**Jose Rizal Memorial State University**

**Gov. Guading Adaza St., Sta. Cruz,**

**Dapitan City, Philippines**

**GSU Gateway User Manual: Setting Up Docker**

By:

Edrean Balonga

Bernadette Bantillan

Bryan Bantillan

Patrick Joshua Nicholas

Jenifer Saguindang

May 19, 2025

**General Service Office**

**Document Overview**

**Purpose**

This manual provides step-by-step instructions to set up and run a full-stack web application using Docker. The front-end is built with Vite and Tailwind CSS, while the back-end uses the Laravel PHP framework. Docker is used to containerize both parts of the application for consistent and portable development environments.

**Intended Audience**

* Developers setting up the project locally
* DevOps engineers managing containerized deployments
* Students or learners practicing full-stack Docker integration

**Prerequisites**

Before using this manual, ensure the following:

* Basic knowledge of the command line
* Git installed
* Docker Desktop installed on your machine

**Importance of Docker**

* **Consistency**: Ensures the application runs the same way across different environments.
* **Portability**: Allows moving containers between machines without compatibility issues.
* **Scalability**: Simplifies deployment and scaling of applications.
* **Isolation**: Keeps the application environment separate from the host system.

**Step 1: Install Docker Desktop**

*a. Download Docker*

Visit: https://www.docker.com/

*For Windows:*

If you're using Windows, the system or software you're trying to run requires either WSL2 or Hyper-V:

* *WSL2 (Windows Subsystem for Linux version 2):* A compatibility layer that allows you to run a Linux environment directly on Windows, including most command-line tools, utilities, and applications. It's useful for development environments that are built for Linux.
* *Hyper-V:* Microsoft's native hypervisor for running virtual machines on Windows. Some tools or setups (especially those using Docker Desktop or full virtualized Linux environments) might need Hyper-V enabled.

*For macOS:*

If you're using macOS, the system or software you're trying to run requires either Intel or Apple Silicon:

* *Intel Macs*: Older Mac computers that use Intel processors.
* *Apple Silicon Macs*: Newer Macs (like the M1, M2, etc.) that use Apple's custom ARM-based chips.

*b. Install and Verify*

* Follow installation prompts.
* After installation, open a terminal and run:

docker --version

You should see something like:

Docker version 24.0.5, build a45e4e4

The command docker --version is used in the terminal to check which version of Docker is installed on your system and to confirm that Docker is installed correctly and to ensure you're using a version that's compatible with the tools or applications you plan to use. When you run this command, it returns the current Docker version along with its build number.

**Step 2: Clone the GitHub Repository**

Open your terminal and run the following codes one at a time:

Cd C:\Users\DELL\Downloads\project-docker\GS

git clone -b JS <https://github.com/jensags/GS.git>

Replace C:\Users\DELL\Downloads\project-docker\GS with the path to the folder you want the clone code to be saved, and replace the URL with the actual git repository HTTPS URL of your repository with your actual branch name. This updates your local copy with the latest changes from GitHub.

*Suggested Project Structure:*

project-root/

├── frontend/ # For Frontend code

│ └── Dockerfile

├── backend/ # For backend code

│ └── Dockerfile

└── docker-compose.yml

**Step 3: Download Node.js, preferably version v20.19.0**

**Step 4: Create Dockerfiles**

A Dockerfile is a plain text file that contains instructions for building a Docker image. You can create it using any text editor or IDE, as long as it's saved with the correct name and format.

a.*Create frontend dockerfile*

1. Open your project folder using your preferred text editor or IDE. Like VS Code, Sublime Text, Atom, Notepad++ (Windows), TextEdit (macOS, in *plain text* mode), PhpStorm, IntelliJ IDEA, or any JetBrains IDE and Terminal editors: vim, nano, emacs.
2. Create a new file in the root of your project folder.
3. Name the file.
4. Open the file and start writing your Dockerfile instructions.  
   Example:

# Use the official Node.js 20 Alpine image

FROM node:20-alpine

# Set working directory inside the container

WORKDIR /app

# Copy only package.json and lock file

COPY package\*.json ./

# Install dependencies using clean install for consistency

RUN npm install

# Copy the rest of the application

COPY . .

# Expose the Vite development server port

EXPOSE 5173

# Optional: Use polling for file changes (needed on some OS setups)

ENV CHOKIDAR\_USEPOLLING=true

# Start the Vite development server

CMD ["npm", "run", "dev"]

1. Save the file in your project directory.

*Explanation for frontend dockerfile:*

# Use the official Node.js 20 Alpine image

FROM node:20-alpine

* This sets the **base image** for your container. You're using **Node.js version 20** with the **Alpine** variant, which is a lightweight Linux distribution that keeps your Docker image small and fast.

# Set working directory inside the container

WORKDIR /app

* This sets the **working directory** inside the container to /app. All following commands (like COPY, RUN) will execute relative to this directory.

# Copy only package.json and lock file

COPY package\*.json ./

* This copies the package.json and package-lock.json (or pnpm-lock.yaml if applicable) into the container. Doing this **before copying the full app** helps Docker cache the layer and avoid reinstalling dependencies every time the code changes.

# Install dependencies using clean install for consistency

RUN npm install

* Installs the project dependencies using npm install. You could use npm ci instead for even more consistency if you're using a lock file.

# Copy the rest of the application

COPY . .

* This copies the rest of your project files (like src/, vite.config.js, etc.) into the container.

# Expose the Vite development server port

EXPOSE 5173

* This tells Docker to expose **port 5173**, which is the default port used by Vite’s dev server.  
  (You'll still need to map this port when running the container using -p.)

# Optional: Use polling for file changes (needed on some OS setups)

ENV CHOKIDAR\_USEPOLLING=true

* This sets an environment variable that tells file watchers (like Vite's) to use polling instead of relying on system events. Useful if you're developing on a system (like WSL or Docker on Windows/macOS) where default file change detection doesn't work reliably.

# Start the Vite development server

CMD ["npm", "run", "dev"]

* This sets the **default command** when the container starts. Here, it runs the Vite dev server with npm run dev.

b.*Create backend dockerfile*

1. Open your project folder using your preferred text editor or IDE. Like VS Code, Sublime Text, Atom, Notepad++ (Windows), TextEdit (macOS, in *plain text* mode), PhpStorm, IntelliJ IDEA, or any JetBrains IDE and Terminal editors: vim, nano, emacs.
2. Create a new file in the root of your project folder.
3. Name the file.
4. Open the file and start writing your Dockerfile instructions.  
   Example:

FROM php:8.2-fpm

# Install system dependencies and PHP extensions

RUN apt-get update && apt-get install -y \

    curl zip unzip git \

    libpq-dev libpng-dev libjpeg-dev libfreetype6-dev \

    && docker-php-ext-install pdo pdo\_mysql gd

# Install Composer (PHP dependency manager)

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Set working directory in the container

WORKDIR /var/www/html

# Copy Laravel project files into the container

COPY . .

# Install Laravel dependencies using Composer

RUN composer install

# Set proper permissions

RUN chown -R www-data:www-data /var/www/html

# Expose the port the app runs on

EXPOSE 8000

# Command to run the Laravel server

CMD ["sh", "-c", "php artisan migrate && php artisan serve --host=0.0.0.0 --port=8000"]

1. Save the file in your project directory.

*Explanation for backend dockerfile:*

# Use the official PHP 8.2 FPM image

FROM php:8.2-fpm

* This sets the base image to PHP 8.2 FPM (FastCGI Process Manager). The FPM variant is commonly used for serving PHP applications in a production environment (it’s optimized for performance).

# Install system dependencies and PHP extensions

RUN apt-get update && apt-get install -y \

curl zip unzip git \

libpq-dev libpng-dev libjpeg-dev libfreetype6-dev \

&& docker-php-ext-install pdo pdo\_mysql gd

* This installs necessary system dependencies and PHP extensions for Laravel:

curl, zip, unzip, git: Utilities commonly required for development or installing other packages.

libpq-dev: PostgreSQL library (if you're using PostgreSQL with Laravel).

libpng-dev, libjpeg-dev, libfreetype6-dev: Libraries for image processing, needed for the gd extension.

docker-php-ext-install: Installs the PHP extensions pdo, pdo\_mysql, and gd, which are required for interacting with MySQL databases and handling images.

# Install Composer (PHP dependency manager)

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

* This command installs Composer (PHP's dependency manager) by copying it from the official Composer image. It’s more efficient than downloading and installing Composer separately.

# Set working directory in the container

WORKDIR /var/www/html

* This sets the working directory inside the container to /var/www/html, which is where your Laravel app will reside.

# Copy Laravel project files into the container

COPY . .

* This copies all the project files from your host machine (the current directory) into the container's /var/www/html directory.

# Install Laravel dependencies using Composer

RUN composer install

* This runs composer install inside the container, which installs all the Laravel dependencies listed in composer.json.

# Set proper permissions

RUN chown -R www-data:www-data /var/www/html

* This ensures that the files in /var/www/html are owned by the www-data user and group (which is commonly used for web servers like Nginx and Apache). It’s important for setting correct file permissions to avoid issues when the app is accessed by the web server.

# Expose the port the app runs on

EXPOSE 8000

* This exposes port 8000 from the container. By default, Laravel’s development server runs on port 8000, so this makes it accessible outside the container.

# Command to run the Laravel server

CMD ["sh", "-c", "php artisan migrate && php artisan serve --host=0.0.0.0 --port=8000"]

* This starts the Laravel application inside the container:

php artisan migrate: Runs any outstanding database migrations (useful for setting up the database).

php artisan serve --host=0.0.0.0 --port=8000: Starts the Laravel development server, binding it to 0.0.0.0 to make it accessible externally and on port 8000.

**Step 5: Create docker-compose.yml in Root**

The docker-compose.yml file is use to **define and manage multi-container Docker applications** in a single, easy-to-read configuration file. Instead of running multiple docker run commands manually for each container. The following are the steps to creating a docker-compose.yml file:

1. Open your project folder using your preferred text editor or IDE. Like VS Code, Sublime Text, Atom, Notepad++ (Windows), TextEdit (macOS, in *plain text* mode), PhpStorm, IntelliJ IDEA, or any JetBrains IDE and Terminal editors: vim, nano, emacs.
2. Create a new file in the root of your project folder.
3. Name the file.
4. Open the file and start writing your Dockerfile instructions.  
   Example:

services:

  mysql:

    image: mysql:8.0

    container\_name: gso\_mysql

    restart: always

    environment:

      MYSQL\_ROOT\_PASSWORD: root

      MYSQL\_DATABASE: aftermvp

      MYSQL\_USER: gso\_user

      MYSQL\_PASSWORD: manageitgso5

    ports:

      - "4406:3306"  # Avoid conflict with local MySQL

    volumes:

      - mysql\_data:/var/lib/mysql

    networks:

      - gso\_network

  backend:

    build:

      context: ./docker-gsobackend

      dockerfile: backend.dockerfile

    container\_name: gso\_backend

    restart: always

    depends\_on:

      - mysql

    environment:

      DB\_CONNECTION: mysql

      DB\_HOST: gso\_mysql

      DB\_PORT: 3306

      DB\_DATABASE: aftermvp

      DB\_USERNAME: gso\_user

      DB\_PASSWORD: manageitgso5

    ports:

      - "8000:8000"

    volumes:

      - ./docker-gsobackend:/var/www/html

    command: ["sh", "-c", "php artisan migrate && php artisan serve --host=0.0.0.0 --port=8000"]

    networks:

      - gso\_network

  frontend:

    build:

      context: ./docker-gsofrontend/GSO Website

      dockerfile: ../../docker-gsofrontend/frontend.dockerfile

    container\_name: gso\_frontend

    restart: always

    ports:

      - "80:5173"

    volumes:

      - ./docker-gsofrontend/GSO Website:/app

    working\_dir: /app

    environment:

      - CHOKIDAR\_USEPOLLING=true

    command: ["sh", "-c", "npm install && npm run dev"]

    networks:

      - gso\_network

  phpmyadmin:

    image: phpmyadmin/phpmyadmin

    container\_name: gso\_phpmyadmin

    restart: always

    environment:

      PMA\_HOST: gso\_mysql

      PMA\_USER: root

      PMA\_PASSWORD: root

    ports:

      - "8080:80"  # Access phpMyAdmin via localhost:8080

    depends\_on:

      - mysql

    networks:

      - gso\_network

volumes:

  mysql\_data:

networks:

  gso\_network:

    driver: bridge

1. Save the file in your project directory.

*Explanation for backend dockerfile:*

MySQL Service:

mysql:

image: mysql:8.0

* Uses the official MySQL 8.0 image from Docker Hub. This serves as your database engine.

container\_name: gso\_mysql

* Names the container **gso\_mysql**, making it easier to reference in other services like Laravel.

restart: always

* Automatically restarts the MySQL container if it crashes or Docker is restarted.

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: aftermvp

MYSQL\_USER: gso\_user

MYSQL\_PASSWORD: manageitgso5

* Sets up initial database credentials:
  + MYSQL\_ROOT\_PASSWORD: root password for MySQL.
  + MYSQL\_DATABASE: creates a database named aftermvp.
  + MYSQL\_USER and MYSQL\_PASSWORD: creates a user with access to the database.

ports:

- "4406:3306"

* Maps port 3306 inside the container to 4406 on your local machine, avoiding conflict with a local MySQL install.

volumes:

- mysql\_data:/var/lib/mysql

* Uses a named volume (mysql\_data) to persist data, so your database isn't wiped when the container stops.

networks:

- gso\_network

* Connects this service to a custom Docker network **(gso\_network)**, allowing communication with other services.

Backend (Laravel) Service:

backend:

build:

context: ./docker-gsobackend

dockerfile: backend.dockerfile

* Builds the backend image from a custom Dockerfile located in ./docker-gsobackend.

container\_name: gso\_backend

restart: always

depends\_on:

- mysql

* Names the container gso\_backend, restarts automatically, and ensures MySQL starts first.

environment:

DB\_CONNECTION: mysql

DB\_HOST: gso\_mysql

DB\_PORT: 3306

DB\_DATABASE: aftermvp

DB\_USERNAME: gso\_user

DB\_PASSWORD: manageitgso5

* Provides Laravel with database connection details via environment variables, matching the MySQL setup.

ports:

- "8000:8000"

* Exposes Laravel on port 8000 (common default for the built-in Laravel server).

volumes:

- ./docker-gsobackend:/var/www/html

* Mounts the Laravel project folder to the container for real-time syncing and code editing.

command: ["sh", "-c", "php artisan migrate && php artisan serve --host=0.0.0.0 --port=8000"]

* Runs Laravel migrations, then starts the server:
  + php artisan migrate: Runs any outstanding migrations.
  + php artisan serve: Starts Laravel dev server on 0.0.0.0:8000.

networks:

- gso\_network

* Joins the custom network to allow communication with the database and frontend.

Frontend (Vite) Service:

frontend:

build:

context: ./docker-gsofrontend/GSO Website

dockerfile: ../../docker-gsofrontend/frontend.dockerfile

* Builds the Vite frontend from a Dockerfile located in the parent directory of the context folder.

container\_name: gso\_frontend

restart: always

* Names the container gso\_frontend and restarts it if it fails.

ports:

- "80:5173"

* Maps the Vite dev server (port 5173) to port 80, so users can visit http://localhost instead of :5173.

volumes:

- ./docker-gsofrontend/GSO Website:/app

working\_dir: /app

* Mounts the source code into the /app directory and sets that as the working directory.

environment:

- CHOKIDAR\_USEPOLLING=true

* Enables polling to detect file changes properly on all operating systems (especially Windows/macOS).

command: ["sh", "-c", "npm install && npm run dev"]

* Installs Node.js dependencies and starts the Vite dev server inside the container.

networks:

- gso\_network

* Joins the custom Docker network so it can later communicate with the Laravel API if needed.

PhpMyAdmin Service:

phpmyadmin:

image: phpmyadmin/phpmyadmin

container\_name: gso\_phpmyadmin

restart: always

* Uses the official PhpMyAdmin image and names it gso\_phpmyadmin.

environment:

PMA\_HOST: gso\_mysql

PMA\_USER: root

PMA\_PASSWORD: root

* Links PhpMyAdmin to the MySQL container using the root credentials set earlier.

ports:

- "8080:80"

* Maps port 80 inside the container to 8080 on your host, so PhpMyAdmin is accessible at http://localhost:8080.

depends\_on:

- mysql

* Ensures PhpMyAdmin only starts after MySQL is running.

networks:

- gso\_network

* Joins the shared network, making it able to communicate with the MySQL service.

Volumes:

volumes:

mysql\_data:

* Creates a named volume (mysql\_data) to store MySQL database data persistently.

Networks:

networks:

gso\_network:

driver: bridge

* Creates a custom bridge network, allowing isolated communication between all the defined services.

### ****Step 6: Environment Setup****

Ensure that your Laravel backend and Vite frontend can properly communicate with the database and with each other through Docker.

#### a. **Configure Laravel Environment File**

Your Laravel project contains a .env file in the root of the backend directory. This file holds configuration values, including database connection details.

**Open** backend/.env**, and locate the database section. Update it to match your** docker-compose.yml **settings:**

DB\_CONNECTION=mysql # Use MySQL as the database driver

DB\_HOST=db # Match this with the MySQL service name in docker-compose

DB\_PORT=3306 # The default MySQL port inside Docker

DB\_DATABASE=laravel # The name of your database (also declared in docker-compose)

DB\_USERNAME=user # MySQL user created by docker- compose

DB\_PASSWORD=password # The corresponding password

This configuration allows Laravel to connect to the MySQL container started by Docker Compose.

#### **2. Configure Vite Front-End Environment File (**frontend/.env**)**

Your front-end (likely React or Vue) uses Vite, which supports environment variables using .env files. This is where you define the backend API URL.

**Create or open** frontend/.env**, and add:**

VITE\_API\_URL=http://localhost:8000/api

This tells your frontend where to send API requests. Vite automatically exposes environment variables that start with VITE\_ inside your app. You can access it in your front-end code like:

fetch(`${import.meta.env.VITE\_API\_URL}/posts`)

**Step 7: Build Docker Images and Restart the Docker Containers**

To build the docker images and restart the docker containers, open your terminal and run the following codes one at a time:

Cd C:\Users\DELL\Downloads\project-docker\GS

Docker-compose up -–build -d

Replace C:\Users\DELL\Downloads\project-docker\GS with the path to the folder where your project is saved. The second code will process the Dockerfile for all services. The -d means "detached mode" (runs in background) and it starts the containers with the latest images.

**Step 8: Verify the Containers**

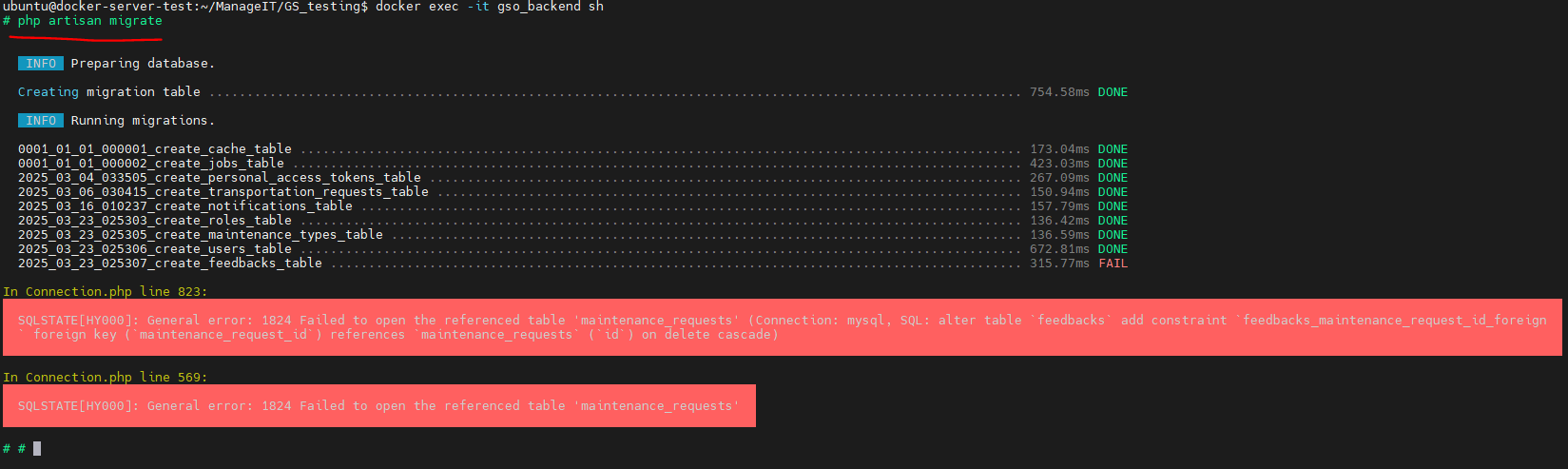
Check if the containers are running correctly through opening your docker app, and click the **localhost URL with a port.** The front-end should load or it should show the welcome page or login page of your frontend, and there should be no error messages in the browser console or white screen. And for backend, it should also show the Laravel welcome page, API response, or JSON (if it's an API) or a status code 200 OK or relevant success message.

You can also run:

docker compose logs -f

To view logs and confirm the new version is active. If there is an error in the container, fix it.

**Step 9: Error in Migration of Backend (Optional)**

****After checking the logs and there is an error in the migration of backend like image below, follow the steps after the image:

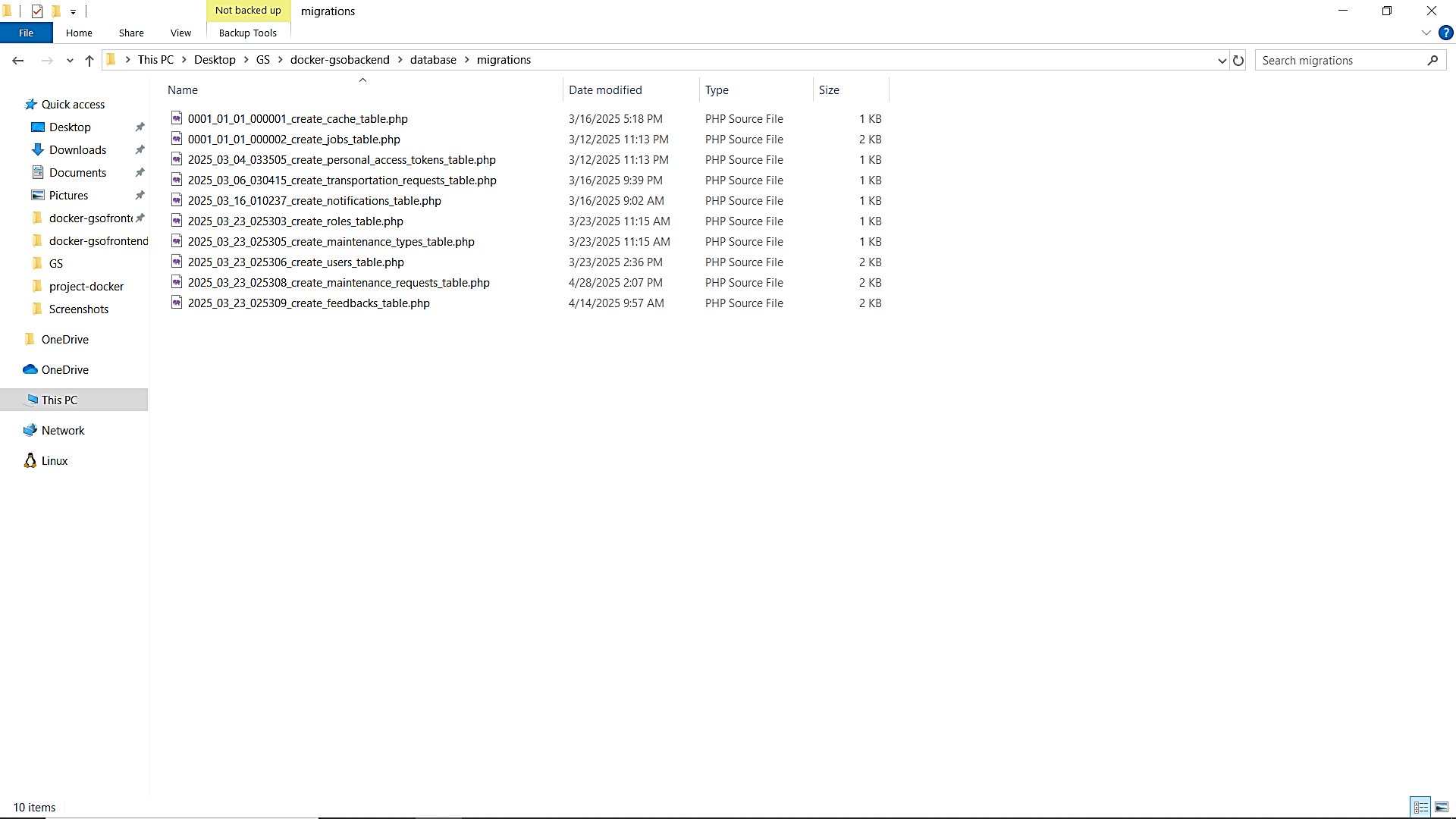
*1. Stop and remove the containers, networks, the default network, the named and anonymous volumes attached to the services, containers not defined in your current* ***docker-compose.yml*** *but connected to the same default network using the command below:*

docker-compose down --volumes --remove-orphans

2. Open the file manager and locate where your migrations located. It is usually located in the database file name of your backend code file.

3. Once you locate the migrations, observe the arrange of the tables and look if the feedback table comes first before the maintenance request table.

4. If the maintenance request table is the last in the arrangement of the file, edit the file name of the feedback table to make to the last order of the table file like the following:

5. Then after editing the order of the migrations, run the following command to build the containers again:

docker-compose up --build -d

6. Check if the containers are running and if the container of the backend is already fix.

**Important note: Note that vendor file in backend is not pushed in github as it is ignored in .gitignored file since vendor is a big file it would bloat the Git repository and slow down cloning and pushing and package updates change files in vendor/, so it can cause a mess in Git if is included.**

**Common Docker Commands**

The following are common docker commands that can be use for managing and troubleshooting Docker containers:

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| *1. Change Directory* | cd C:\Users\DELL\Downloads\project-docker\GS | Navigate to the directory where you want to clone the project. |
| *2. Clone Repo* | git clone -b <branch-name> https://github.com/<username>/<repository-name>.git | Clone the latest code from GitHub. Replace <branch-name>, <username>, and <repository-name>. |
| *3. Stop & Remove Containers* | docker-compose down | Stops and removes current containers and networks. |
| *4. Rebuild & Start Containers* | docker-compose up --build -d | Rebuilds images and starts containers in detached mode (background). |
| *5. Check Logs* | docker compose logs -f | Streams logs from containers to verify successful startup or detect errors. |
| *6. View Running Containers* | docker ps | Lists all currently running containers. |
| *7. Stop Running Containers* | docker stop <container-name> | Stops a specific running container. |
| *8. Start a Stopped Container* | docker start <container-name> | Starts a previously stopped container. |
| *9. Restart a Container* | docker restart <container-name> | Restarts a container. |
| *10. Remove a Container* | docker rm <container-name> | Removes a stopped container from your system. |
| *11. View Container Logs* | docker logs <container-name> | Displays the logs of a specific container. |
| *12. Access Container Terminal* | docker exec -it <container-name> /bin/bash | Opens an interactive terminal session inside a running container. |
| *13. View All Containers (Running and Stopped)* | docker ps -a | Lists all containers, whether running or stopped. |
| *14. List All Docker Images* | docker images | Shows all the Docker images on your machine. |
| *15. Remove a Docker Image* | docker rmi <image-name> | Removes a Docker image from your local system. |

**Conclusion**

This manual has guided you through the process of setting up a Dockerized environment for your full-stack web application. By utilizing Docker, you have created a consistent, portable, and isolated environment for both the front-end and back-end of the project. Docker ensures that your application will function identically across various systems and environments, streamlining the development, testing, and deployment processes.

Should you encounter any challenges or require further customization, we recommend consulting the official Docker documentation or seeking assistance from your development team.

Thank you for following this guide, and we trust that Docker will enhance the efficiency and scalability of your project.