<u>Final Project – Software Development Using AI Tools</u>

<u>Submitters: Tomer Juster and Sione Milhem</u>

<u>Our Project: Travely: an AI vacation planner</u>

In our project we used several APIs and many tools and technologies that we learned in class in order to create a trip planner that plan trips effectively.

<u>Problem we tried to solve:</u>
Planning a trip involves considering many factors, including searching for flights, hotels, activities, and budget management. This project addresses these challenges by providing a comprehensive solution that simplifies trip planning through an intuitive api.
Our API can be used at http://tomerandsionefinalproject.eastus.azurecontainer.io/
We know that the requirement of the assignment do not require a UI but we also created in the project a local react app that demonstrates how someone can use our API and display it's results with a nice UI.

<u>How to run the react app locally:</u>

1. Go in the terminal to the frontend folder
2. Type: npm install
3. Type: npm install @mui/material @emotion/react @emotion/styled
4. Type: npm start
5. The app should start at localhost:3000

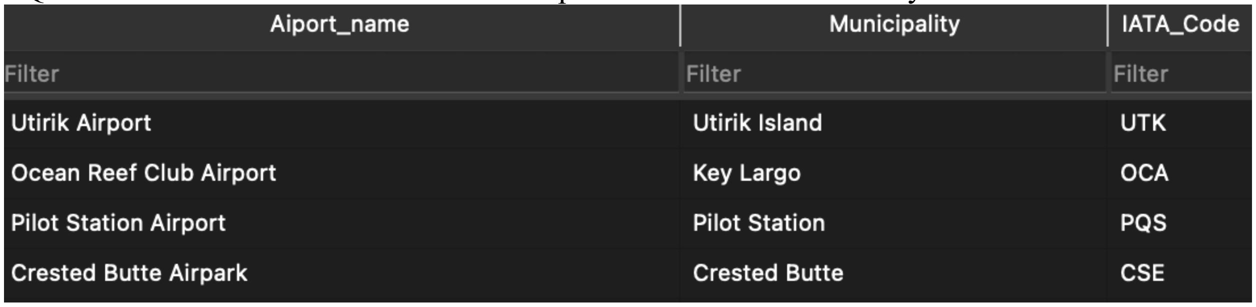
<u>Technologies and APIs Used</u>
Backend: python, FastAPI, OpenAI API, SerpAPI, Tripadvisor API
Frontend: React.js
Database: SQLite
CI/CD: GitHub Actions, Docker, Azure
Unit Testing: unittest
API Testing: Postman

<u>Project Architecture</u>
Our FastAPI provides endpoints for trip planning, and for retrieving airports IATA codes.

/search-for-your-preferred-airports/: Fetch IATA codes and airport names for origin and destination cities from our IATA SQL database. We require these codes in the main function of the trip-plan because we need the codes for APIs like SerpAPI and we also wanted the user to enter his preferences for the airports.

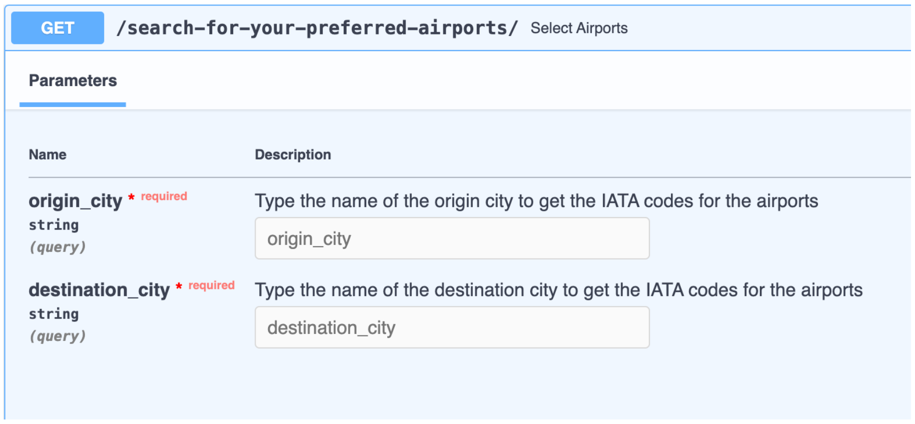
Database Schema

SQLite database stores IATA codes and airport names based on the city names.



Aiport_name	Municipality	IATA_Code
Filter	Filter	Filter
Utirik Airport	Utirik Island	UTK
Ocean Reef Club Airport	Key Largo	OCA
Pilot Station Airport	Pilot Station	PQS
Crested Butte Airpark	Crested Butte	CSE

We use it to provide fast and reliable access to data essential for the trip planning and also to allow the user to see the list of airports in the destination he wants to travel to, so he can choose his favorite option.

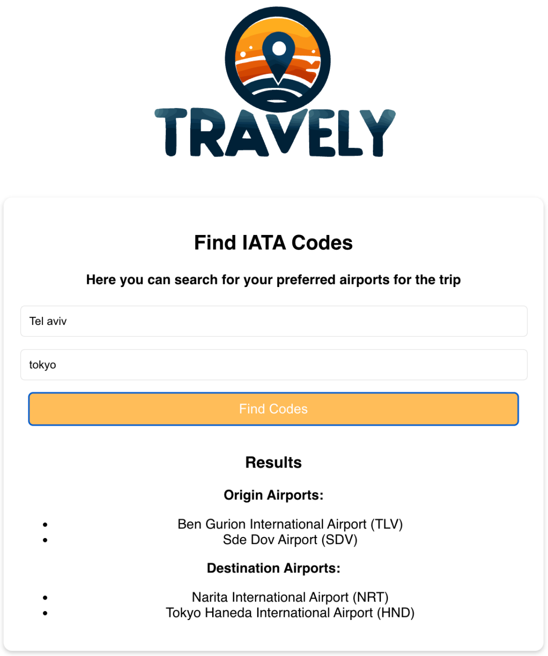


GET /search-for-your-preferred-airports/ Select Airports

Parameters

Name	Description
origin_city * required string (query)	Type the name of the origin city to get the IATA codes for the airports origin_city
destination_city * required string (query)	Type the name of the destination city to get the IATA codes for the airports destination_city

How it looks in the React app:



Find IATA Codes

Here you can search for your preferred airports for the trip

Tel aviv

tokyo

Results

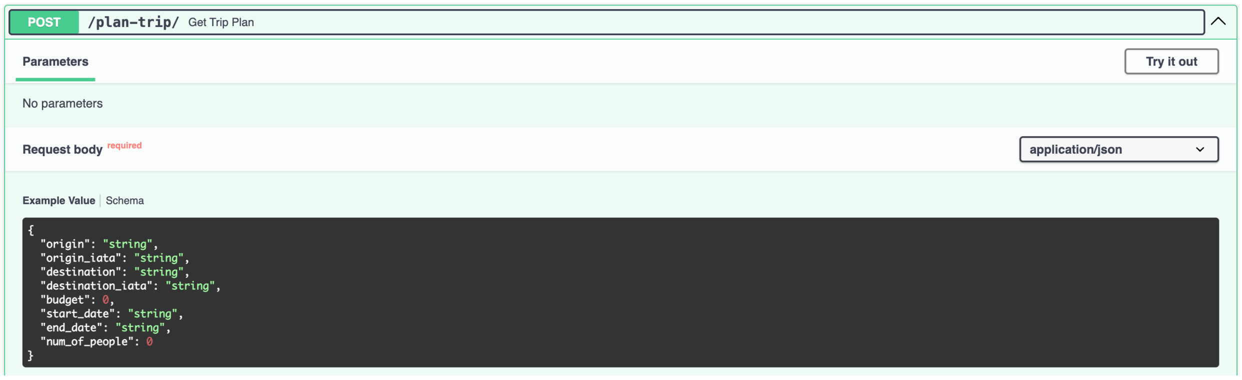
Origin Airports:

- Ben Gurion International Airport (TLV)
- Sde Dov Airport (SDV)

Destination Airports:

- Narita International Airport (NRT)
- Tokyo Haneda International Airport (HND)

/plan-trip/: Submit trip details and receive an HTML with comprehensive trip plan.


POST /plan-trip/ Get Trip Plan

Parameters — Try it out

No parameters

Request body required — application/json

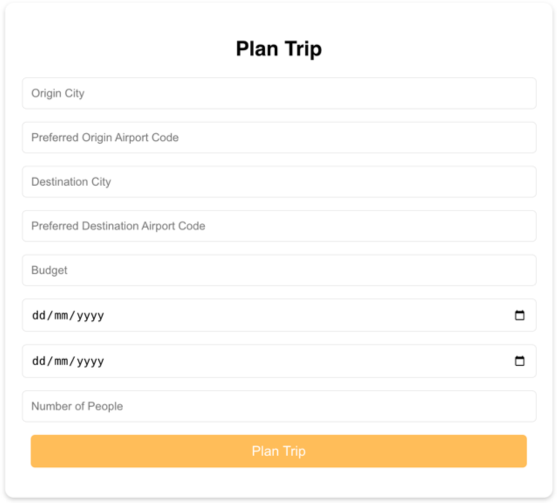
Example Value | Schema

```
{
  "origin": "string",
  "origin_iata": "string",
  "destination": "string",
  "destination_iata": "string",
  "budget": 0,
  "start_date": "string",
  "end_date": "string",
  "num_of_people": 0
}
```

In this endpoint we request from the user to input some details about his trip.
Then in the background we call several APIs:
Tripadvisor: we get data about attractions and restaurants in the destination.
SerpAPI: we get data about flights and hotels, taking into consideration the inputs of the user like dates budget and number of people.
openAI: after we get all the data from our APIs we go over it, format some parts of it, and also use the data to create more data. For example we take the location of each of the attractions and restaurants and build another sorted data structure that holds the distance between them – this in order to make the plan for each day in the trip with attractions that are closer to each other and much more logical.
Eventually we get a big amount of data that is structured like this:

```
Destination: {trip.destination}
Budget: {trip.budget}
Dates: From {trip.start_date} to {trip.end_date}
Number of people: {trip.num_of_people}
Flights Info: {flights_info}
Accommodation Info: {top_hotels}
How to arrange the activities:
{location_based_activities}
Activities Info: {activities_info}
Activities websites:
{activities_details}
```

We input all this data to OpenAI's API and instruct it to generate a detailed trip plan using a prompt we created.
How it looks in the react app:


Plan Trip

Origin City
Preferred Origin Airport Code
Destination City
Preferred Destination Airport Code
Budget
dd/mm/yyyy
dd/mm/yyyy
Number of People

Plan Trip

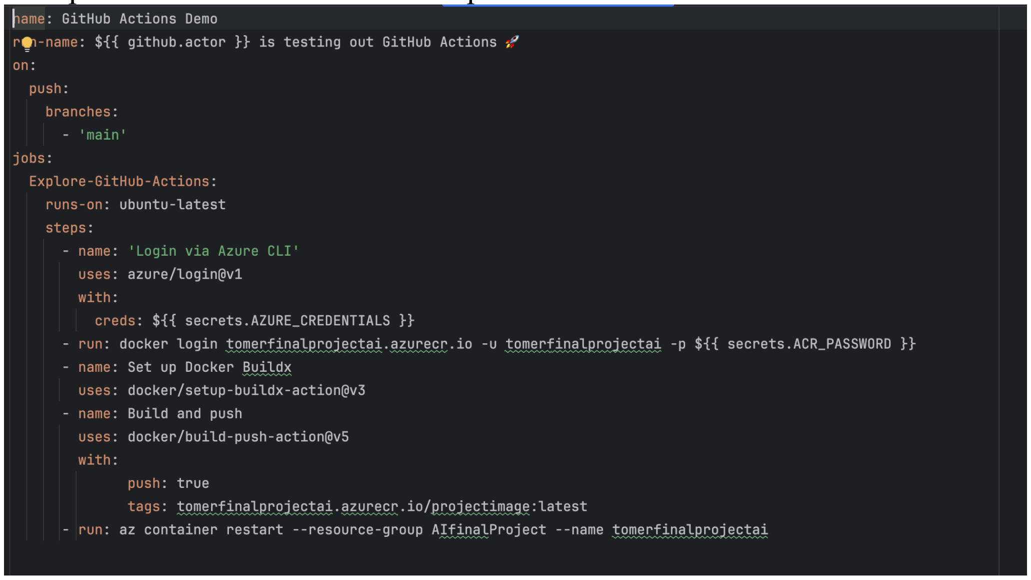
Our function then formats the text that was generated by openAI to fit our html response and we also use OpenAI dall-e in order to create a nice image that fits our trip (can see an example in the video).

Unit Testing and Coverage

We implemented the unit tests using Python's unittest framework. we focused on checking functionalities of the code like distance calculations and sorting, API data fetching, database accessing and html formatting. The tests ensure the application's working as we expect and reliably.
In addition we got a coverage of about 70%. We didn't write tests for OpenAI's responses since we saw the API was reliable and instead focused on testing our own application logic and handling of these responses.
In some of the tests we took a TDD approach which helped us to build the function to provide the output we wanted (for example in the IATA code fetching and distance calculations).

CI/CD Pipeline

Our project uses GitHub Actions for continuous integration, we wrote a yml file that on each push to our main branch starts the deployment process, with Azure hosting the application, we login with credentials to azure and we build and push the new docker image. After all this process we restart our container which allow the latest image to be used.
This process ensured us automatic updates of our API.



```yaml
name: GitHub Actions Demo
run-name: ${{ github.actor }} is testing out GitHub Actions 🚀
on:
  push:
    branches:
      - 'main'
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - name: 'Login via Azure CLI'
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - run: docker login tomerfinalprojectai.azurecr.io -u tomerfinalprojectai -p ${{ secrets.ACR_PASSWORD }}
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      - name: Build and push
        uses: docker/build-push-action@v5
        with:
              push: true
              tags: tomerfinalprojectai.azurecr.io/projectimage:latest
      - run: az container restart --resource-group AIfinalProject --name tomerfinalprojectai
```

Challenges we had and Solutions

Throughout the development of this project, we faced several challenges.

1. Integration of Different Technologies
   The project required integration of the FastAPI with several APIs alongside interaction with our SQLite database. Also connecting our React app to use the API.
   Solution: We defined endpoint specifications for our API early in the development process and used TDD methodologies in our development.
2. Ensuring Real-Time Data Consistency

Our application relies on real-time data for flights, hotels, and activities. Ensuring the consistency of this data presented a challenge. In our POC we had a python program that does the scraping of the informaion alone, but it had relability problems and didn’t work on multiple operating systems.
<u>Solution:</u> we used known APIs like SerpAPI and Tripadvisor, which allowed us to get up-to-date information reliably.

3. <u>Unit Test Coverage</u>
   Achieving high test coverage for unit tests was a bit challenging, especially for functions that had external dependencies, like third-party APIs.
   <u>Solution:</u> We increased the coverage by writing additional tests for functions using mocking to simulate the responses of external APIs. This approach allowed us to test our code's reaction to a variety of API response scenarios without making actual API calls.
4. <u>CI/CD Pipeline</u>
   The initial setup of the CI/CD pipeline was time-consuming, and a bit confusing since it required us to integrate createing a docker image, pushing it to azure and making sure the container has the latest version of the code.
   <u>Solution:</u> using the Github actions platrform was more convenient and intuitive. After watching online tutorials and using stackoverflow, we managed to create a working CI process.

<u>Conclusions</u>

We think our project successfully demonstrates the use of AI tools and software development techniques learned in the course in order to simplify trip planning.
While our project is fully functional, if we had more time we would like to add more features like:

- Recommendations for trips without user input.
- Real-Time Notifications regarding flight changes, hotel booking statuses etc.. and alerts for price drops or special deals.
- Integration with Social Media Allowing users to share their plans or experiences on social media directly from our platform.
- Offline Functionality: Developing offline capabilities that allow for example for users to open an account and save their trip plans in a better way.

This project was a challenging task. It pushed our technical skills and required us to think creatively to solve problems. The project was not only a test of our coding skills but also our ability to work as a team, manage time and create something new from zero with integration to cloud platforms.
Despite the challenges, the project has been incredibly rewarding. It allowed us to see the results of our work in the form of a functional application. It taught us lessons in software development, development methodologies like unit tests and TDD, teamwork, the importance of simple design for the user, getting help from online resources like stackoverflow and more.

In conclusion, while more time would allow us to bring more enhancements, we are proud of what we've accomplished and excited about the potential for growth. we look forward to carrying these experiences into our future work and projects.