DevOps Project

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500 043

Certificate

This is to certify that it is a bonafied record of practical work done by Mr. / Ms. G.Reshma, T.Pujita, B.Priyanka bearing the roll no. 23951A057K,23951A0573, 24955A0526 of B.Tech CSE class Computer Science Engineering branch in the DevOps laboratory during the academic year 2024-25 under our supervision.

Head of the department **Lecturer – in charge**

Signature of External Examiner **Signature of Internal Examiner**

DevOps Project On: Dockerized Travel App using Flask

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500 043, Telangana

Department of CSE
Bachelor of Technology
in
CSE

-BY

G.Reshma
23951A057K

T.Pujita
23951A0573

B.Priyanka
24955A0526

DECLARATION

I certify that

a. The work contained in this report is original and has been done by me under the guidance of my supervisor (s).
b. The work has not been submitted to any other Institute for any degree or diploma.
c. I have followed the guidelines provided by the Institute for preparing the report.
d. I have conformed to the norms and guidelines given in the Code of Conduct of the Institute.
e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Place: Hyderabad **Signature of the Student**

Date: **Roll No:**

ABSTRACT

In today's fast-paced software development environment, ensuring quick, consistent, and reliable deployment of web applications is essential. This project showcases the complete **DevOps lifecycle** applied to a **Flask-based Travel Web Application**, with a strong focus on **Dockerization** and **CI/CD automation using GitHub Actions**.

The main objective is to **automate the process of building, testing (if applicable), and containerizing the Flask app**, making deployment faster and more reliable. The journey begins with version-controlling the codebase using **Git and GitHub**.

From there, **GitHub Actions** is integrated to implement a **Continuous Integration and Continuous Deployment (CI/CD)** pipeline. This pipeline is triggered by events such as code pushes or pull requests. The CI/CD process includes optional testing, building the application, and creating a **production-ready Docker image**. This Docker image can then be pushed to a container registry or deployed on a hosting platform.

Through this hands-on project, we explore key DevOps principles such as:

- **Automated builds**
- **Reproducible deployments using Docker**
- **Consistent environments**
- **End-to-end pipeline automation**

This project demonstrates how combining **Flask**, **Docker**, and **GitHub Actions** can simplify the delivery of modern web applications and promote a culture of automation and reliability in software engineering.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1About CI/CD Pipelining

This project centers around a **Flask-based Travel Application** that helps users explore travel destinations through a simple and intuitive web interface. The application is built using **Flask (Python)** for backend logic and includes HTML/CSS for the frontend. It can display destinations, travel tips, or even mock booking options depending on your implementation.
To ensure **rapid, reliable, and consistent deployment**, the app is **containerized using Docker** and integrated with a **CI/CD pipeline via GitHub Actions**. The goal is to automate every step of the software delivery process—from code updates to final deployment.

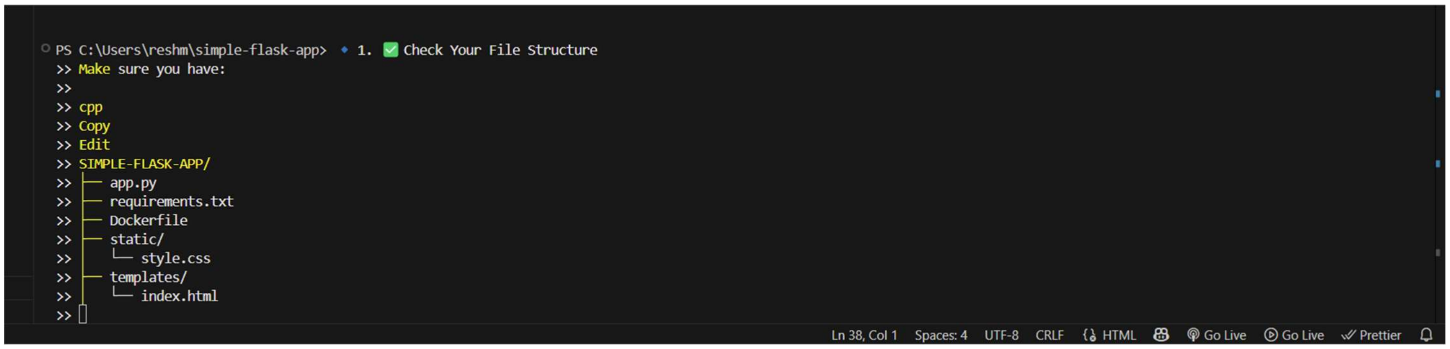
- **Continuous Integration (CI)** ensures that every code push is automatically built and optionally tested.
- **Continuous Deployment (CD)** automates the process of deploying the application as a container to a production-like environment, eliminating the need for manual intervention. This setup allows for:
- Consistent behavior across different systems using Docker containers.
- Faster iterations and feedback loops through automated builds and deployment.
- Real-world exposure to **DevOps practices** like version control with Git, automation via GitHub Actions, and containerization with Docker.
  By combining Flask with Docker and CI/CD, this project highlights the **modern DevOps approach** to software engineering—emphasizing automation, reproducibility, and developer efficiency.

1.2 Requirements

To implement **Dockerization** and **CI/CD** for the Flask-based Travel App, the following tools and configurations were used:

1.2.1 GitHub Repository

- The Travel App's source code is hosted in a **GitHub repository**.
- The repository contains:
    - app.py – the Flask backend logic
    - templates/ – folder containing HTML files (e.g., index.html)
    - static/ – folder containing CSS files (e.g., style.css)
    - Dockerfile – instructions for containerizing the app
    - requirements.txt – Python dependencies
    - .github/workflows/ci.yml – GitHub Actions workflow file for CI/CD automation



```
PS C:\Users\reshm\simple-flask-app> ◆ 1. ✅ Check Your File Structure
>> Make sure you have:
>>
>> cpp
>> Copy
>> Edit
>> SIMPLE-FLASK-APP/
>> ├── app.py
>> ├── requirements.txt
>> ├── Dockerfile
>> ├── static/
>> │   └── style.css
>> ├── templates/
>> │   └── index.html
>>
```

1.2.2 GitHub Actions

- A **YAML workflow file** is used to automate stages like install, build, and deployment.
- The pipeline is **triggered** by events like push or pull_request.
- Common GitHub Actions used:
    - actions/checkout@v2 – Checks out the repository
    - actions/setup-python@v4 – Sets up a Python environment
    - Custom shell commands to:
        - Install dependencies using pip
        - Run Flask app
        - Build Docker image and optionally push it to a registry

1.2.3 Flask Travel App Codebase

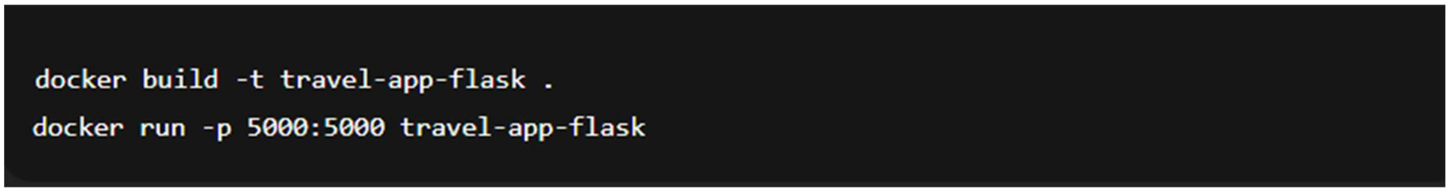
- The application is built using **Flask (Python)** with a simple web interface.
- Uses **Jinja templates** for HTML rendering.
- Folder structure includes:

  - app.py – main Flask app
  - templates/index.html – for user interface
  - static/style.css – for styling
  - requirements.txt – includes Flask, etc.
- Lightweight and easy to extend for adding features like bookings or destination search.

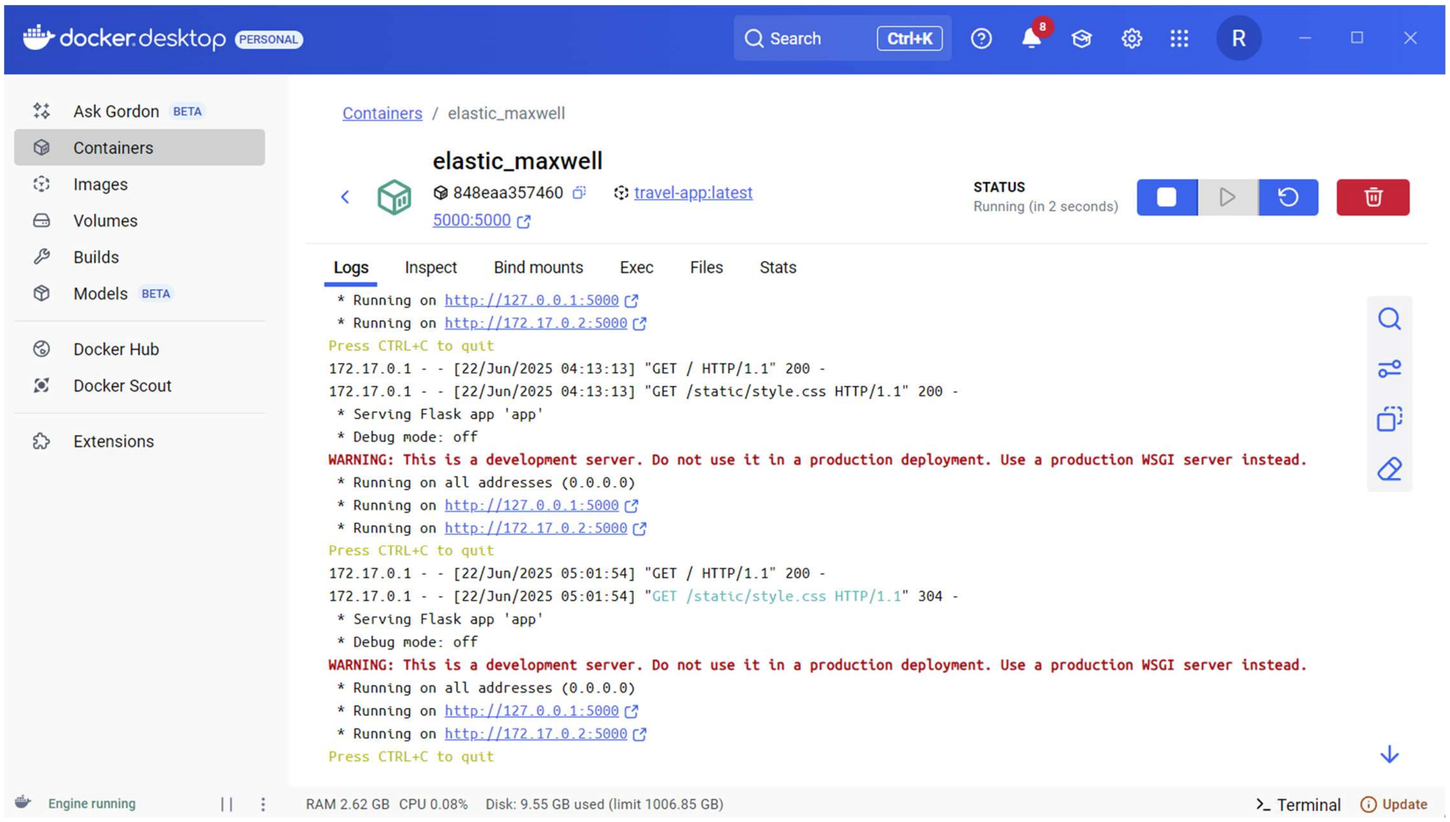
1.2.4 Docker

- **Docker** is used to ensure consistent deployment across different environments.
- Key Docker components include:
  - Dockerfile – defines how to build the container image
  - Optional .dockerignore – to exclude files from Docker context (e.g., .git, __pycache__)

Docker commands used in the pipeline:



```
docker build -t travel-app-flask .
docker run -p 5000:5000 travel-app-flask
```



```
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [22/Jun/2025 04:13:13] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [22/Jun/2025 04:13:13] "GET /static/style.css HTTP/1.1" 200 -
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [22/Jun/2025 05:01:54] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [22/Jun/2025 05:01:54] "GET /static/style.css HTTP/1.1" 304 -
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

1.3 Prerequisites

1.3.1 Technical Knowledge

- **Git & GitHub:**
  For version control, pushing code to GitHub, and enabling collaboration.
- **Flask (Python):**
  Understanding of basic Python and how Flask handles routing, templating, and HTTP requests.
- **YAML:**
  To configure CI/CD workflows using GitHub Actions.
- **Docker:**
  Basic knowledge of Docker concepts like containers, images, and how to write a Dockerfile.

1.3.2 Tooling Setup

- Git installed on your local machine

- Visual Studio Code (VS Code) or any other code editor
- Docker Desktop for building and testing Docker containers locally
- GitHub Account to host your project and use GitHub Actions for CI/CD
- Fully functional React frontend

CHAPTER 2
METHODOLGY

This project follows a structured DevOps methodology to automate the build, testing, containerization, and deployment of a Flask-based Travel App. The aim is to ensure consistent delivery, improved developer productivity, and reliable deployment using Docker and GitHub Actions.

1. Application Development

- The Flask web framework is used to develop the Travel App backend.
- Static files (HTML, CSS) and templates are organized within Flask's directory structure (templates/, static/).
- The main application logic is handled in app.py.

2. Version Control with Git & GitHub

- The entire project is tracked using Git.
- A GitHub repository is created to host the source code and manage collaborative development.
- All changes are committed with meaningful messages and pushed to the remote repository.

3. CI/CD Pipeline with GitHub Actions

- A CI/CD pipeline is configured using GitHub Actions to automate testing and deployment tasks.
- A YAML workflow file (.github/workflows/ci.yml) is created to define the CI/CD process.
- The pipeline runs automatically on code pushes or pull requests and performs:
- o Python environment setup
- o Dependency installation (pip install)

- Code linting (optional)
- Docker image build
- Optional deployment steps (e.g., push to Docker Hub or run container)

4. Dockerization

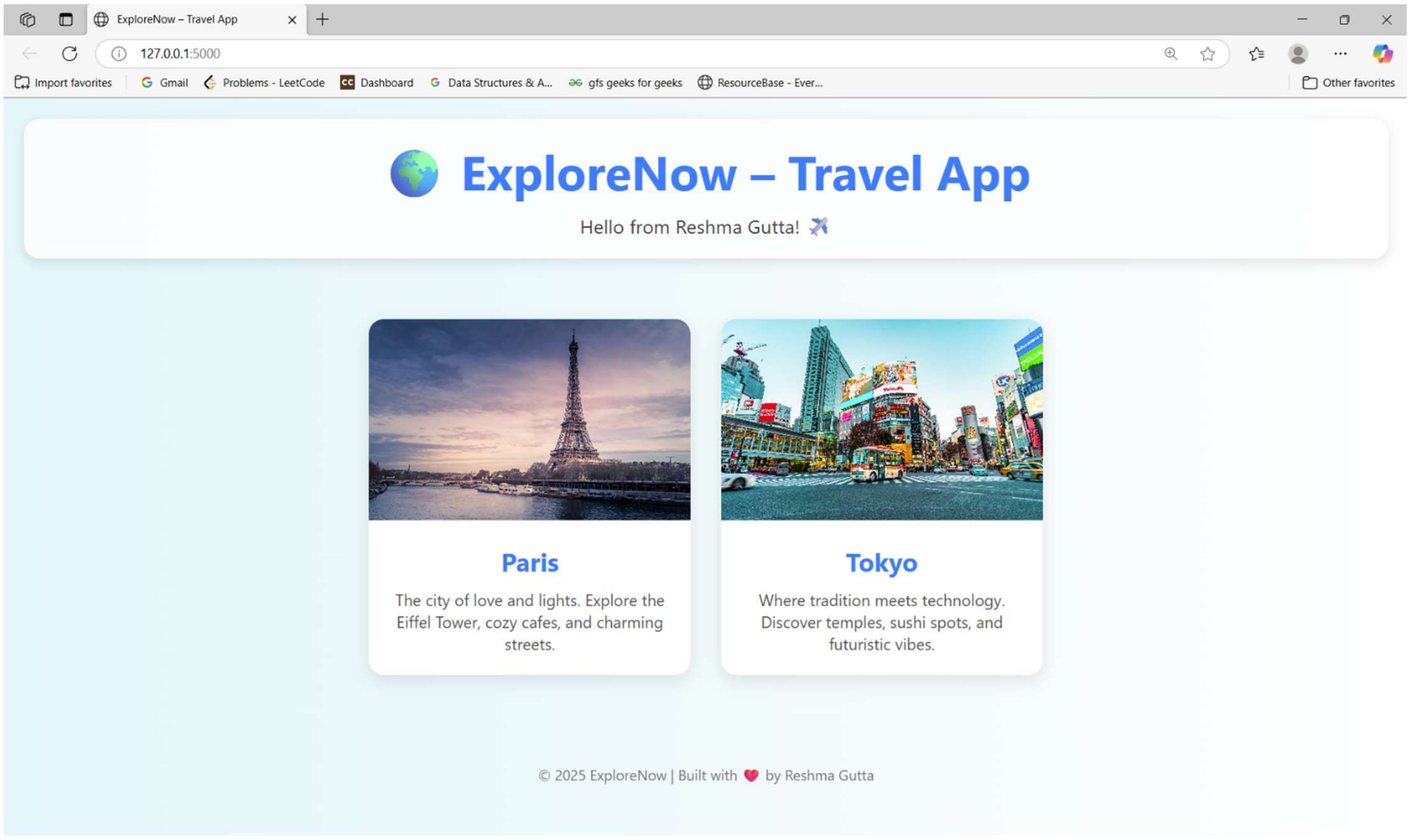
- A Dockerfile is written to containerize the Flask app, ensuring consistent environments across local and production setups.
- The Docker image includes:
- Python base image
- Copied source code
- Installed dependencies
- CMD to start the Flask server

5. Deployment

- The final Docker image is tested locally using docker run.
- (Optional) The image can be:
- Pushed to Docker Hub
- Deployed to cloud servers like AWS EC2, Heroku, or Render
- Automated for deployment through GitHub Actions

CHAPTER 3
RESULTS AND DISCUSSION

Execution:



ExploreNow – Travel App

Hello from Reshma Gutta! ✈️

Paris

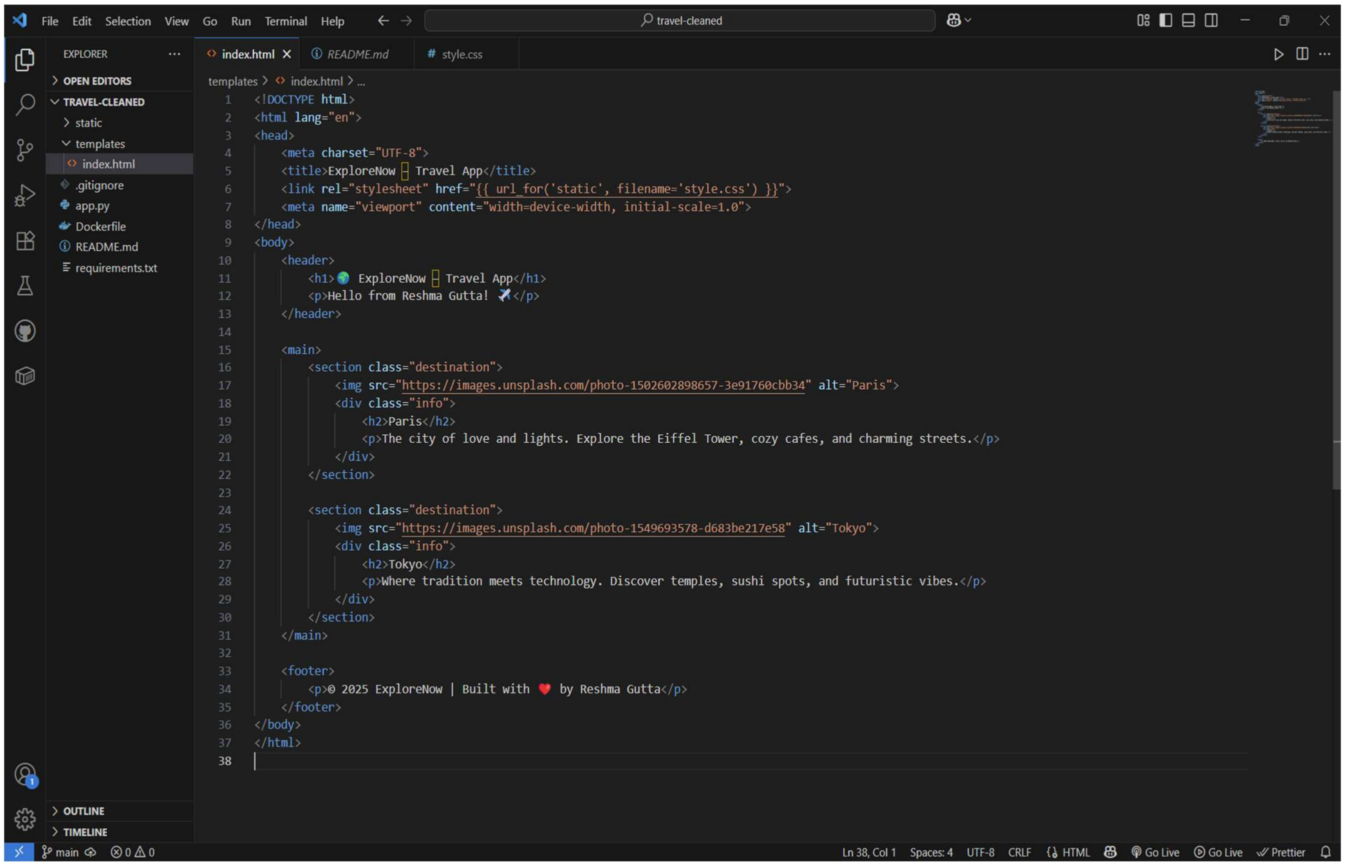
The city of love and lights. Explore the Eiffel Tower, cozy cafes, and charming streets.

Tokyo

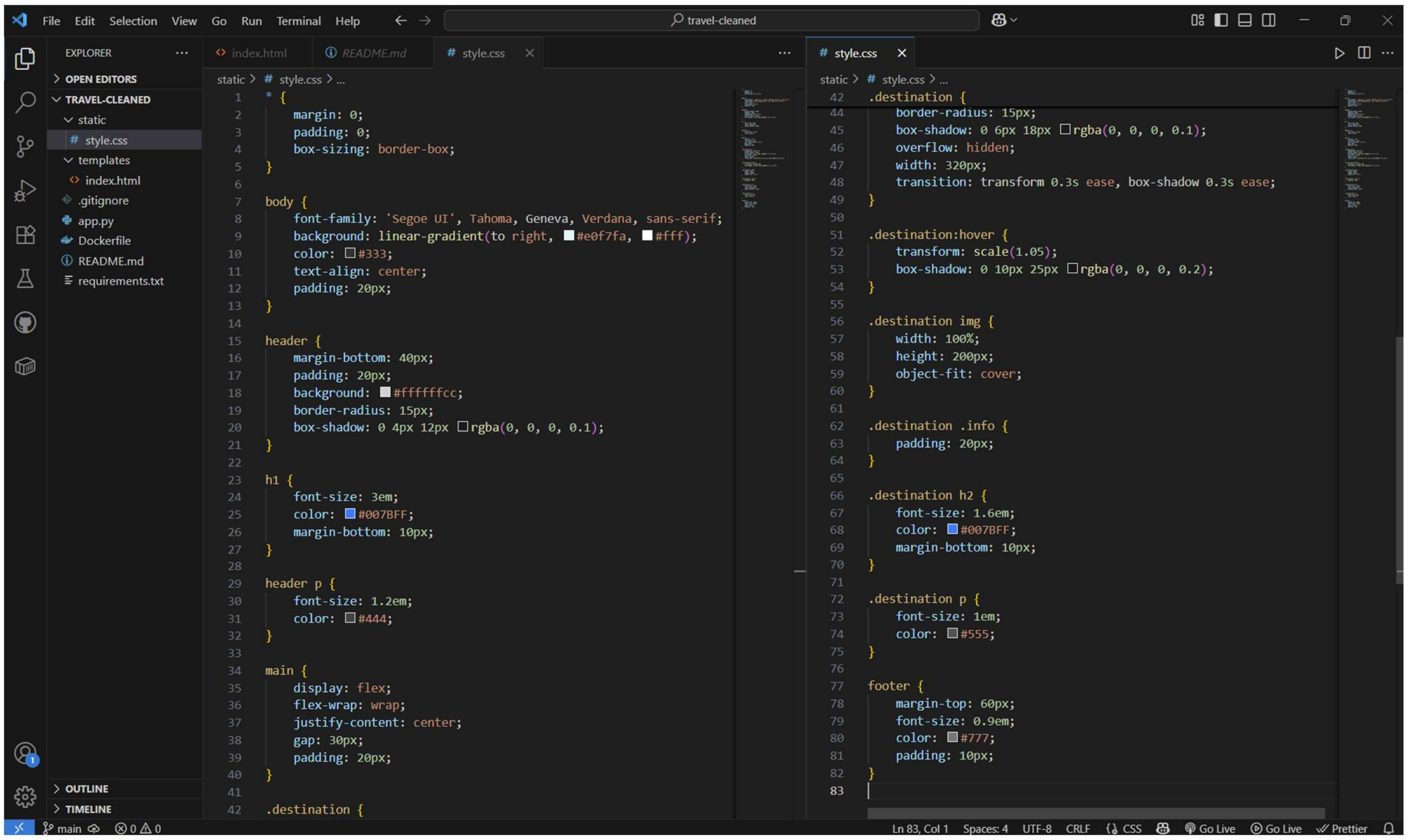
Where tradition meets technology. Discover temples, sushi spots, and futuristic vibes.

© 2025 ExploreNow | Built with ❤️ by Reshma Gutta

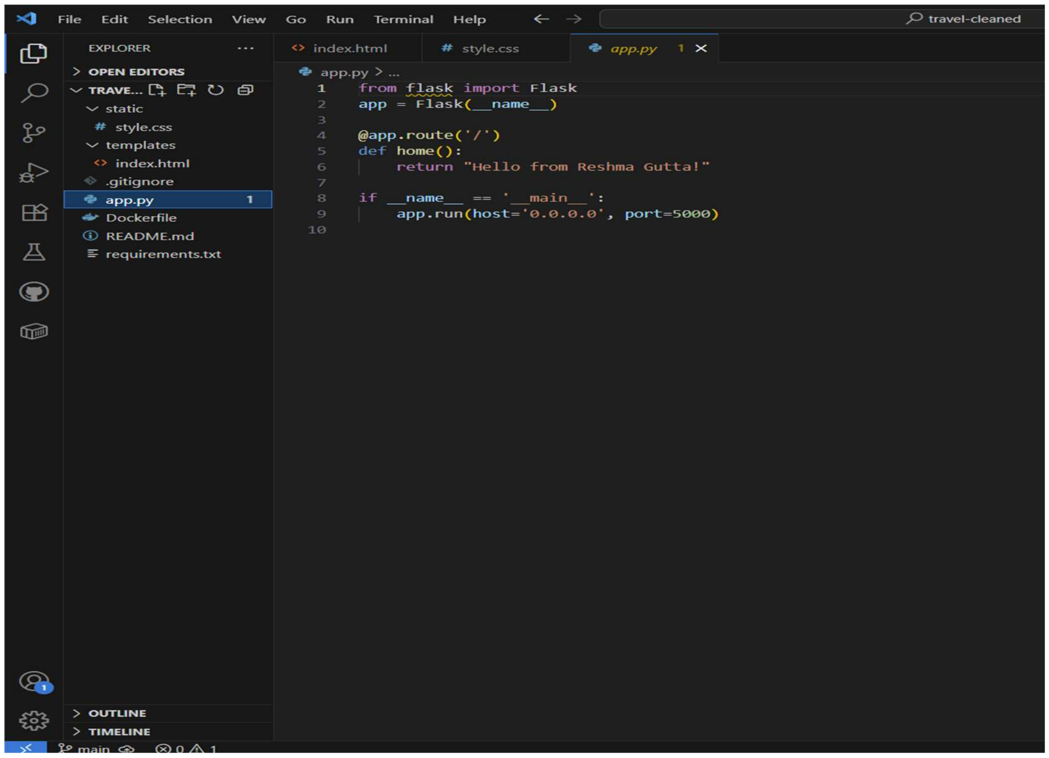
Source Code:



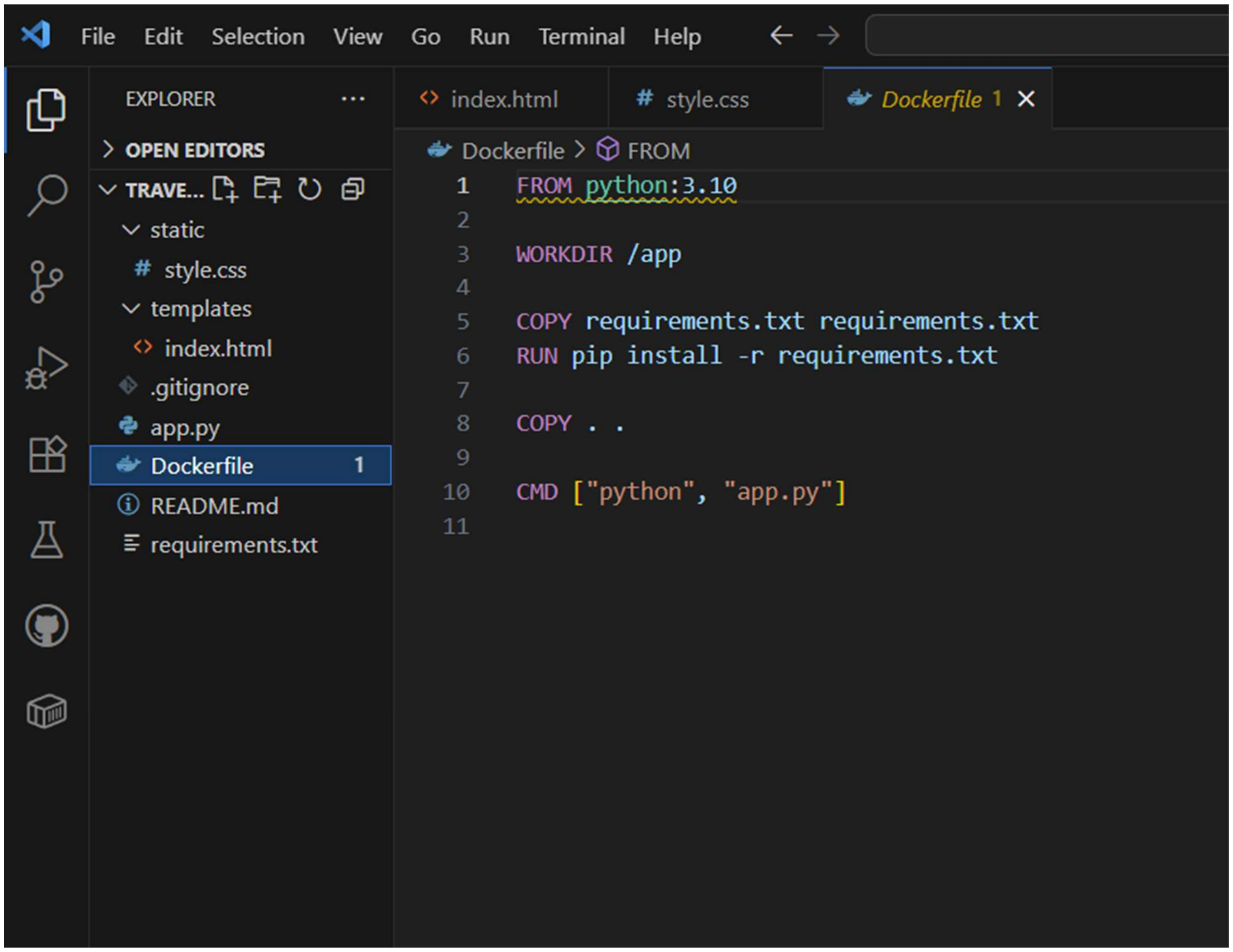
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ExploreNow ▯ Travel App</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <header>
        <h1>🌍 ExploreNow ▯ Travel App</h1>
        <p>Hello from Reshma Gutta! ✈️</p>
    </header>

    <main>
        <section class="destination">
            <img src="https://images.unsplash.com/photo-1502602898657-3e91760cbb34" alt="Paris">
            <div class="info">
                <h2>Paris</h2>
                <p>The city of love and lights. Explore the Eiffel Tower, cozy cafes, and charming streets.</p>
            </div>
        </section>

        <section class="destination">
            <img src="https://images.unsplash.com/photo-1549693578-d683be217e58" alt="Tokyo">
            <div class="info">
                <h2>Tokyo</h2>
                <p>Where tradition meets technology. Discover temples, sushi spots, and futuristic vibes.</p>
            </div>
        </section>
    </main>

    <footer>
        <p>© 2025 ExploreNow | Built with ❤️ by Reshma Gutta</p>
    </footer>
</body>
</html>

```



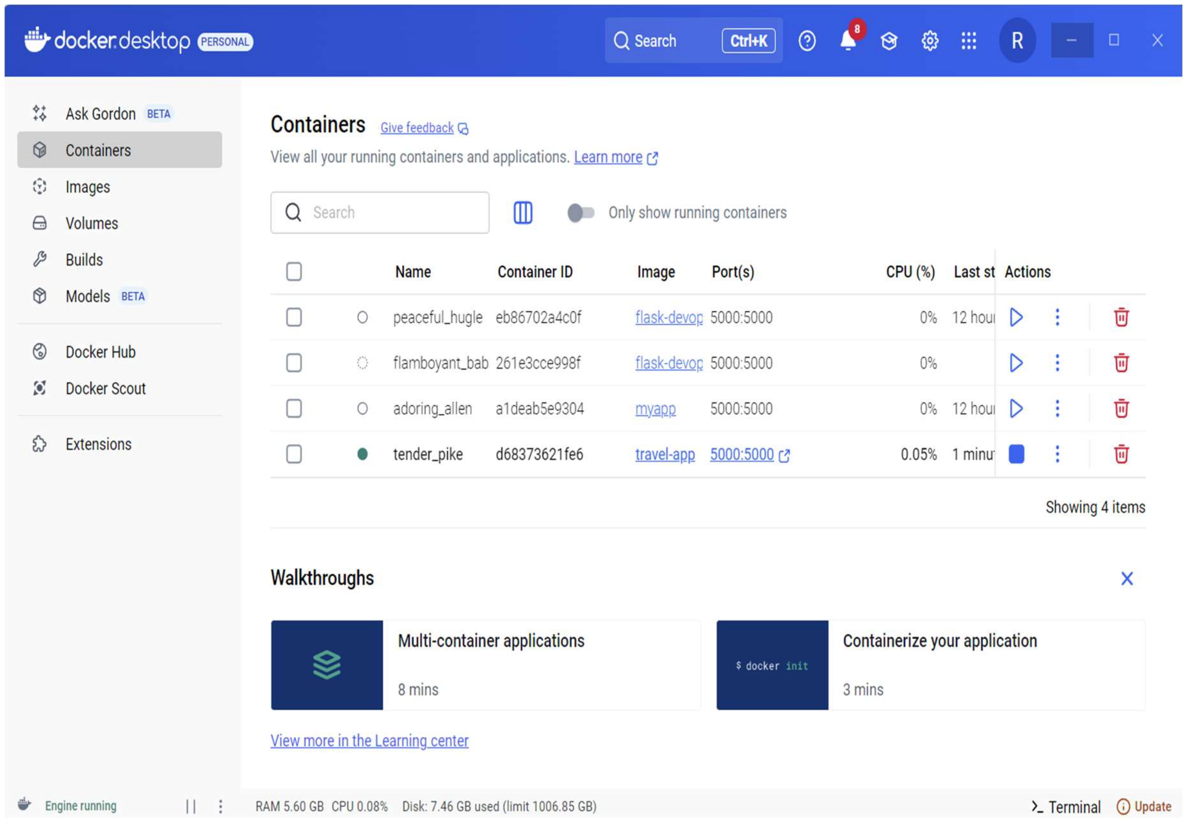
```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(to right, #e0f7fa, #fff);
    color: #333;
    text-align: center;
    padding: 20px;
}

header {
    margin-bottom: 40px;
    padding: 20px;
    background: #ffffffcc;
    border-radius: 15px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}

h1 {
    font-size: 3em;
    color: #007BFF;
    margin-bottom: 10px;
}

header p {
    font-size: 1.2em;
    color: #444;
}

main {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 30px;
    padding: 20px;
}

.destination {
    border-radius: 15px;
    box-shadow: 0 6px 18px rgba(0, 0, 0, 0.1);
    overflow: hidden;
    width: 320px;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.destination:hover {
    transform: scale(1.05);
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);
}

.destination img {
    width: 100%;
    height: 200px;
    object-fit: cover;
}

.destination .info {
    padding: 20px;
}

.destination h2 {
    font-size: 1.6em;
    color: #007BFF;
    margin-bottom: 10px;
}

.destination p {
    font-size: 1em;
    color: #555;
}

footer {
    margin-top: 60px;
    font-size: 0.9em;
    color: #777;
    padding: 10px;
}
```



```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello from Reshma Gutta!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```



```dockerfile
FROM python:3.10

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]

```



Containers Give feedback

View all your running containers and applications. Learn more

Only show running containers

	Name	Container ID	Image	Port(s)	CPU (%)	Last st	Actions
○	peaceful_hugle	eb86702a4c0f	flask-devop	5000:5000	0%	12 hou	
○	flamboyant_bab	261e3cce998f	flask-devop	5000:5000	0%		
○	adoring_allen	a1deab5e9304	myapp	5000:5000	0%	12 hou	
●	tender_pike	d68373621fe6	travel-app	5000:5000	0.05%	1 minu	

Showing 4 items

Walkthroughs

Multi-container applications 8 mins

Containerize your application 3 mins

View more in the Learning center

RAM 5.60 GB CPU 0.08% Disk: 7.46 GB used (limit 1006.85 GB)

CHAPTER 4
CONCLUSION

This project demonstrates how DevOps principles can be effectively applied to a real-world **Flask-based Travel Application** using **Dockerization** and **CI/CD automation**. By leveraging **GitHub Actions**, we automated the entire software delivery pipeline—from pushing code to building and deploying the application—reducing manual effort and improving the speed of releases. Docker played a key role in providing **environmental consistency**, ensuring that the app runs reliably across development, testing, and production environments.

The combination of **containerization** and **CI/CD** not only simplified deployment but also made the application more **scalable**, **portable**, and **maintainable**. This approach follows the core DevOps principles of **automation**, **continuous delivery**, and **operational efficiency**.

This project serves as a solid foundation for future enhancements such as integrating cloud platforms, expanding to microservices, or adopting Infrastructure as Code (IaC) tools. It reflects a complete and modern DevOps workflow for Flask applications.