Group 26 - Software Design Specification

Christoper Nixon
Davy Nolan
Hannah Mahon
Christopher Staunton
John O'Doherty
Paul O'Callaghan

Table of Contents:

---

1. Introduction

1.1. Overview - Purpose of system

DevOps is a term to describe the collaboration between systems development and systems operations. It gives an overall perspective of the whole life-cycle of developing and producing code through clearly defined steps, tools and automation. IBM wants us to build a CI/CD pipeline to incorporate this methodology. The main purpose of this pipeline is to help automate your software delivery. This automation includes running automated tests, the standardisation of software and finally deploying the code.

1.2. Scope

To build this CI/CD pipeline, we must develop:

- A front-end such as a web-app to test that the code was properly deployed
- Automated testing and standards
- Automated deployment

1.3. Definitions, abbreviations

- **CI/CD** : Continuous Integration/Continuous Delivery.
- **CI/CD pipeline**: This helps to automate steps in a software delivery process, such as initiating code builds, running automated tests, and deploying to a staging or production environment. Automated pipelines remove manual errors, provide standardized development feedback loops and enable fast product iterations.
- **Standardisation**: This is a common format of a document or file that is used by one or more software developers while working on the same computer program. Software standards allow for the interoperability between different programs created by different developers.

1.4. References

Travis CI documentation: https://docs.travis-ci.com/

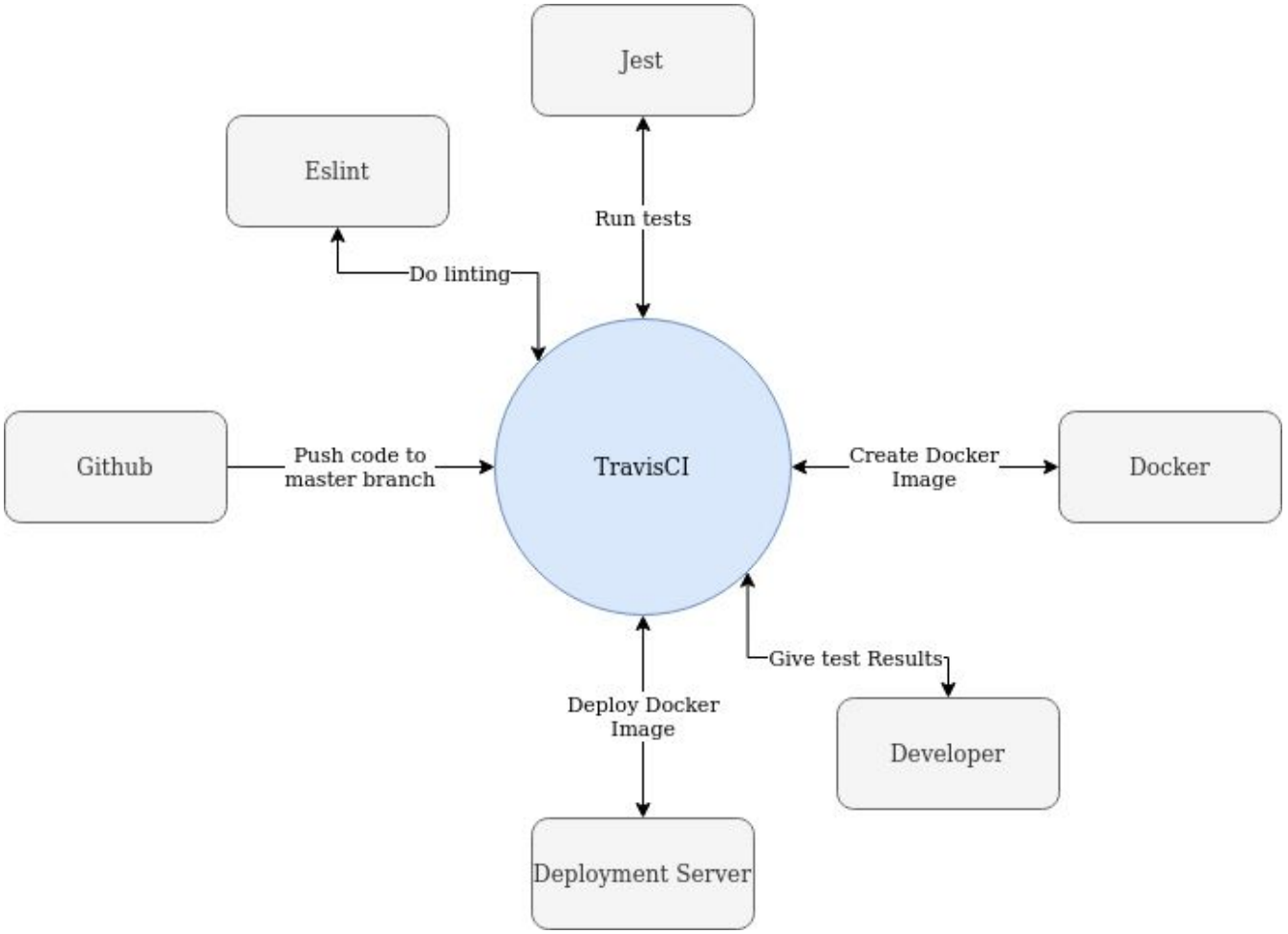
2. System Design

2.1. Design Overview

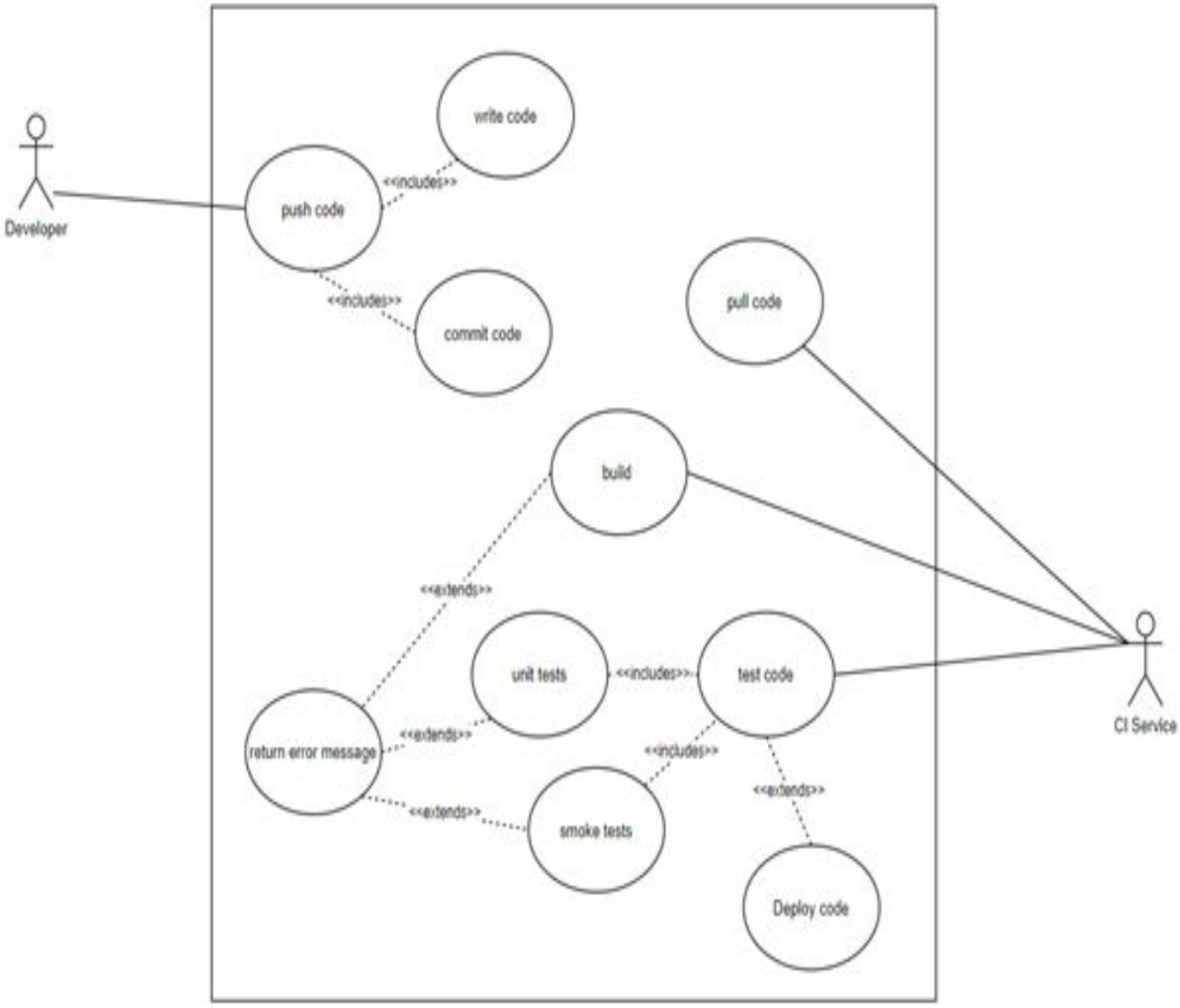
A complete CI/CD pipeline and a simple application to demonstrate the use of this pipeline. The simple application in question will be a reactJS app with an Express backend. This will be hosted using Heroku. Version-control will be handled with Github and changes to the master branch will trigger an automated pipeline run using Travis CI.

2.2. System Design Models

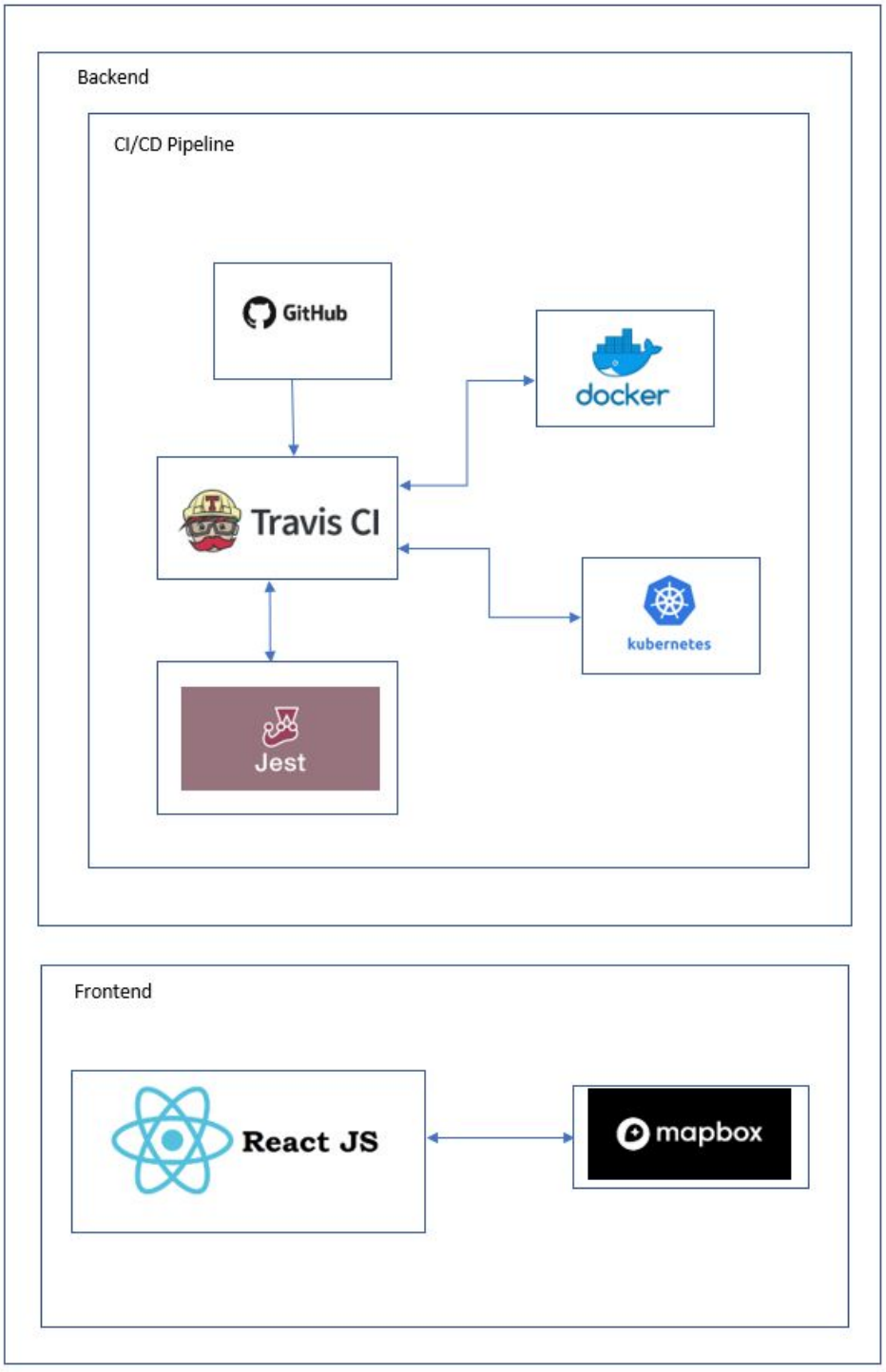
2.2.1.System Context



2.2.2.Use cases (from Requirements)



2.2.3.System Architecture



The diagram above is our system architecture, it shows the interaction between components of our system. Our source code is written in javascript with react related libraries, which is held in a git repository on GitHub. Then TravisCI takes the code and enters the pipeline, where it is built and tested. The tests are written by our team, using react testing library and Jest. The successfully built and tested code is then containerised with Docker. Finally the docker image of our application can be deployed in many different cloud services.

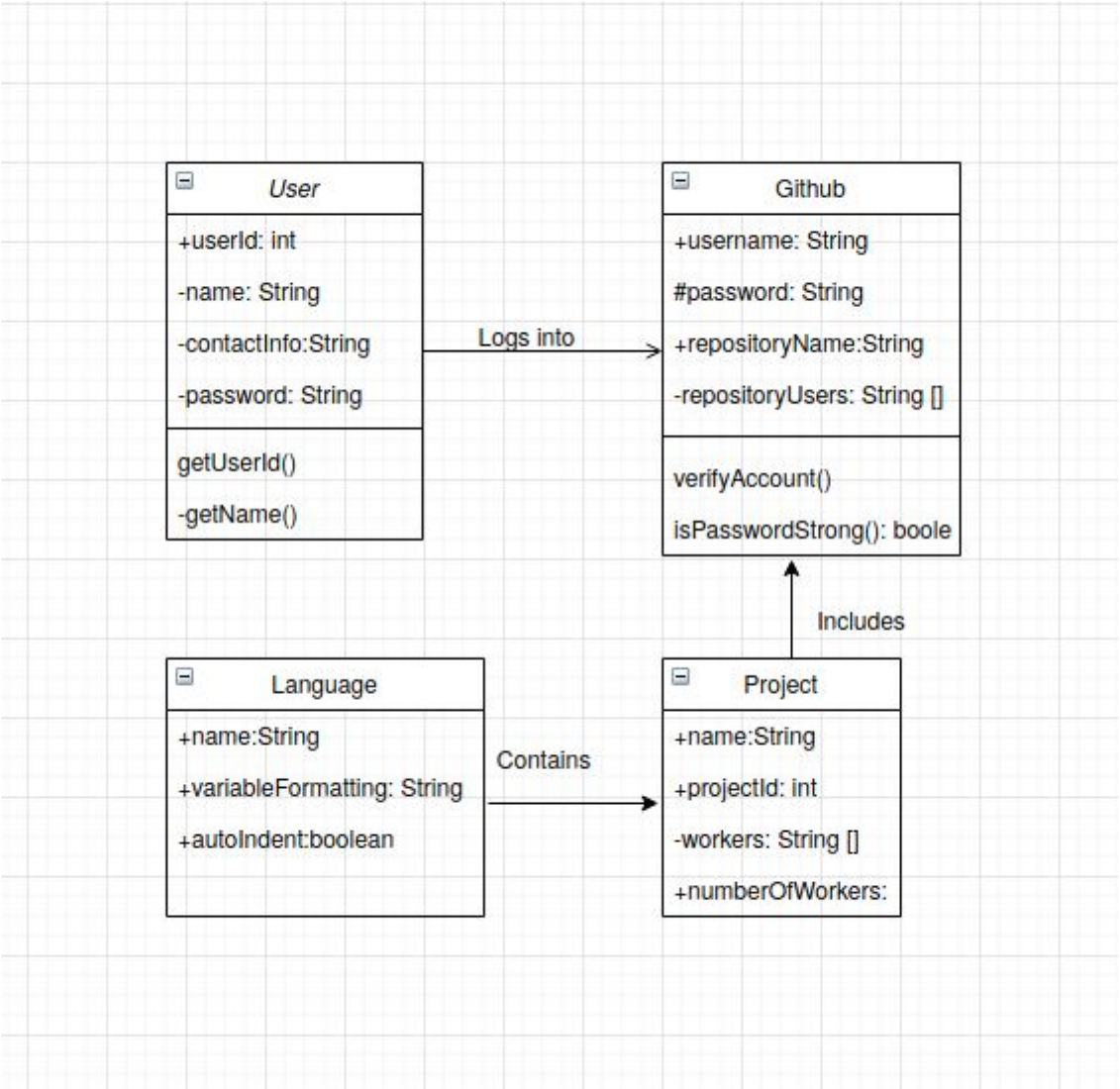
Architectural Decisions:

Architectural Decision:	Use Travis CI as the main Continuous Integration platform.
Issue or Problem Statement:	There are several different CI solutions to choose from
Assumptions:	The code is placed in a version control repository, such as github
Motivation:	Support the Continuous Integration of the application.
Alternatives:	Jenkins, GitLab
Justification:	We chose TravisCI because it is cloud based and easy to run. It is ideal for our team project as it is quick to set up.
References:	https://docs.travis-ci.com/

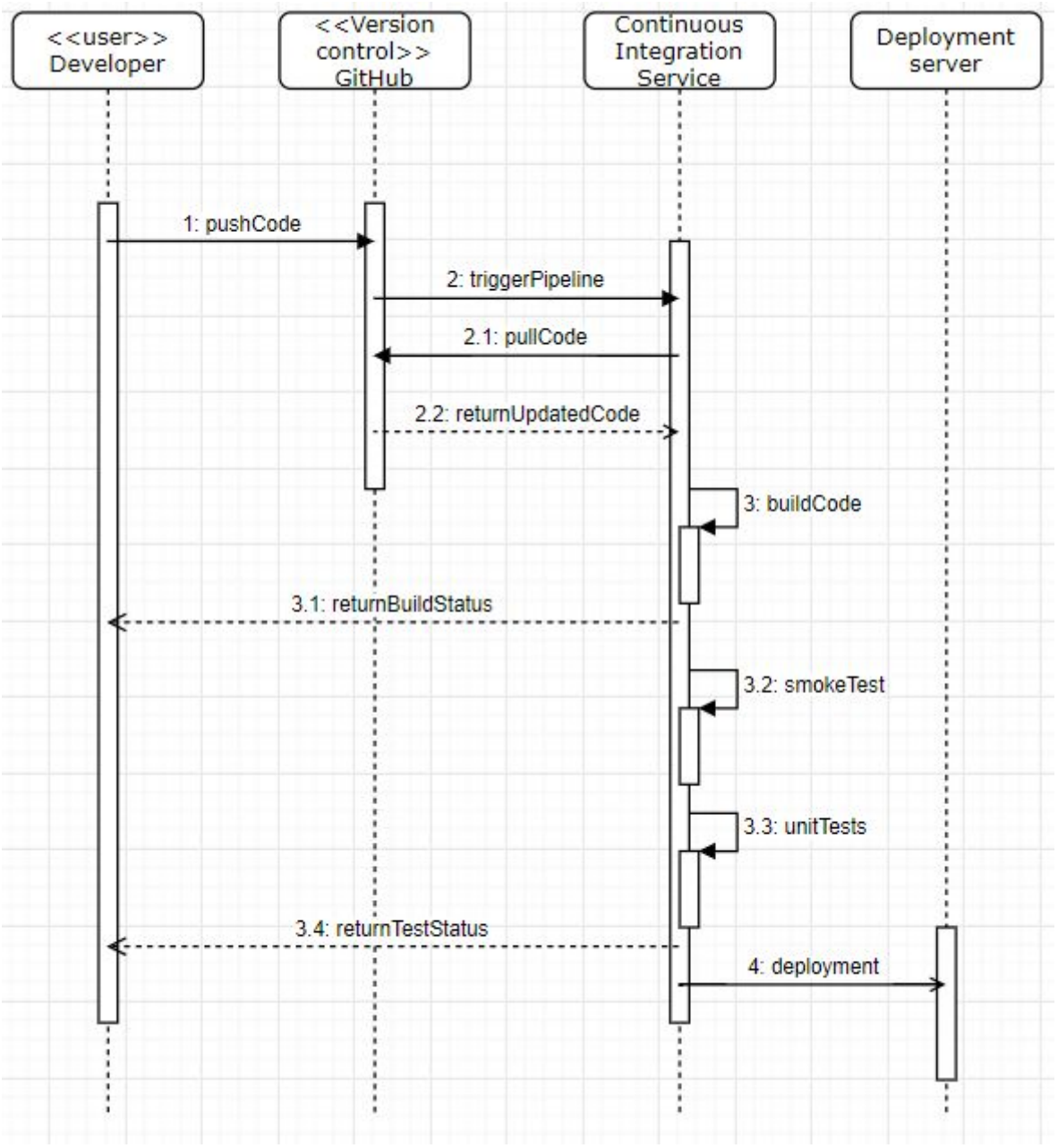
Architectural Decision:	Use GitHub as the main Version-control platform.
Issue or Problem Statement:	There are several different CI solutions to choose from
Motivation:	Support the Continuous Integration of the application.
Alternatives:	TortoiseSVN, Apache Subversion
Justification:	We chose GitHub because it is accessible, open source and can be linked with TravisCI easily.
References:	https://github.com/

Architectural Decision:	Use Docker for Containerisation
Issue or Problem Statement:	We need a way to continuously deploy our application. There are different containerisation platforms.
Motivation:	Containerizing our software allows us to deploy it easily to pretty much any cloud vendor.
Alternatives:	Kubernetes
Justification:	We chose docker for containerization because it is the most popular open-source containerization software available.
References:	https://docs.docker.com/

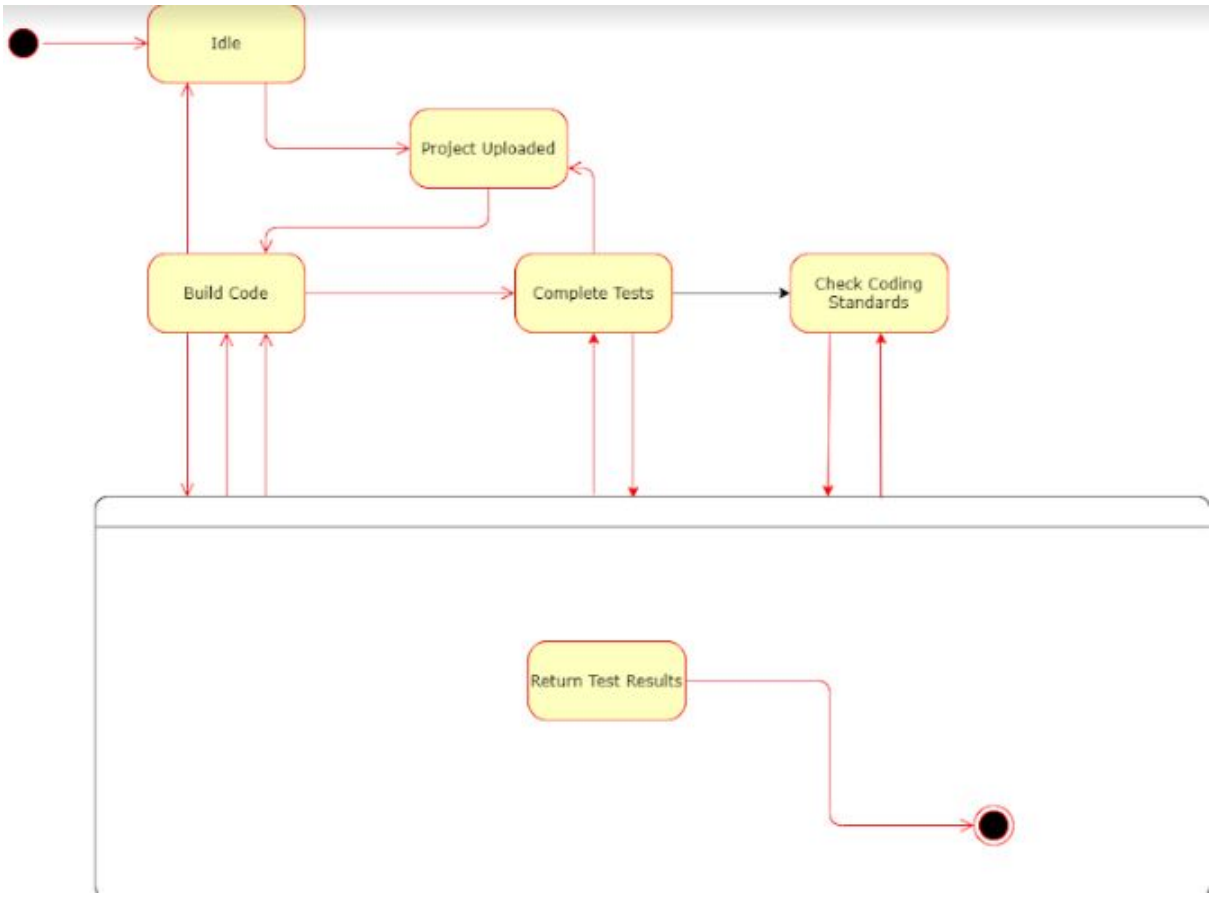
2.2.4.Class Diagrams



2.2.5.Sequence Diagrams



2.2.6.State Diagrams



2.2.7.Other relevant models:

Dynamic Model: