: Streamlit

Streamlit is an open-source all-python based framework for building web applications without any front-end experience. It is widely used in the data science field for data exploration and visualisation. It has a variety of inbuilt components such as widgets and buttons that can facilitate easier and quicker creation and deployment of web apps with only using a few code lines.

API: Yelp Fusion API

Yelp Fusion API is a well-known web API that provides information and user reviews about various businesses worldwide, including restaurants. It requires a private authorization key to verify all requests. We utilised this API because it offers a diverse selection of restaurants across different cities, as well as provides comprehensive details about them. We call the API using the requests module in Python and it returns a JSON file containing the specified details of the restaurant list.

Database: MySQL Workbench

We used the MySQL Workbench to create our databases, tables and monitor changes in the SQL as we go.

In addition to these Python modules, the code expects the existence of a file named "config" containing the following variables:

- API_Key: An API key required for accessing the Yelp API.
- HOST: The host address for the MySQL database.
- USER: The username for accessing the MySQL database.
- PASSWORD: The password for accessing the MySQL database.

3. Implementation Process:

- *Set goals*: Team discussed the purpose of the app and defined key features
- *Start research*: Team started their research on possible APIs, packages, and modules to be used in the app.
- *Set up logs and repositories:* Team created different platforms (Whatsapp, Google Drive, Github, Slack) to track our progress, manage our data and code and communicate effectively.
- *Set up sprint schedule*: Team divided our task into two sprints (*discussed in detail under Agile development section)*
- *Task division*: Team divided the tasks based on the developer's area of expertise, and we employed constant communication and collaboration to help each other out when we get stuck in coding (*More details on task division in Development Approach and Team Member Roles and Agile Development sections)*
- *Coding process/Daily reviews:* Team reports and monitors each other's progress through our communication channels as well as codes were uploaded in Github for efficient collaboration and sharing. (*see Agile development section for details)*
- *Testing and Evaluation*: Team tested the isolated functionalities of our app through unit-testing, and finally checked the connection of core areas (User Profile–SQL–API) through manually integration and checking for bugs before submission.

4. Agile development (did team use any agile elements like iterative approach, refactoring, code reviews)

Our team adopted an agile approach to manage and complete the project. We incorporated several agile elements throughout the implementation process, ensuring flexibility, collaboration, and iterative improvements.

Some of the key agile practices we utilised include:

A. *Iterative development approach*: We broke down the project into manageable tasks and worked on them in two sprints: Sprint 1 focused on feature development, while Sprint 2 encompassed testing and deployment.

The timeline for the workload distribution is as follows (for detailed report, see project log):

Sprint 1: May 1st to May 14th

During the first sprint, we accomplished the following tasks:

- Trying out SQL connection to Python and creating databases and tables (Nia)
- Implementing YELP Fusion API integration, fetching and saving "favourites" restaurant results to a .csv file (later connected to SQL), and creating a GUI using Streamlit for the search functionality. Additionally, we added error handling and performed code refactoring for improved readability (Kathleen and Fed)
- Creating the frontend user interface using Streamlit, establishing SQL connection to Python, and implementing features such as sidebar navigation for favourites, account deletion, log out, food preferences, login, and security measures for changing email passwords and usernames (Mungu).

Sprint 2: May 15 to May 26th

In the second sprint, we accomplished the following tasks:
CODE WRITING/REFACTORING:

- Adding review and rating functionality to the API/Search Results GUI (Kathleen)
- API code refactoring and exception handling (Kathleen and Fed)
- Successfully coded and connected SQL to Python using pymysql (Kathleen)
- SQL code refactoring (Kathleen)
- Debugged errors in the front end such as sign-up/log-in issues, DuplicateWidgetID error issues, made the front end part work correctly (Kathleen)
- Writing Contact Us Page (Kathleen)
- Successfully coded the User Authenticator Class with regex and exception handling for user authentication (Munguu)
- Refactored the sprint1 pages in the front-end part such as (Munguu): Login and Security functionality (security.py), login_signup page by creating an instance of UserAuthenticator class in the function, took part in handling DuplicateWidgetID error issues alongside Kathleen

UNIT TESTING: Testing and evaluating the program

- Unit Testing classes YelpAPI and RestaurantFinder (Kathleen and Fed)
- Unit test for class RestaurantToDB (Nia and Fed)
- Unit testing for front-end pages (login_signup, security, delete_account, food_preferences, send_email, user_authenticator class) (Munguu)

INTEGRATION TESTING: Ensuring proper connections between the front-end, back-end, and SQL database

- Initial successful integration of front-end, back-end, SQL; fixed crucial issues (Kathleen)
- User testing (Fed, Nia, Mungunzaya)

DOCUMENTATION: Documenting the project and recording a website demo for presentation

- Completing project documentation (Nia, Kathleen, Fed, Mungu)

B. *Refactoring*: As part of our commitment to maintain code quality and readability.

C. *Code Reviews*: Collaborative code reviews were crucial for the completion of our project. Regularly, we reviewed each other's code, providing feedback, identifying potential issues.
By incorporating these agile elements, we were able to respond to changes efficiently, adapt our approach based on feedback, and ensure a high-quality implementation. The iterative nature of agile development allowed us to deliver incremental value while maintaining a focus on meeting the project's goals and requirements.

D. Project/Code Management
To manage the project effectively, we adopted the following strategies:

- Maintaining a "project log" file to track our daily contributions, report challenges faced, and document progress.
- Creating a GitHub repository and establishing a branch where we could push our code for easier code management and collaboration.

E. *Daily stand-ups:* we ensure to meet up daily after CFG classes to update each other about our progress and upcoming tasks as well as things we need help on.

5. **Challenges**
The major challenges we faced included:

A. *Integrating different parts of the code.* The main challenge we faced was integrating user profiles, search results, and databases into a cohesive application, which we addressed by creating separate SQL tables and successfully connecting them with foreign keys, ultimately overcoming the challenge.

B. *Learning to use Streamlit effectively.* Even though Streamlit is easier to use than Flask, Django or even hard-coding in HTML itself, we still needed to devote time to learn the basics as we code, as most of us did not use this module before. However, as we were committed to making a working "front-end", we incorporated this in the project, and in return, we learned a lot and improved our problem solving skills by researching and asking each other for help.

C. *Time management was also a challenge as we had to balance our coursework, homework, and job responsibilities*. Most of us could only work after our CFG class hours and could only devote weekends to focus fully on the project. However, we tried our best to deliver our tasks and we often consulted each other for tasks we found difficult, hence, we were able to solve them quicker.

D. *Getting accurate location coordinates using a free Geolocation API.* Due to challenges with free Geolocation APIs, we were unable to retrieve accurate location coordinates for users, despite attempting various options. Limited access tokens or paid services from popular Geolocation APIs also posed setbacks. Consequently, we opted to simplify the process by requesting users to provide a specific city instead.

6. **Achievements**

Our main achievement was successfully collaborating as a group, sharing our successes, and supporting each other whenever needed. We adapted to challenges by seeking help, sharing knowledge, and making necessary adjustments to our implementation approach. Overall, the team worked together to overcome obstacles to complete the project.

The team made small achievements throughout working on the project such as meeting deadlines to complete certain tasks and setting out a plan to keep the project flow moving.

Furthermore, we can highlight our achievement of setting a clear plan for the project by incorporating an Agile Development approach. By establishing a well-defined roadmap and breaking down the work into manageable tasks, we were able to maintain clarity and direction throughout the project's duration. This strategic approach ensured that everyone knew their roles and responsibilities, facilitating efficient collaboration and progress.

Overall, our team's collective efforts, dedication, and ability to adapt to challenges allowed us to achieve success in completing the project. We can take pride in our accomplishments, both big and small, as they reflect our commitment to delivering the final work while working harmoniously as a team.

TESTING AND EVALUATION

- Testing strategy:

  We also incorporated testing strategies such as:

  1. Unit tests: We conducted unit-tests for more than half of our functions (crucial core functions excluding the functions related to streamlit deployment) which would focus on evaluating the backend, API and database functionalities in isolation. We also used the mock and patch techniques included in the unittest module to simulate and replace the calling of modules and establishing connections during unit-testing.
  2. Manual test: We simulated real user interactions with the system still in isolation. We cued in dummy data to see whether items perform as expected
  3. Integration test: We called and combined functionalities of each area into one single, cohesive app, mimicking a smooth flow. We also refactored functions: either split to smaller functions or group them into classes for better organisation.

- Functional and user testing

Functional tests are done in all three areas (SQL, API, User Profiles) of the app, we did unit tests, manual tests and integration testing. For more details, see Appendix II: Testing details. Due to limited resources and time constraints, system testing and acceptance testing were not performed, but it is recommended to include diverse tests in the future to enhance bug detection and resolution.

User testing was completed using some of the members in our group. We each used the Restaurant Recommender: Hungry Hunter web page that we created to create a user login with an email address and password and selected our preferences and dietary requirements. We searched for a location and cuisine, saved chosen restaurants to our favourites and added reviews and ratings for the restaurants we had selected. This was to make sure that the favourite restaurants and the review ratings were saving correctly into the sql tables.

- System limitations

    Only running on the web/ computer. Does not support mobile etc
    App runs when Streamlit is installed
    Front - end went minimalistic, did not have time to upgrade interface design
    No forgotten password retrieval
    No deletion of favourites (time constraints)
    No hashing of passwords
    The system might be sensitive to SQL injection because of no hashing of passwords.

CONCLUSION

This project creates an easy-to-use restaurant web app that provides users with quick and accurate recommendations for restaurants catered to their preferences. Users can also favorite their preferred restaurants, write about their experiences for easy tracking, and share their memories.

For this project, we utilised Streamlit library for our front-end, YelpFusion API for our search results and usage of pymysql module and MySQL Workbench for our databases. We employed an agile-scrum development method by doing two sprints during development to reach our set goals. We also incorporated testing strategies such as unit-testing and manual testing to check the internal and external functionalities of the app. As this is an early version of the app, we are positive that it serves its basic functions to the user in providing restaurant recommendations, keeping track of their favorite places to eat and taking note of their personal dining experiences.

APPENDIX 1:

FUTURE EXTENSIONS

1. Restaurant Reservations:
   Adding a feature that allows users to make reservations at restaurants directly through the app. Possible integration with restaurant reservation systems or our own system that sends data to individual restaurants.
2. Place Order to Delivery:
   Allowing users to place orders for delivery or pickup from restaurants directly through the app. Possible integration with restaurant ordering and delivery systems.
3. Order Tracking:
   Providing users with the ability to track their orders in real-time, and
   Possible integration delivery tracking services.
4. Payment Gateway:
   Integrating a payment gateway like Stripe or PayPal would allow users to make payments for reservations or orders directly through the app.
5. Integration with Taxi Apps:
   Providing users with the ability to book a taxi or ride-sharing service like Uber or Bolt directly from the app.
6. Social Media Integration:
   Integrating with social media platforms like Facebook and Twitter would allow users to share their experiences and recommendations with friends and followers.
7. Membership Programs:
   Building a loyalty program for frequent users of the app could help to keep users to continue using the app.
8. Multilingual Support:
   Providing multilingual support for the app would increase the number of users willing to use the app.
9. Algorithmic Search and Recommendations
   Employing an algorithm that would recommend similar restaurants depending on how the user interacts with the app and its search history. Similar to how Netflix recommends movies based on our watch history.

APPENDIX 2:
Testing Details

Testing	**Area to be tested**	**Process**	**Expected Output**
Unit Test	API connection	Use mock and patch library under unittest module to mimic dependencies	Tested valid and invalid connections. For valid, .json result of the extracted is produced, for invalid connections, we raised an Exception
Unit Test	User Profiles	Unit tests utilise the mock and patch libraries to mimic dependencies, ensuring controlled and predictable testing conditions. By importing the following in the unittest: from unittest.mock import patch, MagicMock ● MagicMock: for creating mock objects ● Patch: for patching the functions in the streamlit module ● import smtplib: for mocking the smtplib.SMTP class	Testing valid, invalid, and edge cases for the 6 front_end pages: Valid connections are expected to produce JSON results, while invalid connections raise exceptions to handle error scenarios. Edge cases are in the following output: ● Valid with Empty JSON Result ● Invalid with Exception (Different Error) ● Valid with Non-JSON Result ● Invalid with Exception (Custom Error)
Unit Test	SQL DB	Imported the modules needed, using mock and patch libraries to mimic the connection and the database.	Testing valid and invalid cases for adding data to the database Valid cases are expected to produce the data from the mock table. Invalid cases are expected to raise an AssertionError.
Integration Test	Smooth flow from Log-in, API search result to DB	Import necessary modules and function in correct files, integrating two home pages to one, facilitating calling of variables across different .py files. Additionally creating function for successfully linking	Seamless transition from log-in (credentials go to DB), access to restaurant search once logged in → able to search for restaurants, save favorites and add reviews and ratings. Successfully saved these in SQL and can be retrieved once My Favorites page is visited.