MID EXAM PROJECT REPORT

Project Title	**:**	**Currency Converter Application**
Subject	**:**	**Distributed and Parallel System**
Class	**:**	**Informatics 3**

By:
Bagus Eka Bagaskara (001202300164)

INFORMATICS STUDY PROGRAM
FACULTY OF COMPUTER SCIENCE
PRESIDENT UNIVERSITY

CHAPTER I
INTRODUCTION

1.1. Background

The "it works on my machine" issue, in which programs don't function on various computers because of differences in operating systems, software versions, and dependencies, is a prevalent problem in contemporary software development. This project uses Docker to implement a containerization technique, which directly solves this problem. The main objective is to ensure that a full-stack web application operates consistently across all platforms by establishing a standardized, reproducible, and isolated environment.

1.2. Application Description

The program created is called "Currency Converter," a dynamic online utility that enables users to convert currencies in real time. Users can choose a base currency, a target currency, and an amount to convert using the application's user-friendly interface. The frontend receives the computed result from the backend service, which retrieves the most recent exchange rates from a public API. The project also incorporates extra elements, such as a page with fascinating financial information and a basic interactive game, to increase user engagement.

1.3. Project Objectives

- To successfully build a full-stack web application with a separate frontend and backend service.
- To containerize both the frontend and backend services using individual `Dockerfile`s.
- To create a seamless deployment process by orchestrating the containers using a `docker-compose.yml` file.
- To demonstrate a clear understanding of how Docker solves environmental inconsistency problems in software development.

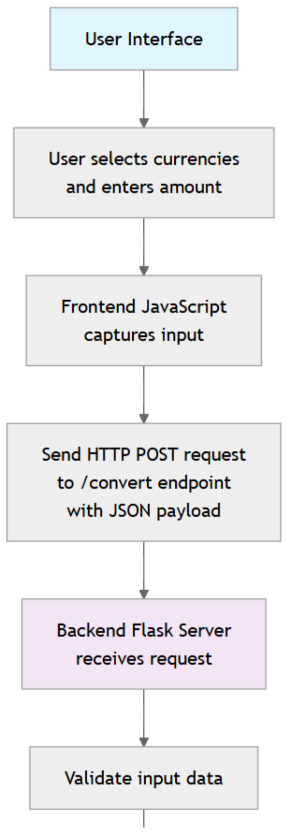
CHAPTER II
APPLICATION ARCHITECTURE AND DESIGN

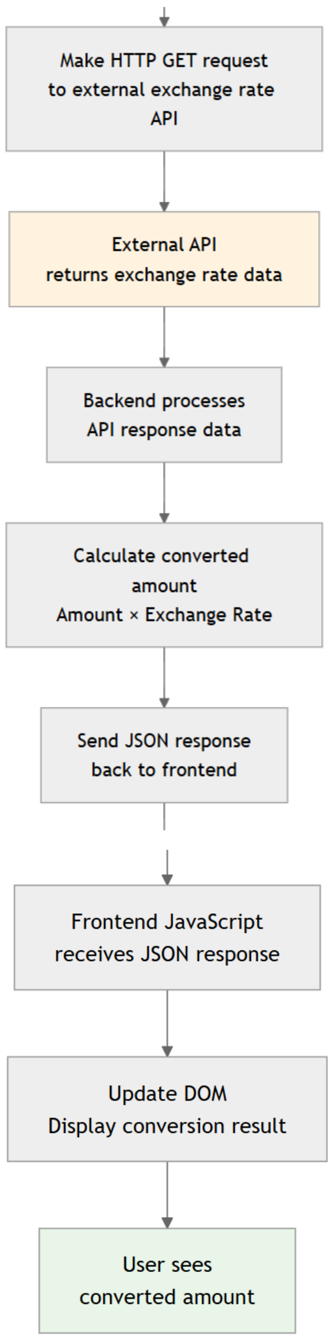
2.1. Application Architecture

This application employs a decoupled Client-Server architecture.

- **Client (Frontend):** A static web interface built with HTML, CSS, and JavaScript that runs in the user's web browser. It is responsible for rendering the user interface and handling user interactions. It is served by a lightweight Nginx web server.
- **Server (Backend):** A RESTful API built with Python and the Flask framework. It is responsible for all business logic, including fetching data from the external exchange rate API and performing the conversion calculations. It runs on a production-ready Gunicorn WSGI server.

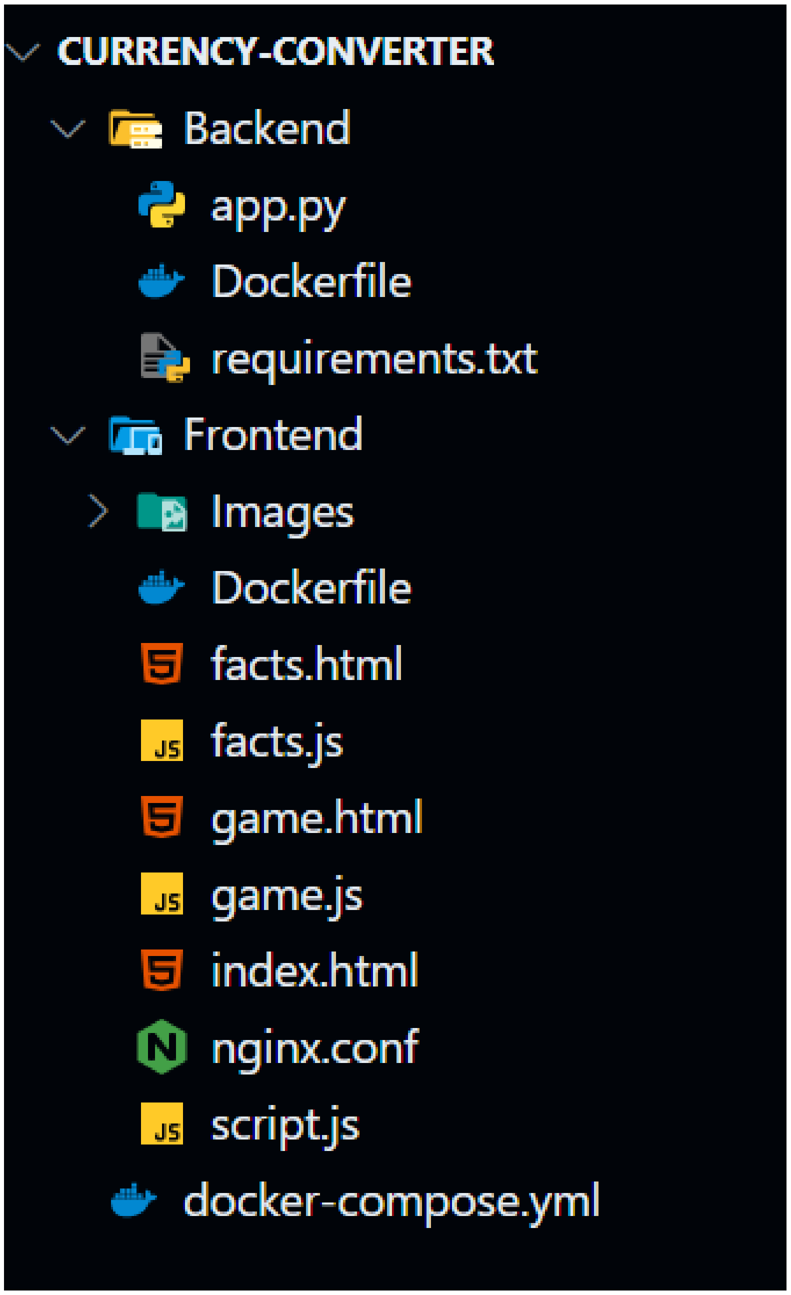
Data Flow Currency Converter:


User Interface
User selects currencies and enters amount
Frontend JavaScript captures input
Send HTTP POST request to /convert endpoint with JSON payload
Backend Flask Server receives request
Validate input data


Make HTTP GET request to external exchange rate API
External API returns exchange rate data
Backend processes API response data
Calculate converted amount Amount × Exchange Rate
Send JSON response back to frontend
Frontend JavaScript receives JSON response
Update DOM Display conversion result
User sees converted amount

2.2. Project Structure

The project is organized into distinct Frontend and Backend directories to maintain a clean separation of concerns.

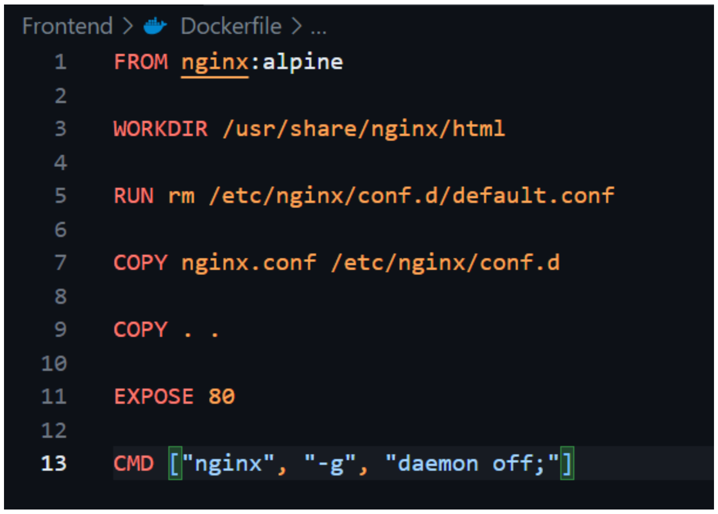


```
CURRENCY-CONVERTER
  Backend
    app.py
    Dockerfile
    requirements.txt
  Frontend
    > Images
    Dockerfile
    facts.html
    facts.js
    game.html
    game.js
    index.html
    nginx.conf
    script.js
  docker-compose.yml
```

CHAPTER III
DOCKER IMPLEMENTATION

3.1. Frontend Dockerfile

The frontend service is containerized to run on an Nginx server. This ensures that the web server environment is consistent.



```
FROM nginx:alpine

WORKDIR /usr/share/nginx/html

RUN rm /etc/nginx/conf.d/default.conf

COPY nginx.conf /etc/nginx/conf.d

COPY . .

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

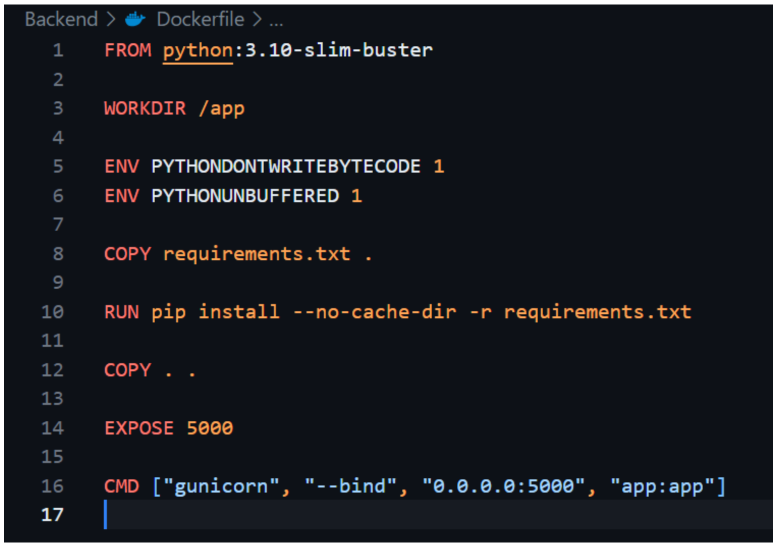
Line-by-Line Explanation:

- **FROM nginx:alpine**: This command initializes the build with a base image. nginx:alpine is chosen because it is an official, secure, and extremely lightweight version of Nginx, resulting in a smaller final image size.
- **WORKDIR /usr/share/nginx/html**: Sets the working directory inside the container. All subsequent commands like COPY will use this path as the destination. This is the default directory where Nginx looks for files to serve.
- **COPY nginx.conf /etc/nginx/conf.d/default.conf**: This copies our custom Nginx configuration file, overwriting the default one. This is crucial for setting up how Nginx handles requests.
- **COPY . .**: This command copies all files and folders from the local Frontend directory on the host machine into the working directory (/usr/share/nginx/html) inside the container.

- **EXPOSE 80**: This is a form of documentation. It informs Docker that the container listens on port 80 at runtime, which is the standard port for HTTP traffic.

3.2. Backend Dockerfile

The backend service is containerized to run Python in a controlled environment with all its dependencies.



```
FROM python:3.10-slim-buster

WORKDIR /app

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]

```

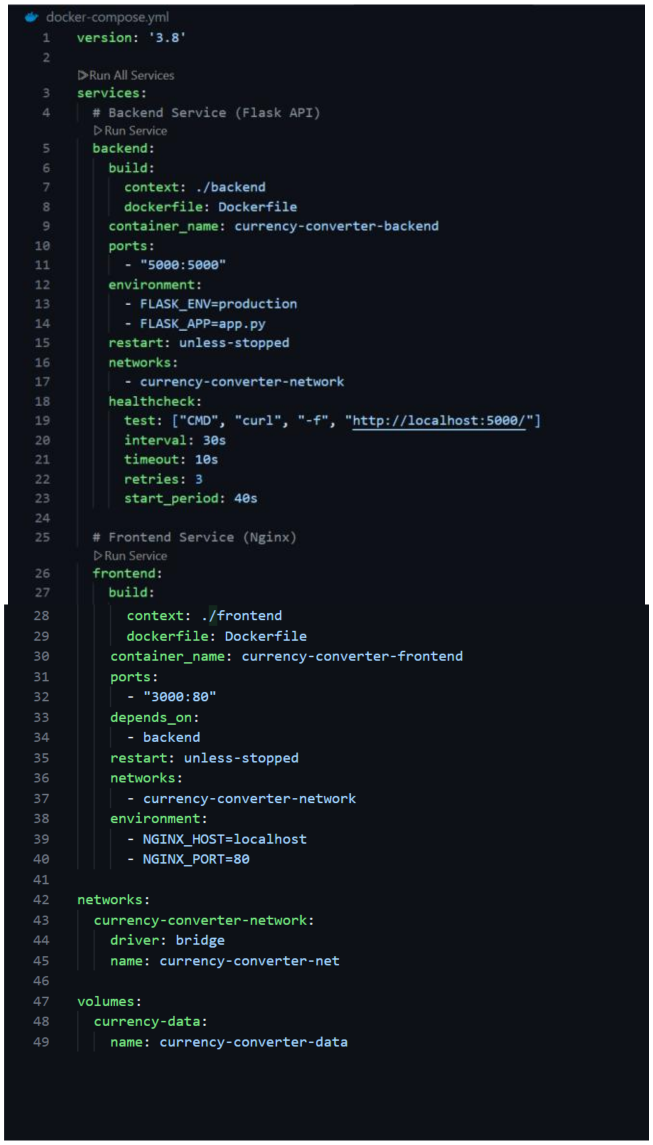
Line-by-Line Explanation:

- **FROM python:3.10-slim-buster**: We start with an official Python image. The slim-buster tag provides a minimal installation, which reduces the image size while still providing necessary tools.
- **WORKDIR /app**: Sets a clean working directory for our application inside the container.
- **COPY requirements.txt .**: We copy the requirements.txt file first. This is a key optimization technique. Docker's layer caching will only re-run the next step (pip install) if this specific file changes, making subsequent builds much faster.
- **RUN pip install --no-cache-dir -r requirements.txt**: This command installs all the Python libraries listed in requirements.txt. The --no-cache-dir flag prevents pip from storing a cache, which helps to keep the image size smaller.
- **COPY . .**: Copies the rest of the backend application code (like app.py) into the container.

- **CMD** ["gunicorn", **"--bind", "0.0.0.0:5000", "app:app"]**: This is the command that will be executed when the container starts. It uses gunicorn, a production-ready server, to run our Flask application (app:app means the app object in the app.py file). bind 0.0.0.0:5000 makes the server accessible from outside the container on port 5000.

3.3 Docker Compose

The docker-compose.yml file is the orchestrator that defines and runs our multi-container application.



```yaml
version: '3.8'

services:
  # Backend Service (Flask API)
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: currency-converter-backend
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - FLASK_APP=app.py
    restart: unless-stopped
    networks:
      - currency-converter-network
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  # Frontend Service (Nginx)
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: currency-converter-frontend
    ports:
      - "3000:80"
    depends_on:
      - backend
    restart: unless-stopped
    networks:
      - currency-converter-network
    environment:
      - NGINX_HOST=localhost
      - NGINX_PORT=80

networks:
  currency-converter-network:
    driver: bridge
    name: currency-converter-net

volumes:
  currency-data:
    name: currency-converter-data
```

Key Sections Explained:

- **services**: This is the main section where we define each container (service).
- **backend: & frontend:**: These are the custom names for our services.
- **build: context: ./backend**: This tells Docker Compose to build the image for this service using the Dockerfile found in the ./backend directory.
- **ports: - "3000:80"**: This maps a port on the host machine to a port inside the container. In this case, traffic to localhost:3000 on our computer will be forwarded to port 80 inside the frontend container.
- **networks:** - **currency-converter-network**: This attaches the service to a custom bridge network, allowing containers to communicate with each other using their service names (e.g., the frontend can reach the backend at http://backend:5000).
- **depends_on: - backend**: This ensures that the backend container is started before the frontend container.

CHAPTER IV
DEPLOYMENT PROCESS AND RESULTS

4.1 Deployment Steps

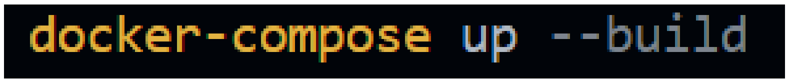
This application is designed for a simple, one-command deployment process.

Prerequisites:

1. **Docker Desktop:** Must be installed and running on the host machine.
2. **Project Files:** The complete project source code must be available locally.

Execution Steps:

1. **Clone Repository:** Download the project from its GitHub repository using git clone [URL].
2. **Navigate to Directory:** Open a terminal or command prompt and use the cd command to navigate into the root directory of the project (the folder containing docker-compose.yml).
3. **Run Docker Compose:** Execute the following command:

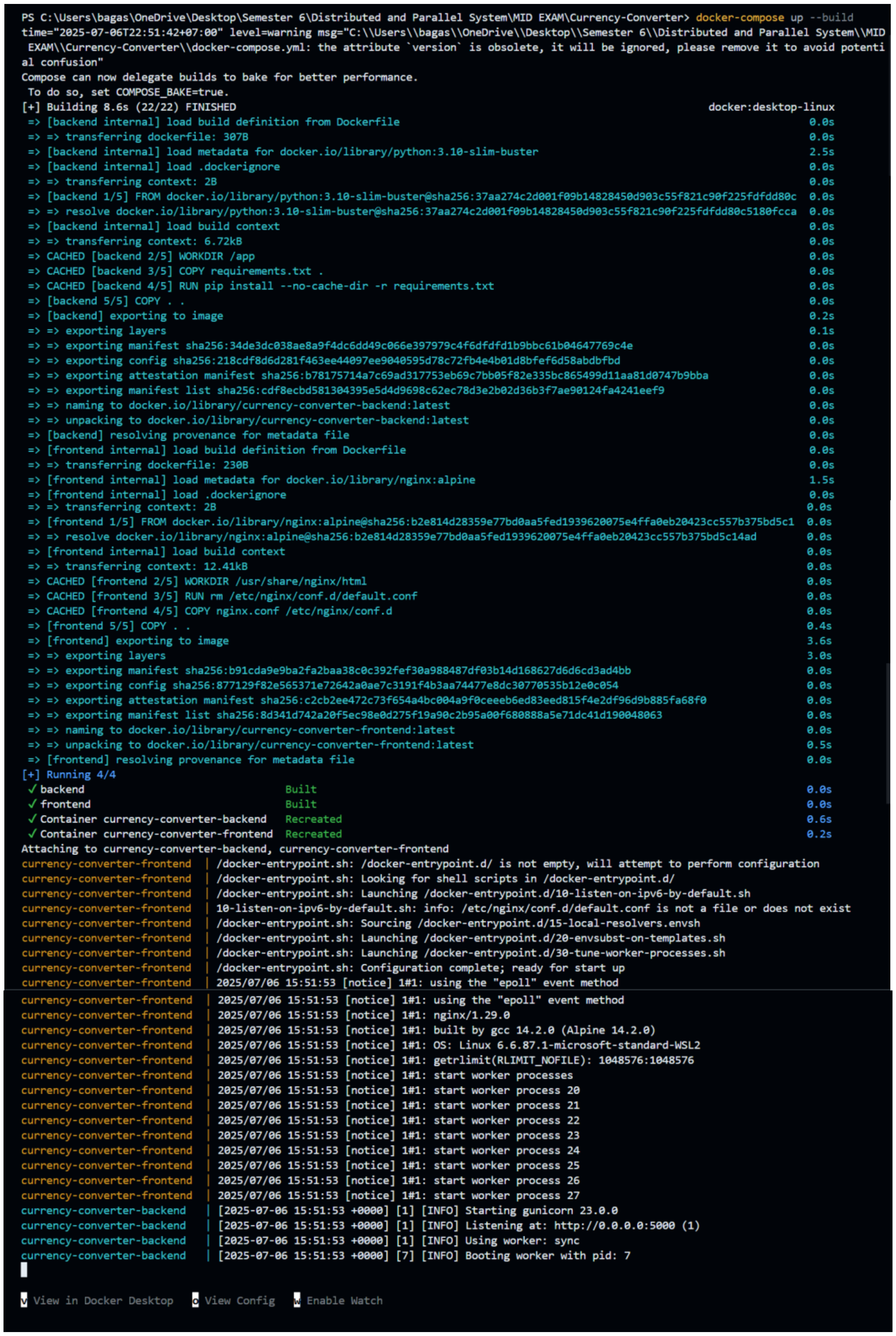


```
docker-compose up --build
```

   - The --build flag forces Docker to rebuild the images from the Dockerfiles if any changes have been made.

4. **Verify Logs:** Wait for the build process to complete. The terminal will display logs from both containers. Look for confirmation messages indicating that the Gunicorn and Nginx servers have started successfully.
5. **Access Application:** Once the containers are running, open a web browser and navigate to **`http://localhost:3000`** (or the host port specified in your `docker-compose.yml` for the frontend service).

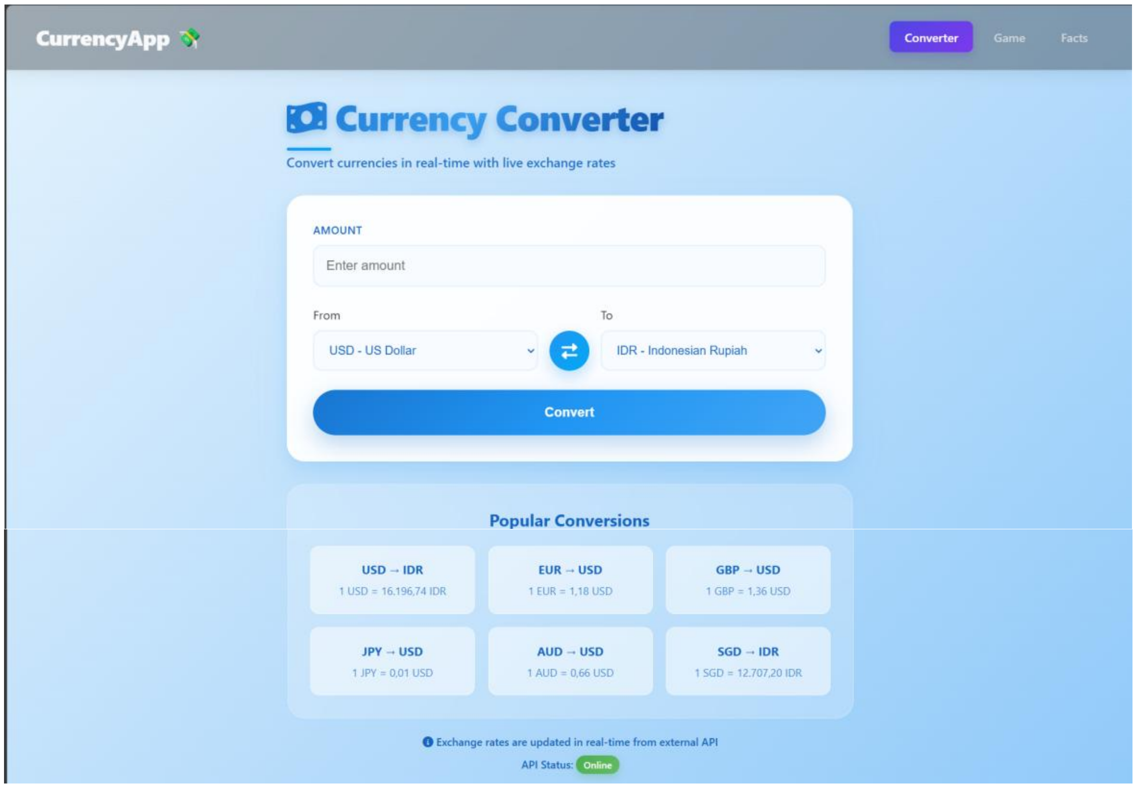
4.2 Results and Application Screenshots



```
PS C:\Users\bagas\OneDrive\Desktop\Semester 6\Distributed and Parallel System\MID EXAM\Currency-Converter> docker-compose up --build
time="2025-07-06T22:51:42+07:00" level=warning msg="C:\\Users\\bagas\\OneDrive\\Desktop\\Semester 6\\Distributed and Parallel System\\MID
 EXAM\\Currency-Converter\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potenti
al confusion"
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 8.6s (22/22) FINISHED                                                                               docker:desktop-linux
 => [backend internal] load build definition from Dockerfile                                                                     0.0s
 => => transferring dockerfile: 307B                                                                                             0.0s
 => [backend internal] load metadata for docker.io/library/python:3.10-slim-buster                                               2.5s
 => [backend internal] load .dockerignore                                                                                        0.0s
 => => transferring context: 2B                                                                                                  0.0s
 => [backend 1/5] FROM docker.io/library/python:3.10-slim-buster@sha256:37aa274c2d001f09b14828450d903c55f821c90f225fdfdd80c  0.0s
 => => resolve docker.io/library/python:3.10-slim-buster@sha256:37aa274c2d001f09b14828450d903c55f821c90f225fdfdd80c5180fcca  0.0s
 => [backend internal] load build context                                                                                        0.0s
 => => transferring context: 6.72kB                                                                                              0.0s
 => CACHED [backend 2/5] WORKDIR /app                                                                                            0.0s
 => CACHED [backend 3/5] COPY requirements.txt .                                                                                 0.0s
 => CACHED [backend 4/5] RUN pip install --no-cache-dir -r requirements.txt                                                      0.0s
 => [backend 5/5] COPY . .                                                                                                       0.0s
 => [backend] exporting to image                                                                                                 0.2s
 => => exporting layers                                                                                                          0.1s
 => => exporting manifest sha256:34de3dc038ae8a9f4dc6dd49c066e397979c4f6dfdfd1b9bbc61b04647769c4e                                0.0s
 => => exporting config sha256:218cdf8d6d281f463ee44097ee9040595d78c72fb4e4b01d8bfef6d58abdbfbd                                  0.0s
 => => exporting attestation manifest sha256:b78175714a7c69ad317753eb69c7bb05f82e335bc865499d11aa81d0747b9bba                    0.0s
 => => exporting manifest list sha256:cdf8ecbd581304395e5d4d9698c62ec78d3e2b02d36b3f7ae90124fa4241eef9                           0.0s
 => => naming to docker.io/library/currency-converter-backend:latest                                                             0.0s
 => => unpacking to docker.io/library/currency-converter-backend:latest                                                          0.0s
 => [backend] resolving provenance for metadata file                                                                             0.0s
 => [frontend internal] load build definition from Dockerfile                                                                    0.0s
 => => transferring dockerfile: 230B                                                                                             0.0s
 => [frontend internal] load metadata for docker.io/library/nginx:alpine                                                         1.5s
 => [frontend internal] load .dockerignore                                                                                       0.0s
 => => transferring context: 2B                                                                                                 0.0s
 => [frontend 1/5] FROM docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c1  0.0s
 => => resolve docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad         0.0s
 => [frontend internal] load build context                                                                                      0.0s
 => => transferring context: 12.41kB                                                                                            0.0s
 => CACHED [frontend 2/5] WORKDIR /usr/share/nginx/html                                                                         0.0s
 => CACHED [frontend 3/5] RUN rm /etc/nginx/conf.d/default.conf                                                                 0.0s
 => CACHED [frontend 4/5] COPY nginx.conf /etc/nginx/conf.d                                                                     0.0s
 => [frontend 5/5] COPY . .                                                                                                     0.4s
 => [frontend] exporting to image                                                                                               3.6s
 => => exporting layers                                                                                                         3.0s
 => => exporting manifest sha256:b91cda9e9ba2fa2baa38c0c392fef30a988487df03b14d168627d6d6cd3ad4bb                               0.0s
 => => exporting config sha256:877129f82e565371e72642a0ae7c3191f4b3aa74477e8dc30770535b12e0c054                                 0.0s
 => => exporting attestation manifest sha256:c2cb2ee472c73f654a4bc004a9f0ceeeb6ed83eed815f4e2df96d9b885fa68f0                   0.0s
 => => exporting manifest list sha256:8d341d742a20f5ec98e0d275f19a90c2b95a00f680888a5e71dc41d190048063                          0.0s
 => => naming to docker.io/library/currency-converter-frontend:latest                                                           0.0s
 => => unpacking to docker.io/library/currency-converter-frontend:latest                                                        0.5s
 => [frontend] resolving provenance for metadata file                                                                           0.0s
[+] Running 4/4
 ✓ backend                                Built                                                                                  0.0s
 ✓ frontend                               Built                                                                                  0.0s
 ✓ Container currency-converter-backend   Recreated                                                                              0.6s
 ✓ Container currency-converter-frontend  Recreated                                                                              0.2s
Attaching to currency-converter-backend, currency-converter-frontend
currency-converter-frontend  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
currency-converter-frontend  | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
currency-converter-frontend  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
currency-converter-frontend  | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
currency-converter-frontend  | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
currency-converter-frontend  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
currency-converter-frontend  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
currency-converter-frontend  | /docker-entrypoint.sh: Configuration complete; ready for start up
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: using the "epoll" event method
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: using the "epoll" event method
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: nginx/1.29.0
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: OS: Linux 6.6.87.1-microsoft-standard-WSL2
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker processes
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 20
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 21
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 22
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 23
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 24
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 25
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 26
currency-converter-frontend  | 2025/07/06 15:51:53 [notice] 1#1: start worker process 27
currency-converter-backend   | [2025-07-06 15:51:53 +0000] [1] [INFO] Starting gunicorn 23.0.0
currency-converter-backend   | [2025-07-06 15:51:53 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
currency-converter-backend   | [2025-07-06 15:51:53 +0000] [1] [INFO] Using worker: sync
currency-converter-backend   | [2025-07-06 15:51:53 +0000] [7] [INFO] Booting worker with pid: 7


v View in Docker Desktop   o View Config   w Enable Watch
```

Successful startup of *both frontend and backend services in the terminal.*


CurrencyApp 💸

Converter Game Facts

Currency Converter

Convert currencies in real-time with live exchange rates

AMOUNT

Enter amount

From: USD - US Dollar

To: IDR - Indonesian Rupiah

Convert

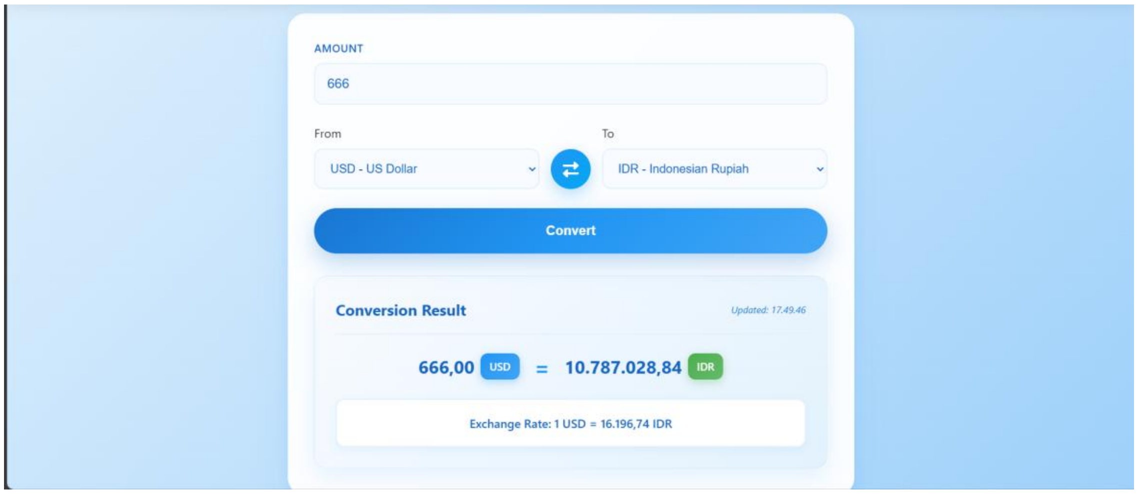
Popular Conversions

USD → IDR 1 USD = 16.196,74 IDR	EUR → USD 1 EUR = 1,18 USD	GBP → USD 1 GBP = 1,36 USD
JPY → USD 1 JPY = 0,01 USD	AUD → USD 1 AUD = 0,66 USD	SGD → IDR 1 SGD = 12.707,20 IDR

Exchange rates are updated in real-time from external API

API Status: Online

Main interface display of the Currency Converter application.


AMOUNT

666

From: USD - US Dollar

To: IDR - Indonesian Rupiah

Convert

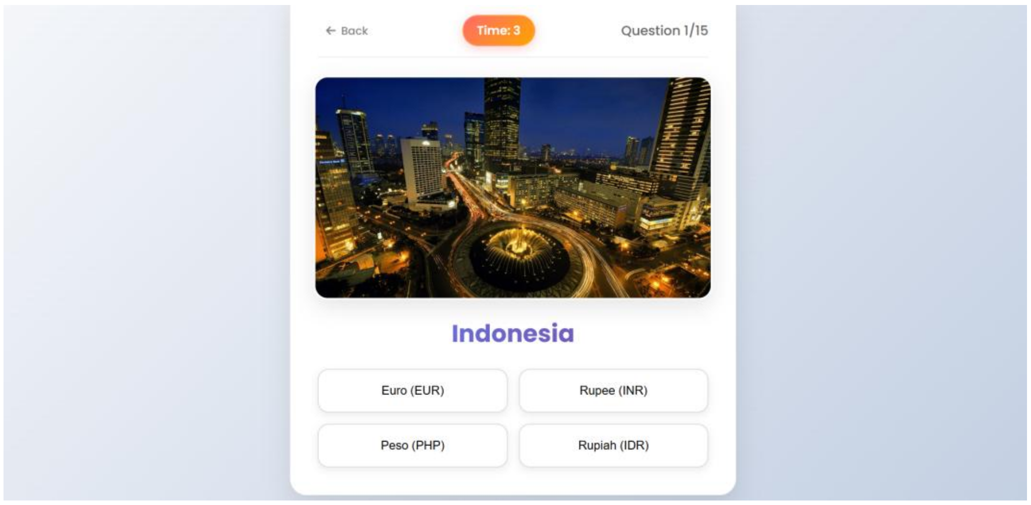
Conversion Result

Updated: 17.49.46

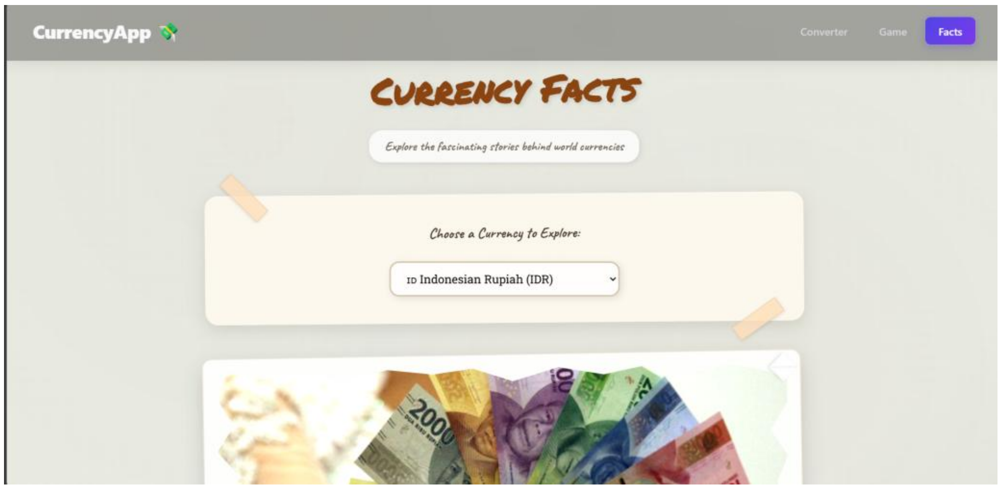
666,00 USD = 10.787.028,84 IDR

Exchange Rate: 1 USD = 16.196,74 IDR

Example of conversion results from USD to IDR, with results displayed dynamically.



Display of the application's additional feature, the currency guessing game.


CurrencyApp
Converter
Game
Facts
Currency Facts
Explore the fascinating stories behind world currencies
Choose a Currency to Explore:
ID Indonesian Rupiah (IDR)

An engaging financial information page display.

CHAPTER 5: CONCLUSION

5.1 Conclusion

This project effectively illustrates Docker's strength and usefulness in contemporary full-stack development. Our application is portable, consistent, and simple to deploy thanks to the frontend and backend services being containerized. The project's goals were completely achieved: the issue of environmental inconsistency was successfully resolved by creating a working application, containerizing it with separate Dockerfiles, and orchestrating it with Docker Compose.

5.2 Future Improvements

While the current application is fully functional, there are several potential areas for future enhancement:

- **Database Integration:** Persist conversion history for users by adding a database service (e.g., PostgreSQL or MongoDB) to the Docker Compose setup.
- **User Authentication:** Implement a user login system to provide personalized experiences.
- **CI/CD Pipeline:** Create a Continuous Integration/Continuous Deployment pipeline (e.g., using GitHub Actions) to automatically test and deploy the application upon code changes.
- **Frontend Framework:** Migrate the frontend to a modern JavaScript framework like React or Vue.js for a more robust and scalable user interface.

APPENDIX

GitHub Repository Link:

https://github.com/BagusEb/Currency-Converter