ADAP CW#02

# Projektdaten

Projekt name: 3D-Prints

Projekt repository: [https://github.com/**ModischFabrications**/wahlzeit](https://github.com/ModischFabrications/wahlzeit)

This week’s tag: adap-cw02 on master

Homework diff: https://github.com/**ModischFabrications**/repository/compare/adap-ws16-start...**ModischFabrications**:adap-cw02

CI: [https://travis-ci.org/**ModischFabrications**/wahlzeit](https://travis-ci.org/ModischFabrications/wahlzeit)

Docker Hub Repo: [https://hub.docker.com/r/**modischfabrications**/wahlzeit](https://hub.docker.com/r/modischfabrications/wahlzeit)

# Hausaufgaben

## Implementation

### Neue Klassen (falls vorhanden)

[…]

### Veränderte Klassen (falls vorhanden)

[…]

### Erklärung

Ich habe mir zuerst das Thema “3D-Printing” ausgesucht, da ich dort eine breite Menge eigener Bilder ohne spezielle Lizenzbedingungen besitze und die Domäne interessant und leicht zu ergänzen ist. Nachteil ist eventuell die Nomenklatur, die Objekte werden selbst auch oft als “(3d) model” bezeichnet, dies kann mit den bisherigen Java-Klassen kollidieren.

Dann habe ich per GitHub UI einen Fork erstellt und diesen geklont. Das Readme dieser Version habe ich markiert, um Verwechslungen auszuschließen und anschließend den “Basis”-Stand mit “adap-initial” getagged.

Travis CI habe ich anschließend durch anlegen einer Verknüpfung mit meinem Repository und eintragen von DOCKERHUB\_USER & DOCKERHUB\_PW (access token, kein echtes Passwort!) eingerichtet, der Build war problemlos möglich. Im Rahmen dieser Maßnahme wurde auch das Docker Image erfolgreich gebaut und hochgeladen.

<https://fsi.cs.fau.de/forum/thread/17715-Fehlermeldungen-erstes-Projektsetup>

Added pictures to wahlzeit\build\resources\main\pictures as default

* Fixing Problems (Windows, IDEA 2019.2.3, Java 11)
  + “Port already in use” -> restart PC/kill
  + Gradle: unable to delete file ->
  + Debugger not working -> localhost:8000 works, run muss aber manuell ausgeführt werden

## Fragen / Aufgaben

### Dockerfiles

Eine “Dockerfile” wird als Instruktionsliste genutzt, um eine Anwendung mit Docker zu containerisieren, also in einer isolierten Umgebung ausführbar zu machen. Basis beider Container ist ein openjdk Image, dass eine Implementierung von Java mitbringt, für einen single-stage build unüblich ist dieses hier als builder definiert. Als nächstes werden alle relevanten Dateien kopiert, implizit gebaut und dann getestet. Zuletzt wird der zu benutzende Port markiert und der Zugangspunkt des Containers festgelegt.

Die Unterschiede von “Dockerfile” zu “simple-Dockerfile” sind vor allem die Verwendung eines multi-stage builds, also eines mehrstufigen Bauprozesses. Dieser erlaubt es, nur die gebauten Files im finalen Container mitzunehmen und alle temporären Artefakte des Bauprozesses “zurückzulassen”, wodurch die Containergröße deutlich reduziert werden kann. Der zweite Unterschied ist die Verwendung eines manuell definierten Zugangspunktes, der nicht auf die von gradlew bereitgestellten Shortcuts zurückgreift.