Assignment

Anton Jürß, Tijana Milentijevic, Robin Sadeghpour Faraj

Information Systems Engineering
TU Berlin, Germany
tijana.milentijevic@campus.tu-berlin.de
juerss@campus.tu-berlin.de
sadeghpour@campus.tu-berlin.de

1 Introduction

The microservice demo of Google implements an online boutique. We have noticed it has no review feature. Therefore, we decided to implement it with our microservice. Our project and the associated repositories are available at: `https://github.com/Continious-Software-Engineering`.

2 Development Approach

This project was developed in an agile manner. More specifically we used the Kanban approach because of the short production time and small team size. We set up a Github organization and a Github Project Kanban board with four columns: To do, In progress, In Review and Done. Regarding the git workflow, we decided to use a feature branch workflow with required pull requests for merging to main.
The Kanban method does not require any roles, so all of us were developers. Every group member contributed equally. Anton Jürß was mainly responsible for integrating the service in the existing frontend. Anton extended the Web-Interface. Also, he extended the frontend server handlers and gRPC methods to provide the needed functionality to make requests to the review service. Tijana Milentijevic was in control of implementing the service functionality itself. Tijana connected the service to the database and provided the basic CRUD functionalities. Furthermore, Tijana integrated the functionality of the gRPC methods. Robin Sadeghpour was responsible for DevOps topics and the general engineering and infrastructure. Robin led design decisions, implemented the CI/CD pipelines, set up the Version Control System and also set up the Kubernetes Cluster and migrated the service to it. Everyone contributed to each part of the development, since most of these tasks were done in Pair-Programming, where one person is the "Driver" and the other person is the "Leader". The API Interface was designed as group. Further, everybody had to review pull requests. The requirements were specified in a meeting at the beginning of the development process and afterwards manifested in the form of Github Issues, which were then displayed as cards on the Github Project Kanban board. During the development process, some of the requirements were proven to be not well suited for

this project. Because of that, we thoroughly discussed alternatives in meetings and updated the requirements (Github Issues).

3 Technology Decisions

For this assignment, we decided to use Java as our programming language, since we are the most familiar with it compared to other programming languages. To manage the dependencies of our project and automate the builds, we applied the maven build automation tool. We used Spring Boot as Java framework for microservices. Spring Boot is open-source, lightweight and quick to set up. Spring Boot included Spring IoC Containers, allowing an easy implementation of the Inversion of Control (IoC) and Dependency Injection (DI) Principle. To integrate gRPC in our Spring Boot application, we applied gRPC-Spring-Boot-Starter Project which allows an easy integration of gRPC into Spring Boot by adding one dependency and one annotation. For testing concerns we used the JUnit Framework along with the Mockito Framework. Mockito simplifies the mocking of interfaces. Hence, it simplifies the use of test doubles. Also, we integrated Project Lombok in our service. This provides several annotations which can minimize the number of lines of code and also simplify the integration of different tools (e.g. Logger). In our case, we made use of the LOG4J Logger Library. Furthermore, user reviews had to be stored in a database. To do so, we used MongoDB Atlas which is a NoSQL database. It is free of charge, lightweight and easy to set up and to connect it to our Spring Boot application.

4 Quality Approach

We followed the IoC and DI Design Principles in our service. These design principles achieve loose coupling of classes. This results in enhanced maintainability, reusability, cohesion and testability of the code. This also complements well with Solitary Unit testing which is our approach for reliability. We implemented unit tests using JUnit. The loose coupling of classes which results from those principles allows us to easily mock interfaces in our unit tests and use test doubles with Mockito. By using a Continuous Integration Pipeline and requiring that all tests pass in combination with the approval of another developer for our pull requests, we ensured reliability and code quality of the service. Once the Continuous Integration Pipeline is successful and the commit is merged to main, we run one Pipeline for Continuous Deployment, which builds the Docker Image and pushes it to our Docker repository on the Google Cloud Platform. Then it deploys the Docker Image to the Kubernetes Cluster, which is running on the Google Kubernetes Engine. We also have another Docker Deployment Pipeline, that runs parallel to the other pipeline. It builds and pushes the Docker Image to a public Dockerhub Repository. This approach enhances the reliability aspect. To approach the observability aspect, we implemented Logging in our microservice. We catch and log errors that occur in our service to the console. Further, we log the gRPC service calls made to our service as information to the console.

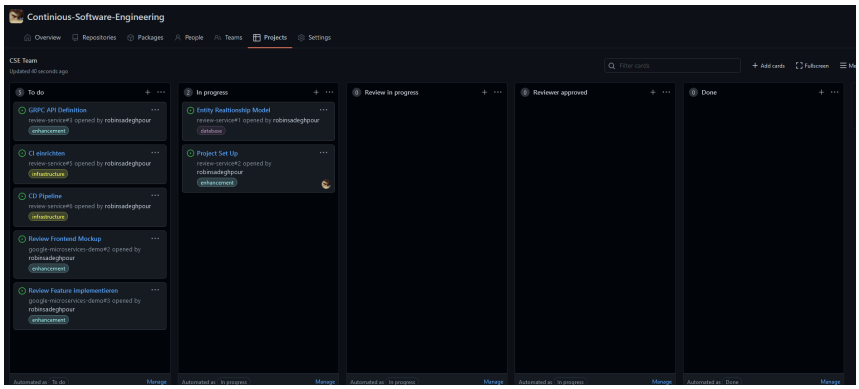

Fig. 1. The Github Project Kanban Board at the begin of the development

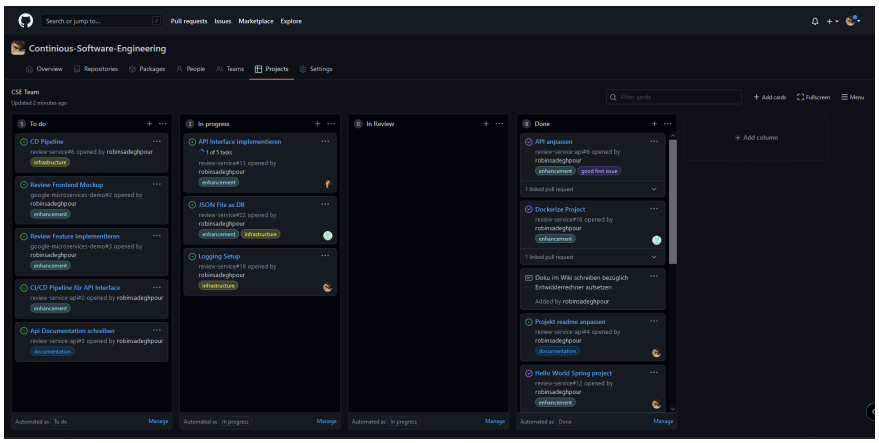

Fig. 2. The Github Project Kanban Board at the peak of the development

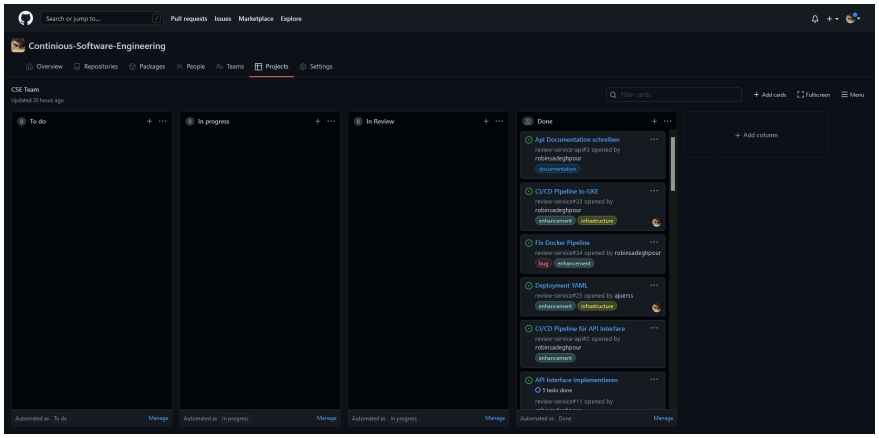

Fig. 3. The Github Project Kanban Board at the end of the development

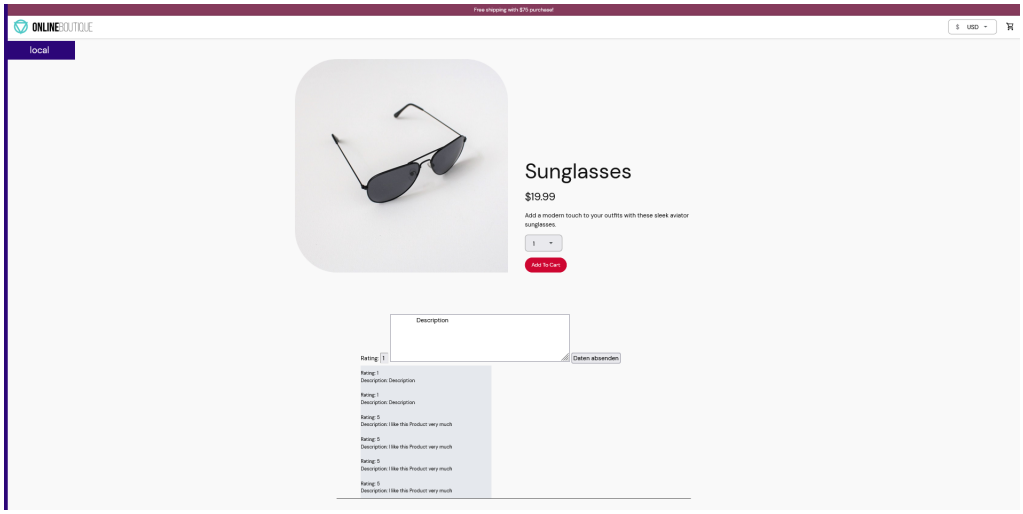

Fig. 4. The Review Feature in the online boutique frontend

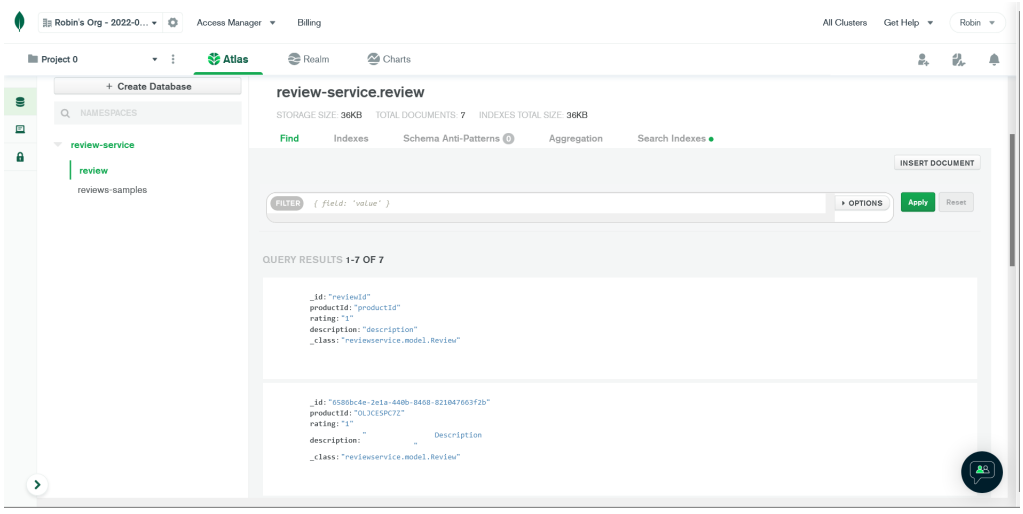

Fig. 5. Database records

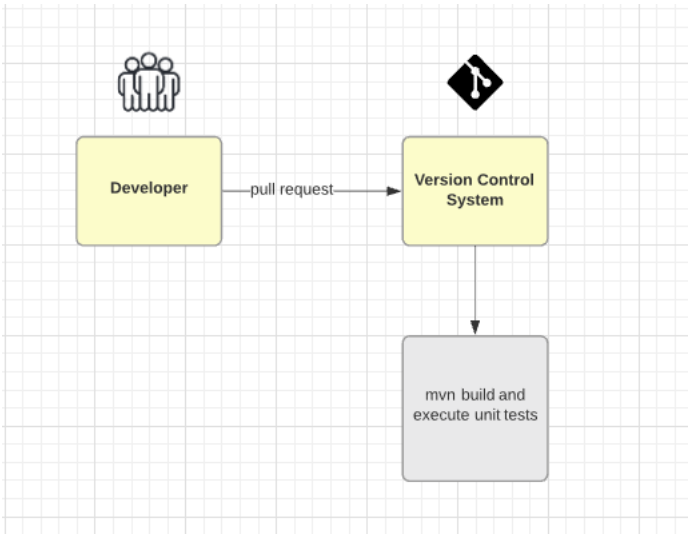

Fig. 6. Continuous Integration

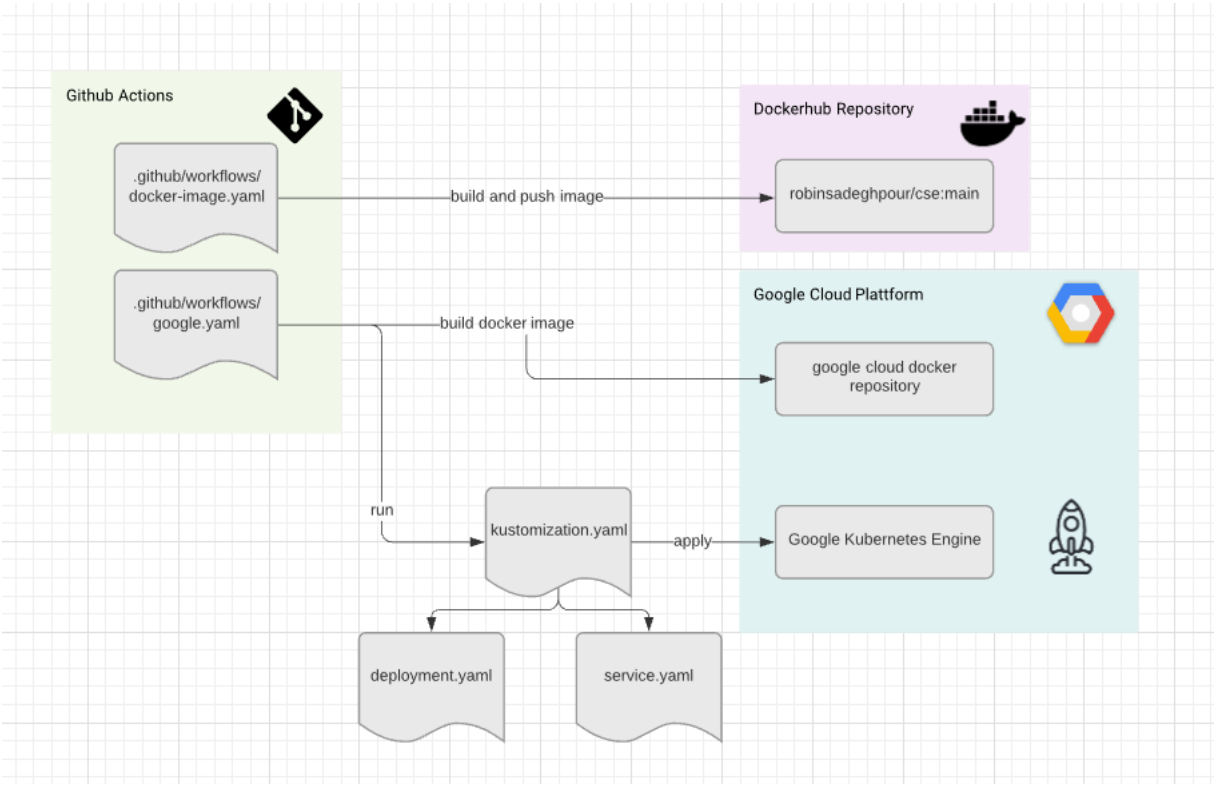

Fig. 7. Continuous Deployment