Adap CW#01 Homework

Project name: <not specified yet>

Project repository: https://github.com/PVahldiek/wahlzeit

This week's tag: adap-hw01 on main

This week's diff: <begins-next-week>

What did i do this week?

- **This week's required content:**
  I set up the Github account by forking the original wahlzeit repository from http://github.com/dirkriehle/wahlzeit into my personal git repository: https://github.com/PVahldiek/wahlzeit.

- **Set-up environment:**
  In order to set up the wahlzeit environment, i installed the necessary applications. I already have Java JDK 11 installed. I checked if „java -version" works – it did. Also i installed Docker Desktop on my Windows PC. Afterwards i cloned the forked repository on my local PC with the command:
  ```
  git clone https://github.com/PVahldiek/wahlzeit.git --config core.autocrlf=input
  ```
  I did not install the PostgreSQL server, because i wanted to use the docker container. This took me a bit of time, because initially building docker container on my local PC failed. I needed to install a „WSL-driver for Linux Kernel". After the install of the kernel and a reset to factory settings in Docker desktop, it worked and i could start containers.

- **Build and start application:**
  Like that i builded the docker container with „docker-compose build" and started it with „docker-compose up". After opening the application in my webbrowser, i created an account and added three random pictures. I placed a screenshot with the application and the photos for illustration at the end of the document.

- **Commit and tag with adap-hw01 on GitHub**
  I made a commit and tagged it afterwards on GitHub.

Compare Dockerfiles:

The Dockerfile.simple uses an „alpine-slim" image. This is done in the FROM Statement. Next the gradle files are copied into the /app/ directory. Also the source files are copied into /app/src and the gradle resources are copied into /app/gradle and /app/gradlew. Then the directory is switched to the /app, where the needed gradle and source files are located for building the app. All tests are executed, and the container is published at port 8080. At last it builds and runs the project with the „./gradlew appRun" command.

The Dockerfile (multi stage) does all the steps taken in the Dockerfile.simple generally as well. But it does not build and run the application right away. It is divided into two seperate stages. The stages

begin with the FROM Statements. So a multi-stage Dockerfile has multiple FROM Statements. In the first stage, after executing the tests with ./gradlew test, it does not build and run the application with ./gradlew appRun. Instead it only builds the projects with „./gradlew assemble“. Before building, it also copies all necesarry files for building, like in the Dockerfile.simple. In the second stage, it uses a jetty image. This is a smaller image, with a jetty webserver installed. In the next step, it uses the built files from the previous container that are generated during the built (.war files). It copies these files into the /var/lib/jetty directory from the second container, which is used from the webserver. Afterwards the container is published on port 8080.

Differences:

- No build and run straight away, instead seperated into a build stage and a run stage
- Using two images, one for building and one for running the app.

Advantages:

- After building the app, the size of the image for running the application can be much smaller than the one for building. So the image size can be reduced by multi-stage Dockerfiles. It uses a small image, where only the jetty webserver and a few components, necesarry for running the application are installed.
- Also it is much easier to handle. On larger scripts, the division of the script makes it easy to maintain. Also with multi-stage Dockerfiles, there is the possibility to have separate stages for application deployment (like DEV Stage, PRODUCTION Stage etc.)

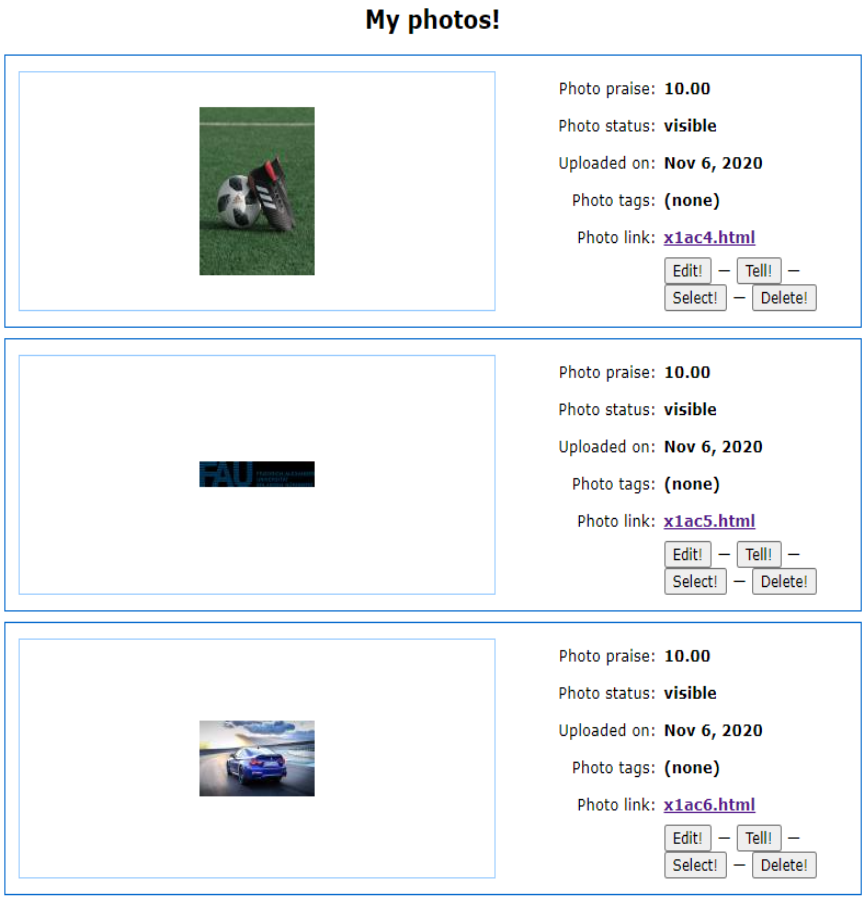

Abbildung 1: Wahlzeit pictures