CENG 495 - Cloud Computing
Spring 2024
Homework 2

Adıgüzel, Gürhan İlhan
e244802@metu.edu.tr

May 12, 2024

1 Introduction

This report documents the deployment process of the Expense Splitter Website developed with Docker containers and Docker Compose. In this report, decisions during deployment, a detailed overview of the deployment workflow, challenges encountered and solutions, deployment steps, system architecture and screenshots of the deployed application are provided.

2 Decisions Made During Deployment

- **Front-end Technology**: I chose Flutter for front-end development due to have easy UI design to want to use clean-architecture principles and also my past experience and familiarity with the language.
- **Back-end Framework**: I selected Flask a lightweight web framework written in Python, for back-end development due to its simplicity.
- **Database Solution**: I chose MongoDB as the database solution for its support for handling unstructured data, providing efficient data management for the application and also because I am experienced in using it.

3 Challenges Encountered

- **Creating Local MongoDB Container:** Initially, I had difficulty in setting up a local MongoDB container. To handle this, I installed the local MongoDB Docker image and configured the port to 27017.
- **Building Dockerfiles for Flask Microservices:** I had the wrong address request issue when creating Dockerfiles for Flask microservices and trying to communicate with the MongoDB container. To overcome this, I configured the Flask application to connect to the MongoDB container using "host.docker.internal" instead of "localhost" to ensure communication between services.
- **Dockerizing the Flutter Web App:** While dockerizing the Flutter web app and I faced a compatibility issues. To resolve this, I authored a Dockerfile within the Flutter project, enabling the building and running of the 'flutter build web' within a container.
- **CORS Errors Between Frontend and Backend:** There were Cross-Origin Resource Sharing (CORS) errors between the frontend and backend components. To handle this issue, I modified the backend files to allow CORS from all origins, ensuring unrestricted communication between frontend and backend.
- **Composing Different Containers:** To simplify the orchestration process, I carefully defined dependencies between containers in the 'docker-compose.yml' file. This ensured smooth communication and cooperation among services.

4 Deployment Workflow Using Docker and Docker Compose

4.1 Containerization with Docker

1. **Backend Microservices**: Flask microservices were containerized using Docker and Dockerfiles were created for each microservice, and Docker images were built and run as Docker containers.

2. **Database Setup**: MongoDB was deployed locally as a Docker image using the following commands:

   ```
   docker pull mongodb/mongodb-community-server:latest
   docker run --name mongodb -p 27017:27017 -d mongodb/mongodb-community-server:latest
   ```

   These commands pulled the latest MongoDB Docker image and created a container named ”mongodb” with port 27017 exposed for access.

3. **Frontend Development**: The Flutter web app was containerized using Docker, with a Dockerfile defining the build environment and dependencies. A Docker container was instantiated to host the Flutter web app.

4.2 Orchestration with Docker Compose

1. **Compose File Creation**: A Docker Compose file (`docker-compose.yml`) was created to define services, networks, and volumes for the application. Configuration for MongoDB, Flask microservices, and the Flutter web app was specified in the Compose file.

2. **Service Orchestration**: Docker Compose was used to orchestrate the deployment process. The `docker-compose up` command initiated the deployment, creating containers, networks, and volumes as per the specified configuration.

5 System Architecture Overview

The system architecture consists of frontend, backend, and database components encapsulated within Docker containers. Interaction between components is facilitated through Docker networks.

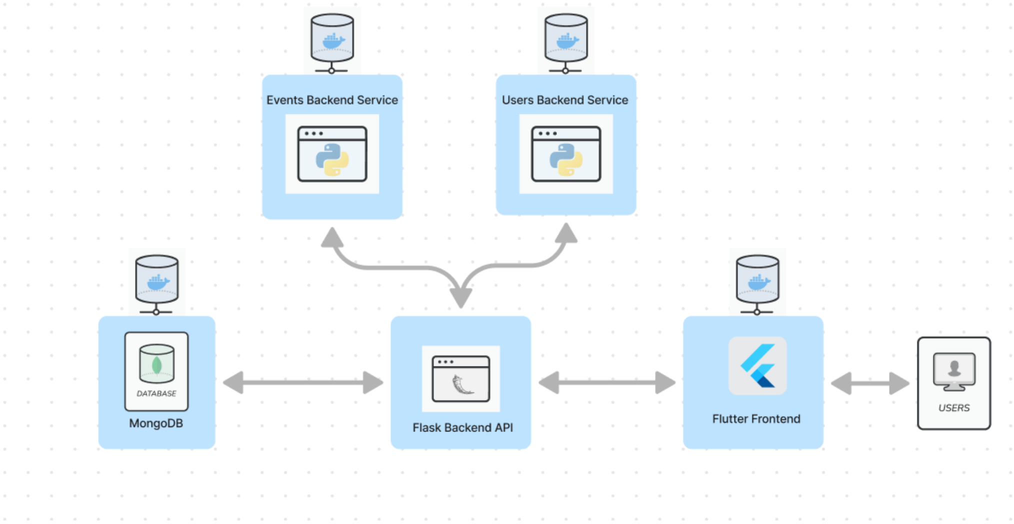
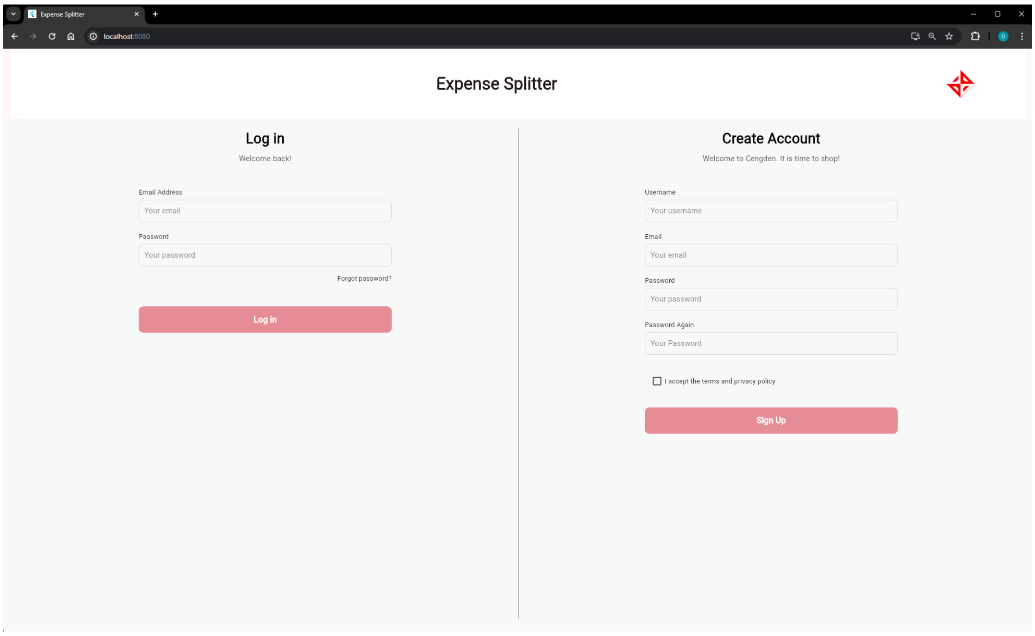


Figure 1: System Design Diagram

6 Screenshots



Expense Splitter

Log in

Welcome back!

Email Address

Your email

Password

Your password

Forgot password?

Create Account

Welcome to Cengden. It is time to shop!

Username

Your username

Email

Your email

Password

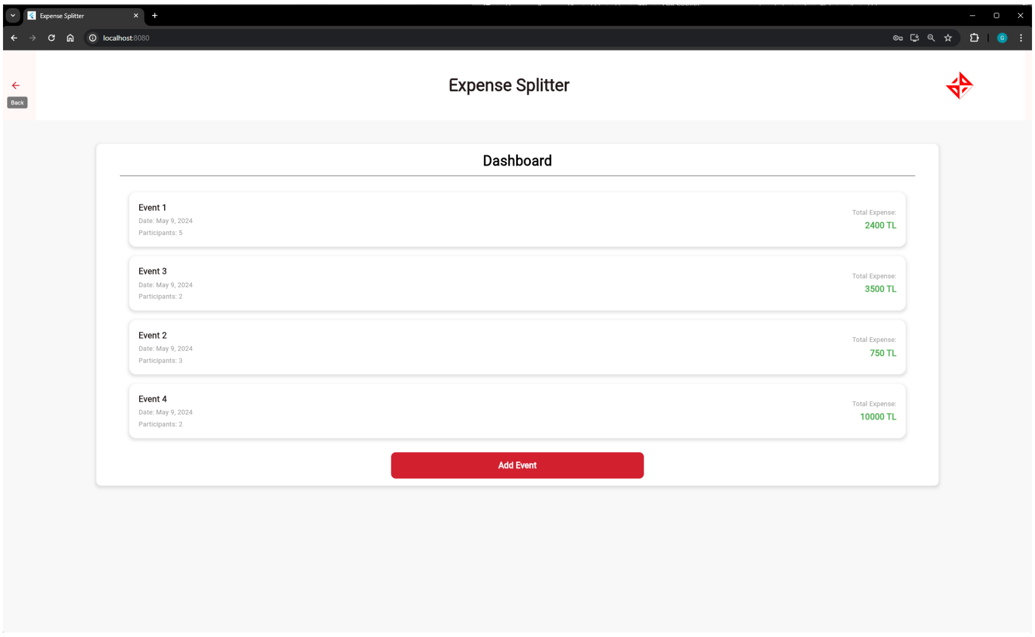
Your password

Password Again

Your Password

□ I accept the terms and privacy policy

Figure 2: Register Page



Expense Splitter

Dashboard

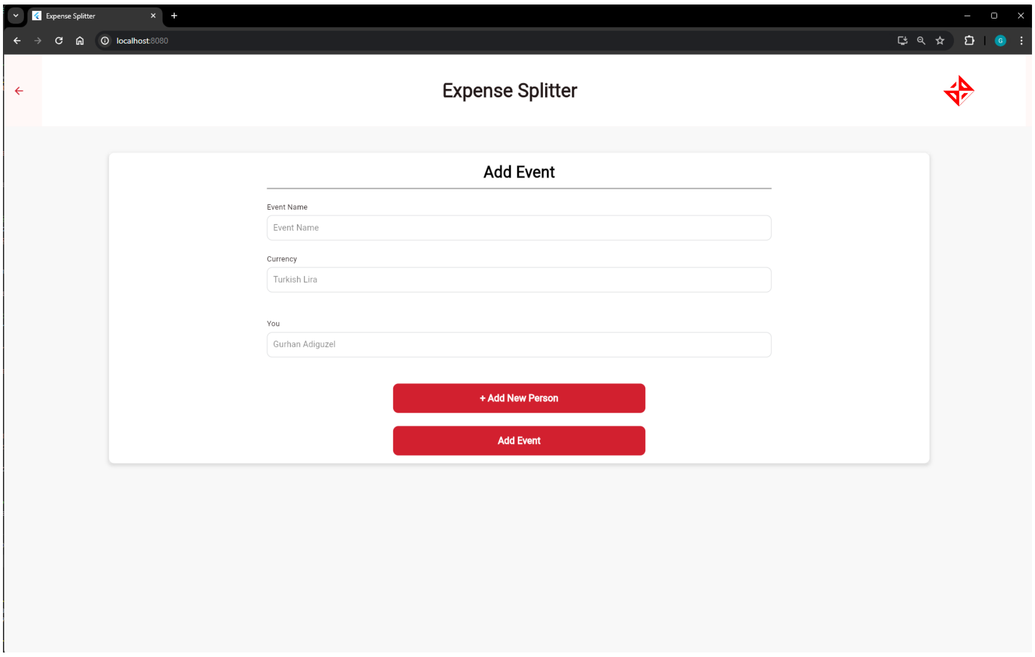
Event 1
Date: May 9, 2024
Participants: 5
Total Expense:
2400 TL

Event 3
Date: May 9, 2024
Participants: 2
Total Expense:
3500 TL

Event 2
Date: May 9, 2024
Participants: 3
Total Expense:
750 TL

Event 4
Date: May 9, 2024
Participants: 2
Total Expense:
10000 TL

Figure 3: Dashboard Page



Expense Splitter

Add Event

Event Name

Event Name

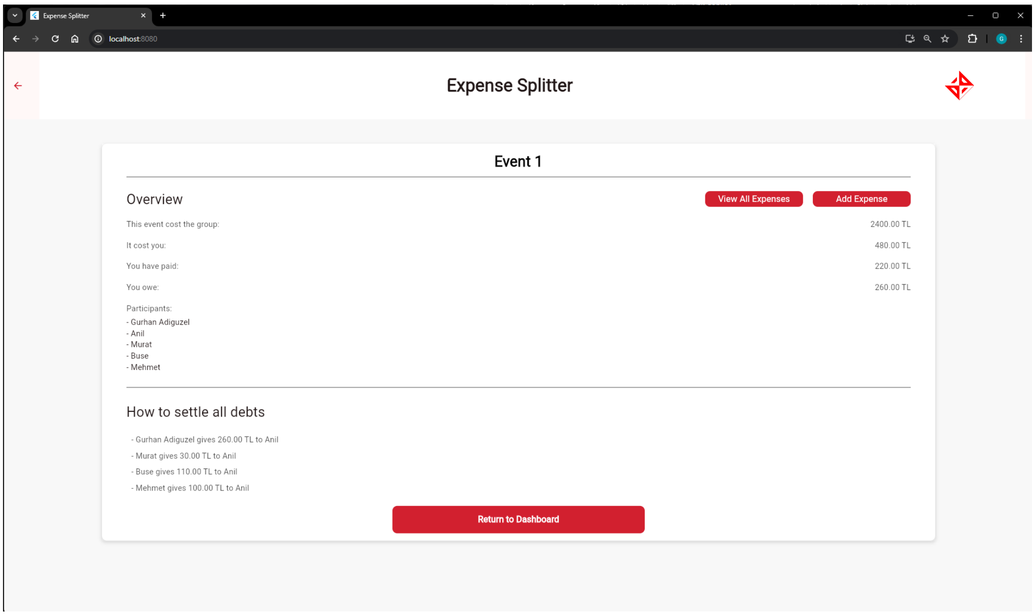
Currency

Turkish Lira

You

Gurhan Adiguzel

Figure 4: Add Event Page



Expense Splitter

Event 1

Overview

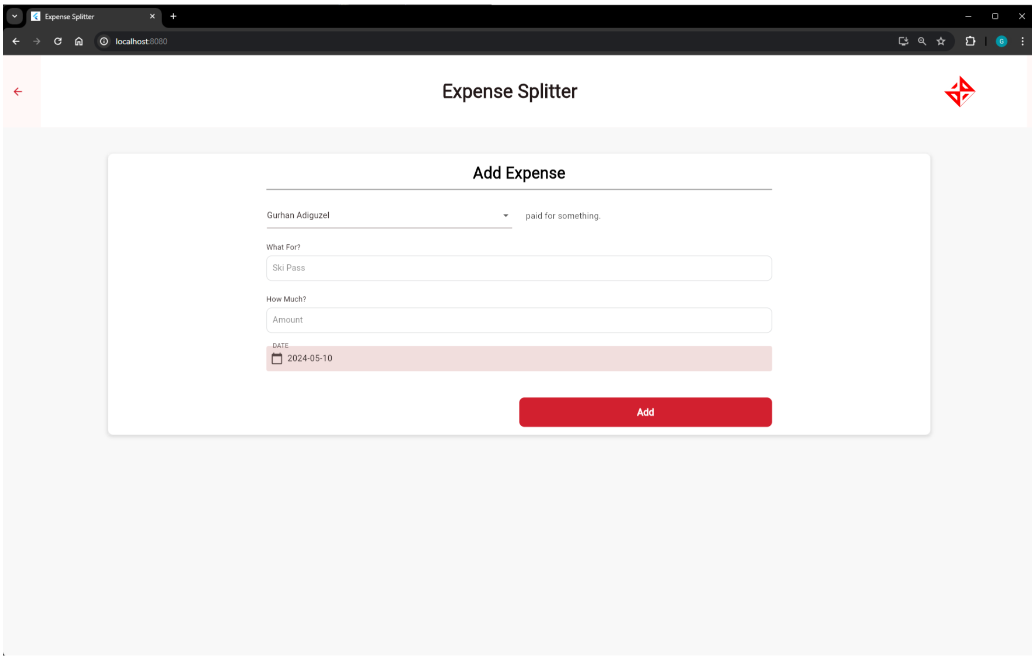
This event cost the group:	2400.00 TL
It cost you:	480.00 TL
You have paid:	220.00 TL
You owe:	260.00 TL

Participants:
- Gurhan Adiguzel
- Anil
- Murat
- Buse
- Mehmet

How to settle all debts

- Gurhan Adiguzel gives 260.00 TL to Anil
- Murat gives 30.00 TL to Anil
- Buse gives 110.00 TL to Anil
- Mehmet gives 100.00 TL to Anil

Figure 5: Event Details Page



Expense Splitter

Add Expense

Gurhan Adiguzel ▾ paid for something.

What For?

Ski Pass

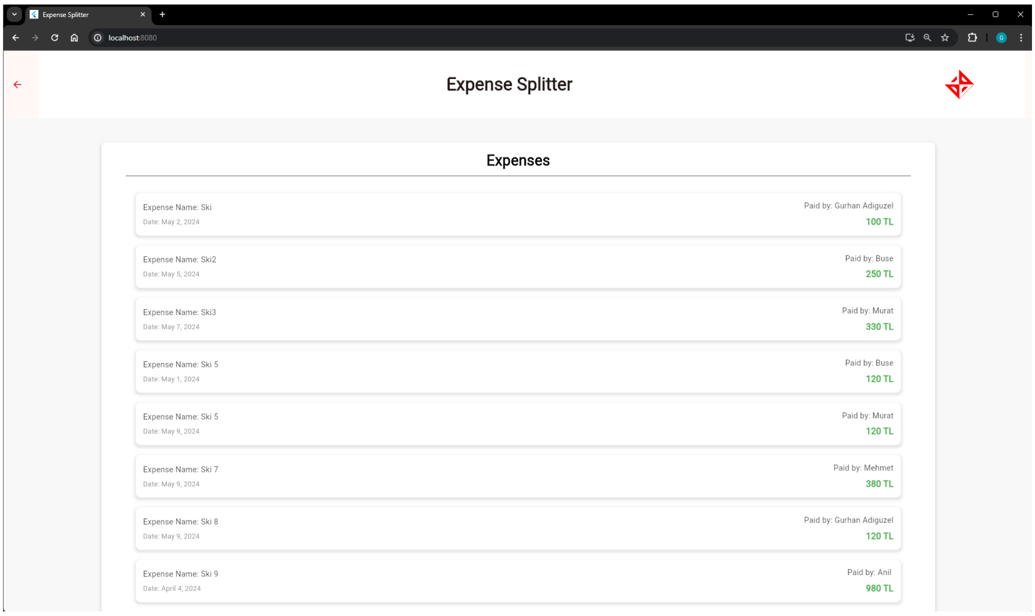
How Much?

Amount

DATE

2024-05-10

Figure 6: Add Expense Page



Expense Splitter

Expenses

Expense Name: Ski
Date: May 2, 2024
Paid by: Gurhan Adiguzel
100 TL

Expense Name: Ski2
Date: May 5, 2024
Paid by: Buse
250 TL

Expense Name: Ski3
Date: May 7, 2024
Paid by: Murat
330 TL

Expense Name: Ski 5
Date: May 1, 2024
Paid by: Buse
120 TL

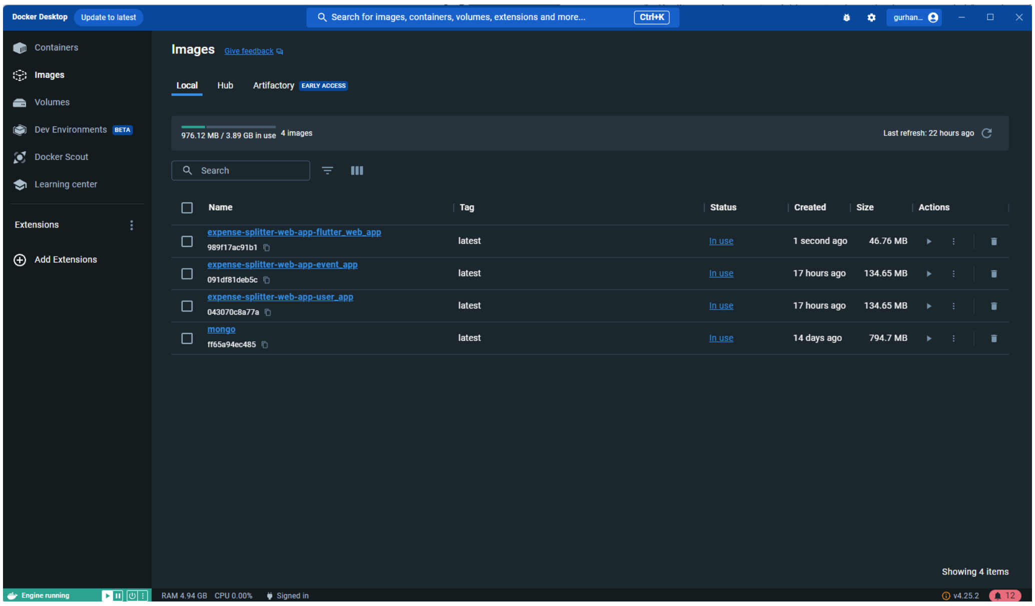
Expense Name: Ski 5
Date: May 9, 2024
Paid by: Murat
120 TL

Expense Name: Ski 7
Date: May 9, 2024
Paid by: Mehmet
380 TL

Expense Name: Ski 8
Date: May 9, 2024
Paid by: Gurhan Adiguzel
120 TL

Expense Name: Ski 9
Date: April 4, 2024
Paid by: Anil
980 TL

Figure 7: Expenses Page



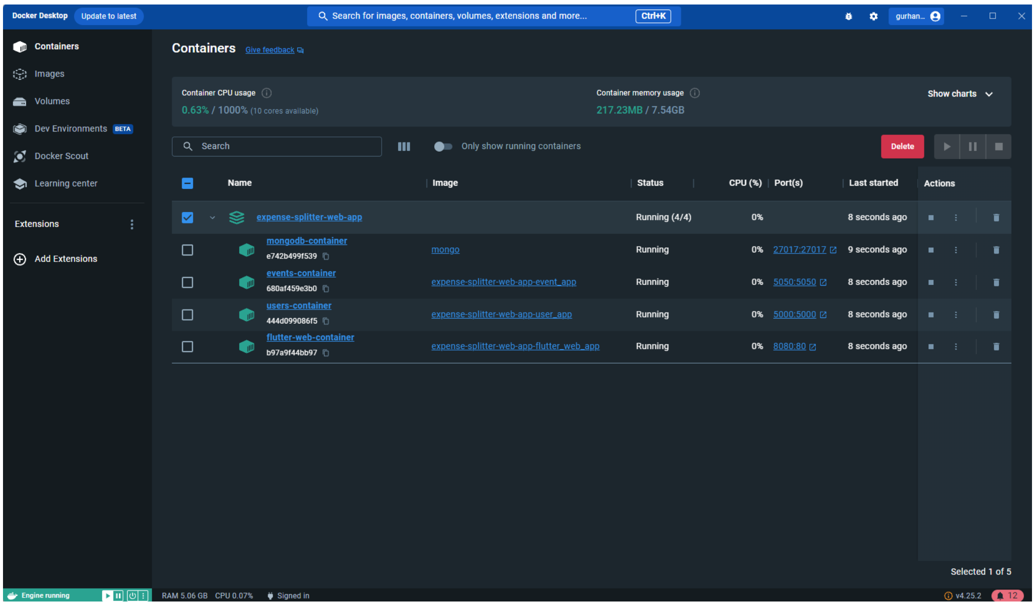
976.12 MB / 3.89 GB in use 4 images

Last refresh: 22 hours ago

□	Name	Tag	Status	Created	Size	Actions
□	expense-splitter-web-app-flutter_web_app 989f17ac91b1	latest	In use	1 second ago	46.76 MB	
□	expense-splitter-web-app-event_app 091df81deb5c	latest	In use	17 hours ago	134.65 MB	
□	expense-splitter-web-app-user_app 043070c8a77a	latest	In use	17 hours ago	134.65 MB	
□	mongo ff65a94ec485	latest	In use	14 days ago	794.7 MB	

Showing 4 items

Figure 8: Docker Images



Container CPU usage
0.63% / 1000% (10 cores available)

Container memory usage
217.23MB / 7.54GB

☐	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
☑	expense-splitter-web-app		Running (4/4)	0%		8 seconds ago	
□	mongodb-container e742b499f539	mongo	Running	0%	27017:27017	9 seconds ago	
□	events-container 680af459e3b0	expense-splitter-web-app-event_app	Running	0%	5050:5050	8 seconds ago	
□	users-container 444d099086f5	expense-splitter-web-app-user_app	Running	0%	5000:5000	8 seconds ago	
□	flutter-web-container b97a9f44bb97	expense-splitter-web-app-flutter_web_app	Running	0%	8080:80	8 seconds ago	

Selected 1 of 5

Figure 9: Docker Containers

7 Conclusion

In conclusion, the deployment of the Expense Splitter web application using Docker and Docker Compose has been successfully accomplished. By executing the command ‘docker-compose up‘, all services can be initiated, enabling the application to be accessible and fully operational within a containerized environment.