Dsy Challenge Task: Image Puzzle

Team

- Oliver Dietsche
- Josip di Benedetto
- Joshua Beny Hürzeler

Table of Contents

How to run

- Install Docker and Docker Compose
- Run `docker-compose up --build` in the root directory of the project
- Frontend can be opened in browser with https://imagepuzzle.docker.localhost
- Backend can be opened in browser with https://imagepuzzle.docker.localhost/api/

Tipps

- In Chrome it should work and resolve domain. If not, add the domain `imagepuzzle.docker.localhost` to the local host file forwarding to `127.0.0.1`
- For viewing the database in development i recommend using HeidiSQL. Please add following to the docker-compose.yml below the `database:` to expose database to host:

```
ports:
  - "5432:5432"
```

- certificate warning can be ignored
- to see http/3 you need to first reload frontend once

Project Idea

We wanted to develop a Image Guesser Game similar to the one from the swiss online newspaper Watson. There the came is called "AUFGEDECKT"

Tech-Stack

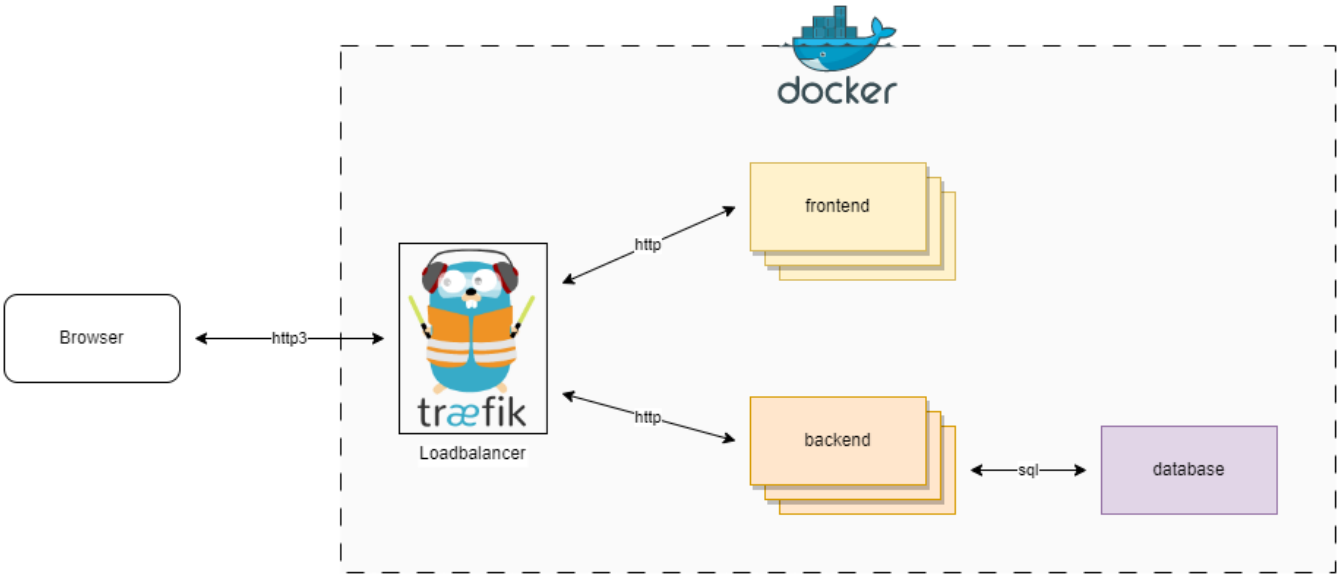
We decided to use simple technologies we all know or wanted to deepen our knowledge. The following technologies are used:

- Frontend: plain HTML/CSS/JS with Caddy as webserver
- Backend: Python Flask
- Database: Postgres
- Load-Balancer: Traefik
- Containerization: Docker Compose

Responsibilities

- Oliver Dietsche: Frontend
- Josip di Benedetto: Backend
- Joshua Beny Hürzeler: Load Balancer, Database

Architecture



Load Balancer

- Traefik is used as load balancer
- It is set up with manual configuration (file provider), not using the autodiscovery feature from traefik.
- load balancer is configured to route all requests to backend and frontend.
- it is decided to use a single domain and use path based routing to route to backend and frontend. This is because of CORS and keeping it simple
- everything under the path /api/* is redirected to the backend and everything else to the frontend.
- certificates are generated with mkcert and self-signed. So there will be a warning in the browser.
- The configuration is located in docker-compose.yml and in the loadbalancer folder.
- Traefik check if frontend and backend is available via health check every 5s.

Frontend

- Caddy is used as webserver
- A simple HTML page is served which contains the frontend logic
- The frontend is located in the frontend folder
- Frontend has following features:
  - shows title of game
  - shows a 8x8 grid
  - shows a text field and submit button to enter guess
  - user can click on a grid field to reveal the image fragment
  - image will be passive directly loaded from webserver
  - in js only request to get initial image ID and guesses are made to backend via json
  - count how many guesses and how many image fragments are revealed and show the stat in the end
  - give up button which shows the whole image and correct word
  - can show the log of all guesses

Backend

- Python Flask is used as backend
- Backend is located in the backend folder
- Backend has following features:
    - send requested image fragment to frontend (parse from url and send correct fragment)
    - receive guess from frontend and check if correct
    - save all guesses in log table of database (guess, image, correct/incorrect)
    - answer on give up request
    - provide all entries from log table
    - has a health check endpoint which checks if database is available

Database

- Postgres is used as database
- Database is located in the database folder
- There are migration scripts which create tables import some initial images to the database
- it is only accessed by backend container and not duplicated
- Database has following tables:
    - table with all images fragmented with index and image name and content
    - Log table with all guesses

Database Schema

```
CREATE TABLE image (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    solution VARCHAR(255) NOT NULL,
    image BYTEA NOT NULL
);

CREATE TABLE imagefragment (
    id SERIAL PRIMARY KEY,
    image_id UUID NOT NULL,
    position INTEGER NOT NULL,
    fragment BYTEA NOT NULL,
    FOREIGN KEY (image_id) REFERENCES image(id)
);

CREATE TABLE guesslog (
    id SERIAL PRIMARY KEY,
    image_id UUID NOT NULL,
    guess VARCHAR(255) NOT NULL,
    correct BOOLEAN NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    FOREIGN KEY (image_id) REFERENCES image(id)
);
```

API Endpoints

URL Arguments

- imageId: UUID
- position: Integer (0-63)

GET `/random-image-id`

Response

```
{
    "imageId": "asdf"
}
```

POST `/guess`

Request

```
{
    "imageId": "b870f836-7999-4d25-9285-1c8ddede5cbe",
    "guess": "Bat"
}
```

Response

```
{
    "correct": true
}
```

GET `/img/{imageId}/{position}`

Response

returns image fragment at position

GET `/solution/{imageId}`

Response

```
{
    "solution": "Bat"
}
```

GET `/img/{imageId}`

Response

returns original image

GET `/guesslog`

Response

```
[
    {"correct":true,"created_at":"Tue, 28 Mar 2023 16:13:24
GMT","guess":"flower","image_id":"b870f836-7999-4d25-9285-1c8ddede5cbe"},
    {"correct":false,"created_at":"Tue, 28 Mar 2023 16:13:44
GMT","guess":"blume","image_id":"b870f836-7999-4d25-9285-1c8ddede5cbe"}
]
```

GET `/health`

Response

```
{
    "status": "ok"
}
```

Images

Images are downloaded from Unsplash. The images are licensed under Unsplash License. They are cropped square and located inside the database/images/in folder. The file name is the solution word. For initial import a script was created (see database/images/import-images.py). To run the script the database port needs to be exposed to localhost:5432 (Edit docker-compose.yml i*f* needed). Conda was used to install the requirements and run the script. After initial script run a pgdump was created and placed inside the database/migrations folder. How to run script with conda:

1. 'cd database/images'
2. 'conda install --file .\requirements.txt'
3. 'python import-images.py'
4. 'docker exec dsy-challenge-task-database-1 pg_dump -E UTF8 -U postgres --column-inserts --data-only postgres > ../migrations/1_fill-database-with-content.sql'

Lessons Learned

Oliver

Working on this project, I got introduced to some technologies that were completely foreign to me: Caddy and Traefik. In the past when I had to serve files from a server, I used nginx or apache, but I've never configured load balancing before. For me it's still a little bit confusing that there are so many different technologies you can use, some can serve files, some can do load balancing and some can do both or even more. In our case we used Caddy to serve our files because we wanted to try out something we didn't know prior to this module. And for the same reason we also decided to use Traefik to do the load balancing even though Caddy is also capable of doing it.

I learned how to organize multiple repositories in one monorepo and for a project at this scale I think it works very well and it would be overhead to manage one repository per subproject. Other benefits I value of a monorepo are that as a frontend developer, I can take a look at how they implemented endpoints in the backend directly within the same project. At the same time I also know benefits of going the extra mile and having one repository per subproject. I think the most significant benefits are: you can manage access individually, you don't pull code that don't directly concern you, you avoid potential merge-conflicts.

Working with docker wasn't a first time thing for me. I learned that, due to its aggressive caching, it's really important to carefully decide on the order of statements withing a dockerfile. Lets consider the frontend dockerfile. We have to place the statement that copies the files to be served from our filesystem into the container towards the end of the file. If we would have put it before e.g. the statements that download the requirements, these requirements would be downloaded again and again every time we make a change in any of the files served.

Even though I wasn't working on the database, I learned some things while discussing code with my teammates. Coding a script that automates filling the database with images seems very challenging to me, because I've never connected to a database within python before.

Josip

This module was a great opportunity for me to refresh my knowledge of docker and other new technologies that I was slightly aware of, like load balancing and HTTP3. I already worked with docker when finishing my apprenticeship, it was new by that time. This project contributed to expand my knowledge and skills in application-containerization.

I already worked with nginx and Apache server before. Caddy web server was new to me. Since I am new to software engineering, I also learned a lot about application architecture and how to manage multiple parts of the application (frontend, backend, database, traefik, caddy) as a containerized app in a monorepo. Apart from SE project, I now have a bit more of experience in creating and managing the backend, which contributed to my python knowledge and also how to manage project development with git.

At the beginning of the project, I had to read a bit about docker-compose, how the configuration works, and also how to interact with the database from the backend while running both applications as containers.

Joshua

I already had knowlege with docker, which i could deepen in this project. What was new for me was caddy and also traefik. I learned how to configure them and using them. They are both very easy to use and configure. I think I will use them in the future. Especially caddy is a nice alternative for using as webserver inside docker instead of apache or nginx, due it is more lightweight. Traefik autoconfiguration is a bit overwhelming for me and I did not understood it completly. Therefore, also to have more control over the configuration, I decided to use manual configuration via file provider.

What I also learned is to use the docker-compose file to also build frontend, backend and database which are located in subdirectories. I like the structure as mono repo. It is great for this small project. But I think for bigger projects it is better to have the frontend, backend and database in separate repositories.

Challenging was the importing of the images into the database. I had to invest some time to get a script for splitting the images and importing them into the database. I used python for that task, due there was a libary for simple splitting the images and also for connecting to the database.