Group 26 Project Plan

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

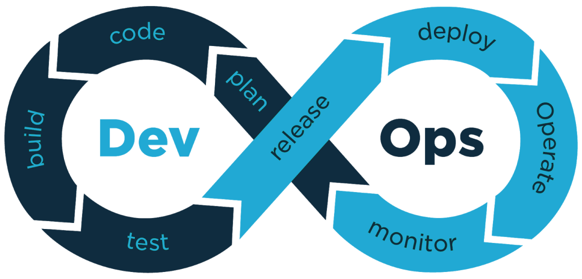


Table of Contents

1. Project Goals and Objectives

Background:

Our client is IBM and our project is to build a continuous integration and continuous deployment pipeline to help automate software delivery. The pipeline will include automating testing, standardisation, packaging of the code and finally deployment to both a development and production environment. The project will also have a front-end to show that the code has been deployed properly.

Objectives:

The term “DevOps” has been used to describe what our project aims to do. DevOps is a term used to illustrate the integration of software development and operational procedures. Having a Continuous Integration/Continuous Deployment (CI/CD) pipeline allows for merging of code development and the operational procedures of deploying . The pipeline will be able to build, test and deploy applications with speed and quality which means that the manual steps such as testing and indenting will be removed. This will ensure high quality software releases and allow for a faster release cycle. DevOps is also aligned with agile methodology which means that it is an incremental process with room to go back to a previous step and test each step as we go instead of testing all of the code at the end. Again, this means that errors are reduced and the project has more chances of success. Our client left the decisions of what technology stack to use up to us as they wanted to see what we came up with, having familiarised ourselves with the concept of a CI/CD pipeline. They may use this project as an example of the benefits of using such a pipeline to other teams within IBM.

Goals:

The goals for our group project are:

- A working CI/CD pipeline that automatically tests and deploys the code.
    - A Build stage, where changes to a shared source code repository trigger the start of the pipeline, a new build of the project.
    - An Analyse stage, where standardisation of the code will be performed, as well as static analysis of the code, such as unit tests.
    - A Bake stage, where the code will be packaged in some way that allows for easy deployment.
    - A Develop Deploy stage, where the packaged project will be deployed to a development environment.
    - A Test stage, where end-to-end testing of the application will be performed, for example testing the UI of the application.
    - A Production Deploy stage, where the packaged project will be deployed to a production environment.
- A working front-end app which will be a simple map-based quiz game.
- Deliver understandable, well-written code that is easily scalable.
- Clear documentation of architectural decisions, and documentation for how Users, Developers, and Application Administrators would use the application/ pipeline.

1. Project Scope

Project Deliverables:

The deliverables we aim to have for the project are; a code bundle including a pipeline and a front-end web app. The pipeline should be easily scalable and the front-end will be a game where the user will have a set of questions about different regions of the world. Once they guess the answer, the correct answer will be marked on the map. This front-end is not our main focus, it is just to show that the pipeline has correctly deployed the code.

We also aim to have the following documentation;

- Requirements document
- Project Plan
- Software Design Specification
- Management Report
- Developmental Report
- A ReadMe file on our github
- Architectural report, which was asked for by IBM to detail our architectural decisions
- A User guide detailing how an end-user would use the application.
- A Developer guide, detailing how a developer would add code to the application.
- A Application Administrator, detailing how an application administrator would manage and upgrade the application.

Project Boundaries:

In Scope:

- A shared code repository where merging to a branch will trigger the CI/CD pipeline
- Build stage of the pipeline, where the project will be built
- Analysis stage of pipeline where the code will undergo static testing and code coverage tests
- A deploy stage of the pipeline
- A shared product backlog of prioritised user stories.

Out of Scope:

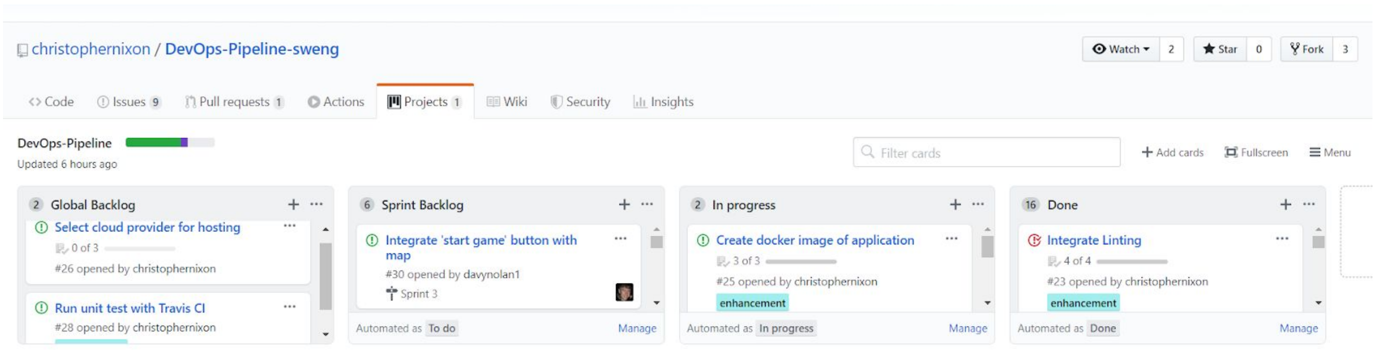
- A front end web application that shows that the code has been properly deployed

Project Backlog:

We have implemented our Project Backlog as a Github Project, which can be found at our public Github Repository here:

https://github.com/christophernixon/DevOps-Pipeline-sweng/projects/1.

Here is a screenshot of what it currently looks like, the backlog can also be seen in our Requirements document and Software Design Specification outline which are included in Appendix One and Two respectively.



christophernixon / DevOps-Pipeline-sweng

Watch 2 | Star 0 | Fork 3

Code | Issues 9 | Pull requests 1 | Actions | Projects 1 | Wiki | Security | Insights

DevOps-Pipeline
Updated 6 hours ago

Filter cards | Add cards | Fullscreen | Menu

2 Global Backlog
- Select cloud provider for hosting — 0 of 3 — #26 opened by christophernixon
- Run unit test with Travis CI — #28 opened by christophernixon

6 Sprint Backlog
- Integrate 'start game' button with map — #30 opened by davynolan1 — Sprint 3

Automated as To do — Manage

2 In progress
- Create docker image of application — 3 of 3 — #25 opened by christophernixon — enhancement

Automated as In progress — Manage

16 Done
- Integrate Linting — 4 of 4 — #23 opened by christophernixon — enhancement

Automated as Done — Manage

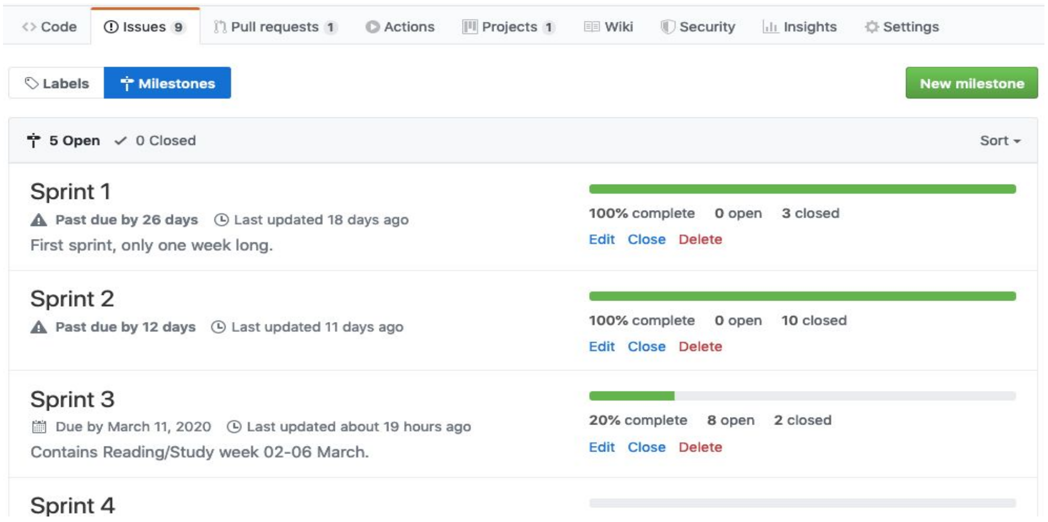
2. Project Approach

Our aim for our project is to build a working pipeline that tests and deploys the code to a front-end web application. We have decided to use Travis CI which is a continuous integration and continuous deployment service that works alongside Github. For the packaging of our application, we decided to put it into a Docker Container. We will then deploy this container to IBM cloud, in a Kubernetes Cluster. For our front-end, we are creating a ReactJS application using Create-React-App. Our front-end consists of a game which is based around the map of the world. The user can answer a set of questions with the answer always being a location. Once they take a guess, the answer will be revealed by a marker on the map of the location.
Our approach to this project is putting our main focus on the pipeline and the testing of the code as this was given to us in the brief and is what our client is expecting from us. Once we have a skeleton of the pipeline working, we will work on the front-end making this look aesthetically pleasing however its main purpose is to show that the code has been properly deployed.

Scrum Sprints:

Our project is broken down into two week sprints, so over the 11 weeks of the semester we will have roughly 6 sprints. In our github repository we have added Milestones for each sprint, allowing us to assign github Issues at the start of each sprint and see the progress we have made for each sprint. These milestones, along with a screenshot of what they currently look like can be seen below.

https://github.com/christophernixon/DevOps-Pipeline-sweng/milestones



5 Open ✓ 0 Closed

Sprint 1
Past due by 26 days · Last updated 18 days ago
First sprint, only one week long.
100% complete 0 open 3 closed

Sprint 2
Past due by 12 days · Last updated 11 days ago
100% complete 0 open 10 closed

Sprint 3
Due by March 11, 2020 · Last updated about 19 hours ago
Contains Reading/Study week 02-06 March.
20% complete 8 open 2 closed

Sprint 4

Sprint Date:	**Objectives:**	**Dates Set:**
Sprint 1: 19th - 31st Jan	● Meet as a team for first time ● Pick Project, Email Macu about project choice ● Receive Project ● Meet Client	● 31st Jan- First Client Meeting ● 27th Jan - Sprint Planning ● Scrum meetings: ○ Tuesdays 2-3pm ○ Thursday: 12-1pm
Sprint 2: 3rd-14th Feb	● Appoint Roles within team ● Research Technology ● Requirements doc + presentation	● 14th Feb - Second Client Meeting with sprint review
Sprint 3: 17th - 28th Feb	● Technologies picked ● Skeleton of front-end ● Travis CI up and running deploying code ● Quiz questions decided	● 28th Feb - Third Client meeting with sprint review
Sprint 4: 2nd-13th March	● Project Plan and Software design software design reports done ● Architecture report done for client ● Docker containers working ● Linting working ● Skeleton of static tests	● 3rd March meeting - Sprint backlog review after reading week ● 13th March - Fourth Client meeting with sprint review
Sprint 5: 16th-27th March	● A working pipeline ● Static tests and code coverage working ● Code being properly deployed according to standards set ● Front-End map game working fully	● 27th March - Fifth client meeting with sprint review

Sprint 6: 30th Mar - 10th April	● Project working entirely - working CI/CD pipeline with a front-end quiz being properly deployed ● Final Presentation and project demo ready ● Management/ Development reports finished ● Client handover and approval	● April - final Client meeting with sprint retrospective

3. Project Organisation

Staff:

Name:	Prior Technical Skills	Prior Projects
Christoper Nixon	Natural Language Processing, Using Cloud Services: AWS and IBM. Languages: Python, Java, SQL, Bash, C, Haskell, Prolog, Assembly.	• Traffik Analysis Hub with IBM. • Personal project for analysis Whatsapp chat data: https://github.com/christophernixon/sugardaddy
Davy Nolan	Python, Java, C, SQL, React, Haskell, Assembly, Bash, Javascript, Natural Language Processing, IBM Cloud Services, Prolog.	• Traffik Analysis Hub with IBM (2019 programming project + summer internship).
Hannah Mahon	React, Java, C, Sql, React Native, Haskell, Javascript, Android Studio, Assembly	• Built an app in react native for programming project in second year
Christoper Staunton	Java, C, Assembly	N/A
John O'Doherty	Java, C, Assembly	N/A
Paul O'Callaghan	Java, C, Assembly	N/A

Staff Chart:

Our client Mihai Criveti from IBM has requested that he be the product owner for the project. The staff chart is broken down as the following:

Sprint Date:	Roles Appointed:	Work appointed:
Sprint 1: 19th - 31st Jan	● Product Owner: IBM ● Scrum Master: Davy Nolan ● Team Members: ○ Chris N ○ Hannah ○ Chris S ○ Paul ○ John	● Team effort: Pick project ● Hannah: email Macu project preference ● Davy: Set up client meeting ● Chris N: Set up communication network
Sprint 2: 3rd-14th Feb	● Product Owner: IBM ● Scrum Master: Hannah Mahon ● Team Members: ○ Chris N ○ Davy ○ Chris S ○ Paul ○ John	● Team effort: appoint roles within team, requirements doc + presentation, research technologies ● Hannah: handover requirements doc to client ● Chris N : Pick technologies ● Davy: set up second meeting with IBM
Sprint 3: 17th - 28th Feb	● Product Owner: IBM ● Scrum Master: Chris Nixon ● Team Members: ○ Davy ○ Hannah ○ Chris S ○ Paul ○ John	● Chris N: travis CI up and running ● Hannah: mapbox api and react.js working together ● Chris S: technology research written out ● Paul: working on a text doc for the front-end ● John: Quiz questions written out ● Davy: Skeleton of front-end,e.g buttons

Sprint	Roles Appointed	Work Appointed
Sprint 4: 2nd-13th March	● Product Owner: IBM ● Scrum Master: Davy Nolan ● Team Members: ○ Chris N ○ Hannah ○ Chris S ○ Paul ○ John	● Team Effort: software design spec, architecture report for client ● Hannah: Project Plan ● Davy: Set up docker image and fix button. ● Chris N: docker containers up and running ● John: Linting working ● Paul: Changing where our map is centered. ● Chris S: Creating first unit tests
Sprint 5: 16th-27th March	● Project Owner: IBM ● Scrum Master: Hannah Mahon ● Team Members: ○ Chris N ○ Davy ○ Chris S ○ Paul ○ John	● Team effort: working pipeline ● Hannah: Presentation of build logs to developers ● Davy: Production deployment environment ● Chris N: Development deployment environment ● John: User guide ● Paul: Developer Guide and Application Administrator guide ● Chris S: Finishing off testing stage of pipeline
Sprint 6: 30th Mar - 10th April	● Project Owner: IBM ● Scrum Master: Chris Nixon ● Team Members: ○ Hannah ○ Davy ○ Chris S ○ Paul ○ John	● Third Years: final presentation & Management report ● Second years: developmental report

4. Risk Analysis

Risk Element	Impact (1-5)	Likelihood (1-5)	Risk Factor(I *L)
Lack of experience with technologies - React, Travis CI, Docker	5	3	15
Unable to complete non functional requirements due to time restrictions	1	3	3
Difficulty integrating static testing and providing code coverage	5	2	10

Risk Mitigation

Risk:	Measures to reduce risk:
Lack of experience with new technologies	We will dedicate time at the beginning of the sprint to researching technologies and learning how to use them by watching youtube tutorials etc. With react, the third years will help the second years as we've had experience with it before
Unable to complete non functional requirements due to time restrictions	We will stick to our sprint schedule as best we can to manage our time properly and have enough time at the end to complete the non functional requirements
Difficulty with integrating static tests and providing code coverage	We will appoint at least one second year and one third year to this job to ensure that the static tests will be integrated properly and the code coverage is present

5. Project Controls

We will use the following tools to ensure our project runs smoothly:

- **Github**: We will be using github for our code repository. Our client has access to this so that he can follow our progress throughout the project.
- **Github Projects**: This is a design feature on github to appoint tasks to group members and to see what is to be done and what has been done. This tool will allow us to keep track of what each team member has to do by the end of each sprint and will make sure we keep on top of the tasks that we have set at the start of the sprint.
- **Slack**: This is a messaging platform in which we can communicate with our team, client and demonstrator from the one group chat. This is important to set up meetings with both our client and demonstrator.
- **Google Drive**: This is where all our reports and meeting minutes can be shared amongst each other and are saved on the drive to ensure our documents are saved properly and can be accessed by each individual group member
- **Facebook Messenger**: This is the easiest form of communication amongst just the team members so we can organise meetings, discuss what has been done and remind each other of deadlines on a day-to-day basis.
- **Our sprint schedule**: This is an important schedule which features in this report to make sure we are keeping track of our time and using it wisely and also to make sure at the end of each sprint we have done what is outlined.
- **Cisco Webex**: This is a video calling web application that lets us communicate with our client and have meetings at the end of each sprint to review what has been done and what we have left to do for the following sprints.

6. Communication

Client Communication:

Date:	**Purpose:**
31st January	● Initial first meeting to establish requirements and introduce the team
14th February	● Second Meeting - sprint 1 review and requirements doc sign off
28th February	● Third Meeting - Sprint 2 review
13th March	● Fourth Meeting - Sprint 3 review, project plan and architecture diagram review
27th March	● Fifth Meeting - Sprint 4 review
10th April	● Sixth Meeting - Sprint retrospective and project sign off

Project Team Meetings:

Date:	**Purpose:**
22nd Jan	First Initial meeting - pick project and meet team members
28th Jan	Assign roles, discuss project given
4th Feb	Requirements, technologies discussed
11th Feb	Requirements Presentation practice
18th Feb	First stage of implementation: jobs given

25th Feb	Assign work for reading week
10th March	Review work done over reading week, project plan discussed
19th March	First stage of implementation done - skeleton of both pipeline and front end
24th March	Second Stage of implementation started- working pipeline with testing
31st March	Front end finished, working pipeline deploying code and final presentation started
2nd April	Non-functional requirements started
7th April	Final reports, presentation and code handover done

7. Appendice

Appendix One: Requirements Document

Group 26 Requirements Document

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

Table of Contents:

Introduction

1.1 Overview:

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 Objectives and Success Criteria

The success criteria are to have:
- A working pipeline that automatically build and tests the code
- The pipeline also must ensure the code adheres to certain coding standards
- Once the code pass each of the above steps, it is then deployed
- A working front-end web-app that can show the code was deployed properly.
- The pipeline must work smoothly and in a timely fashion

1.4 Definitions and 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.

1.5 References:

Travis CI documentation: https://docs.travis-ci.com/
Smarty Pins UI:
https://www.learningliftoff.com/smarty-pins-trivia-makes-google-maps-educational/
http://googlesystem.blogspot.com/2014/07/google-maps-trivia-game-smarty-pins.html

Proposed System

3.1 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.

3.2 Functional Requirements:

1) A clear, documented process for developers to add code to a shared code repository.
2) Merging to a branch in the code repository will trigger a CI/CD pipeline.
3) The simple application must be written with certain code standards. The pipeline triggered by changes to the Github repository will include quality checking as a stage in the pipeline.
4) There will be a build stage in the pipeline, where an attempt will be made to build the application.
5) There will be an analysis stage in the pipeline which will include running static analysis / code coverage tools and standardisation of the code.
6) There will also be a deploy stage in the pipeline, where an attempt is made to deploy the application.
7) A testing stage of the pipeline, where the end-to-end application is tested. This may for example include UI testing.
8) Moving from stage to stage of the pipeline needs to be automatic and conditional on the successful passing of the previous stage.
9) Each member of the team needs to be able to add a feature to the application and trigger the entire pipeline.
10) A shared Product Backlog of prioritized user stories.
11) Online documentation relevant to the roles of User (user guide), Developer (contributor guide) and Application Administrator (manage and upgrade guide).

3.3 Non-Functional Requirements:

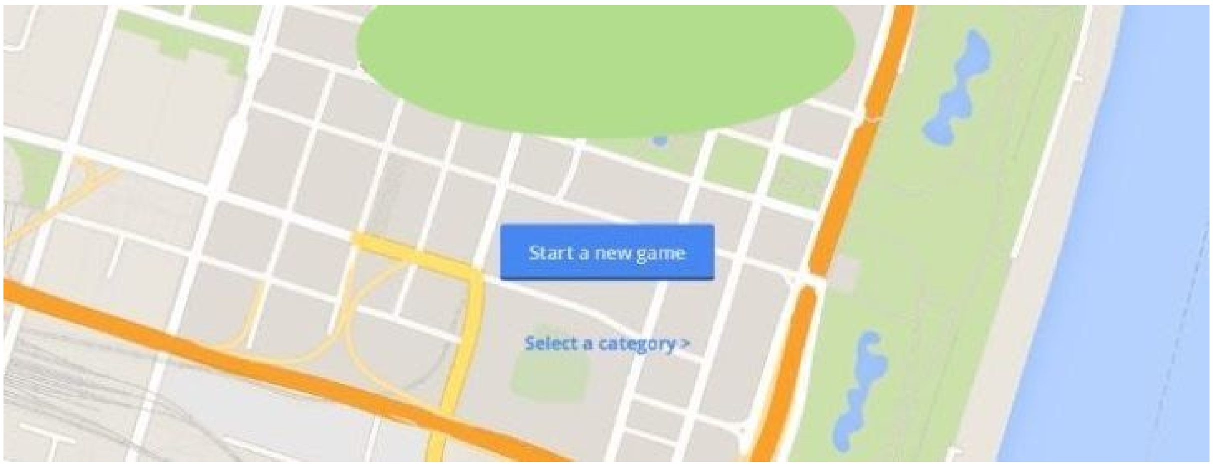
1) Integrating notifications from the pipeline with some sort of messaging, for example Slack.
2) Augment the pipeline with a local stage, by using Git Hooks to enforce coding standards before any code can be added to Github.
3) Add a test deployment environment, so that end-to-end testing can happen before the application is fully deployed.

3.4 System Prototype (models):

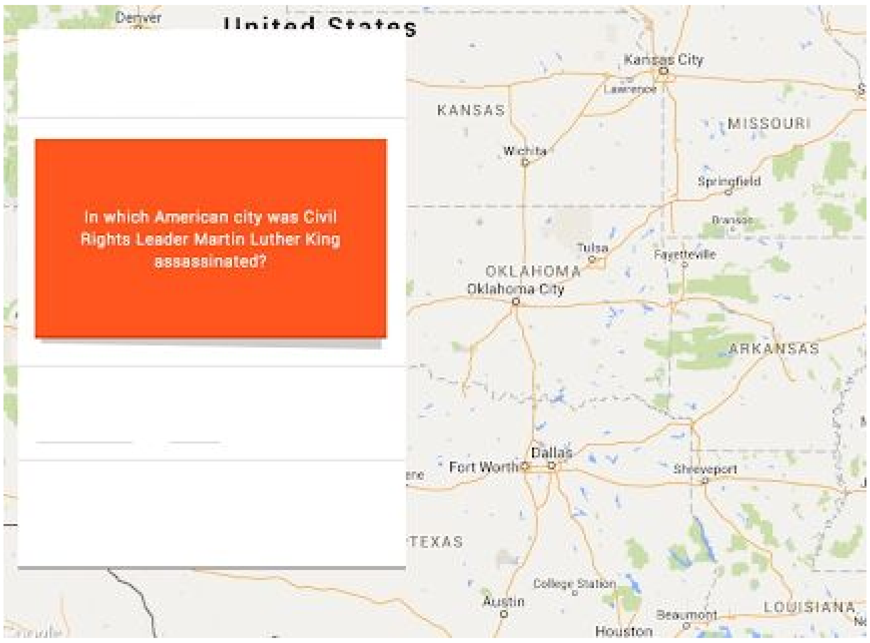
3.4.1 User Interface Mock-ups:

The focus of our project is creating a Ci/CD pipeline, which will not have a User Interface. All of our interactions with it will be through the tools which it consists of, so in our case with Github, Travis CI and Heroku.
However, the ReactJS app that we are going to build will be a simple location-based quiz game. This will have a User Interface, because it is a useful and graphical way to demonstrate the end-product of the CI/CD pipeline. The design for our UI will be inspired by a trivia game developed by Google in 2014 called Smarty Pins.

- The landing page will consist of a large map, a title, and a button for starting a new game/quiz.

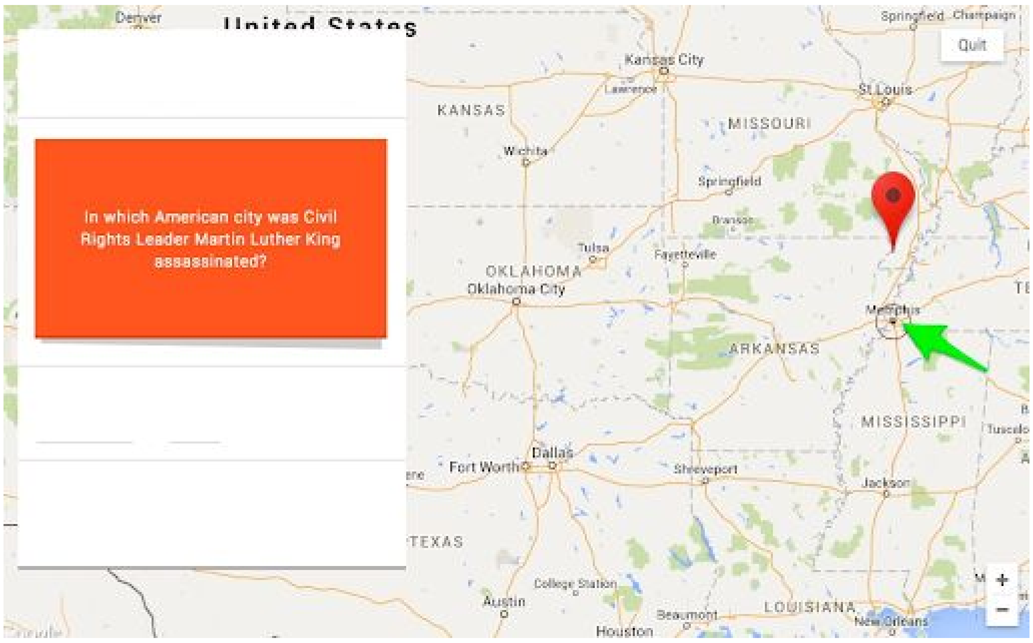


- Upon starting a new game, the user will be presented with a question in a pop-up box.



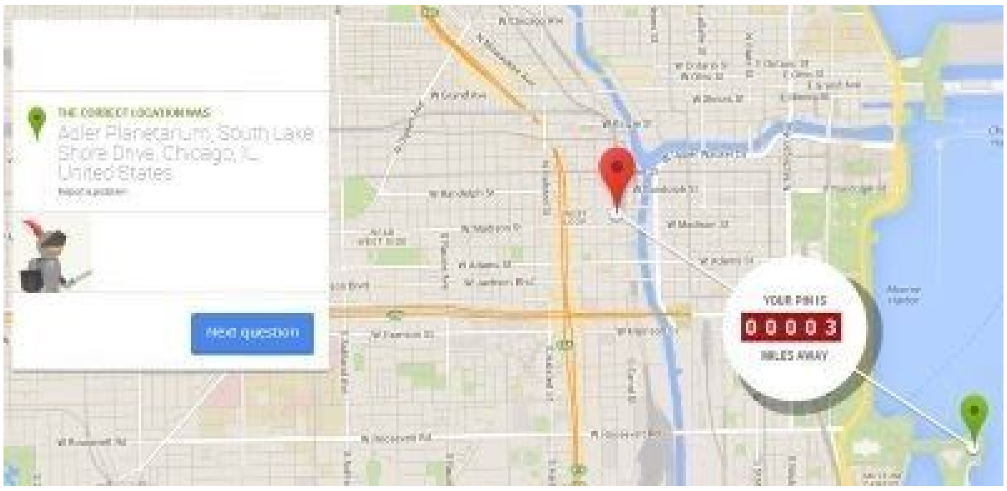
In which American city was Civil Rights Leader Martin Luther King assassinated?

- The user will then be able to navigate to the place on the world map where they think the answer is (all questions will have a location as an answer) and drop a pin.

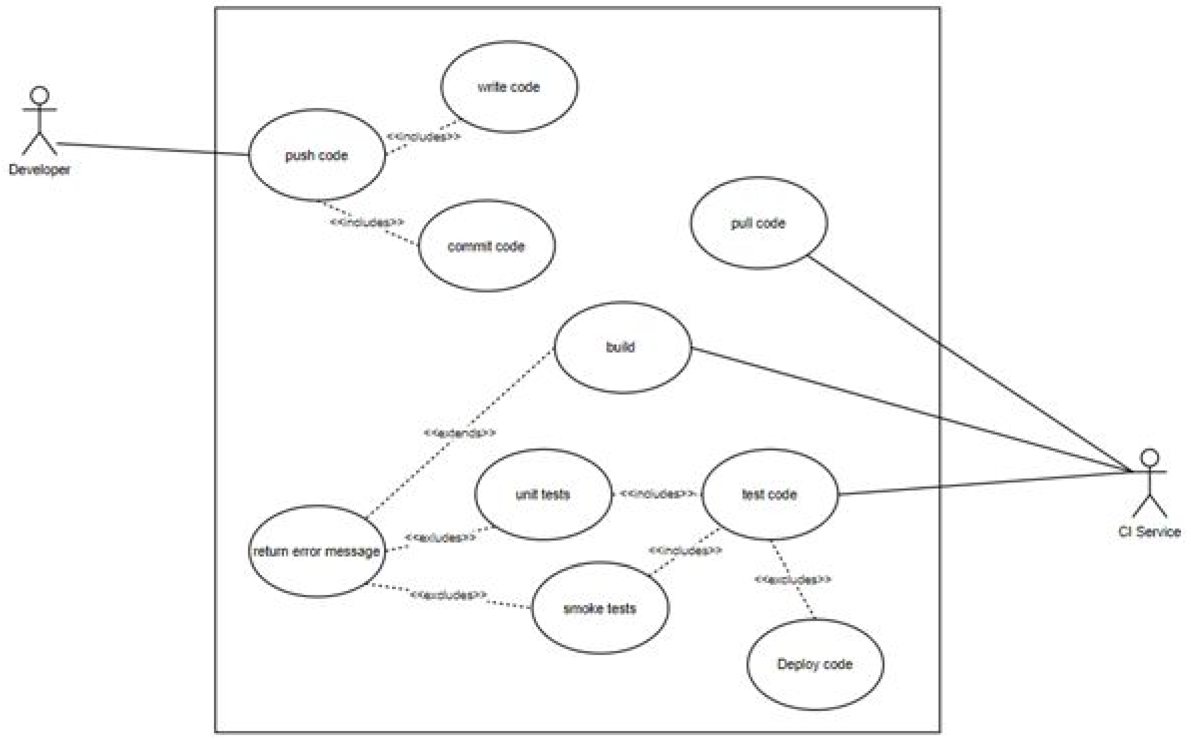


In which American city was Civil Rights Leader Martin Luther King assassinated?

- The user will then be informed how far away from the correct location they were in km.


THE CORRECT LOCATION WAS:
Adler Planetarium, South Lake Shore Drive, Chicago, IL, United States
Next question
YOUR PIN IS
0 0 0 0 3
MILES AWAY

3.4.2 Use Cases (including text narratives):



Name on diagram: Push code

Actors: Developer

Entry condition: The developer must have written their code and commit it sufficiently. Then the developer can push the code to the code repository.

Exit condition: The developer has successfully pushed the code.

Normal Scenario:

- the developer must write his/her code.
- developers must commit the code they want to add.
- then the developer will push the code onto the code repository.

Error Scenario:

- The developer forgets to commit their code.

Name on diagram: Pull code

Actors: Continuous Integration Service (TravisCI)

Entry condition: The code repository has been updated/modified.

Exit condition: The code has been pulled.

Normal Scenario:

- The code on the repository is changed by a developer.
- CI service detects this and pulls the code, to be entered in the pipeline.

Error Scenario:

- The CI service has an invalid link to the repository. No code can be pulled.

Name on diagram: build code

Actors: CI Service

Entry condition: Code has entered the pipeline.

Exit condition: Code has been successfully built. Or code build fails.

Normal Scenario:

- Developer pushes code to the repository.
- CI Service pulls code from repository and enters it into the pipeline.
- Then the code is compiled/built into a runnable form.

Error Scenario:

- code enters the pipeline.
- code cannot be appropriately compiled.
- error is flagged, and users are notified.
- The job within the pipeline is aborted.

Name on diagram: Test Code

Actors: CI Service

Entry condition: Code has passed the build stage successfully.

Exit condition: Testing of the code returns no errors.

Normal Scenario:

- code enters the pipeline.
- code is built successfully.
- code is tested with smoke tests to catch large bugs.
- code is tested with unit tests to potentially catch smaller specific bugs.
- code passes the test stage and moves onto the deploy stage.

Error Scenario:

- code enters the testing stage.
- the unit testing catches a small bug.
- users/developers are notified, and the job is ended.

Name on diagram: Deploy code

Actors: CI Service

Entry condition: The code has passed the testing stage.

Exit condition: The code is successfully deployed.

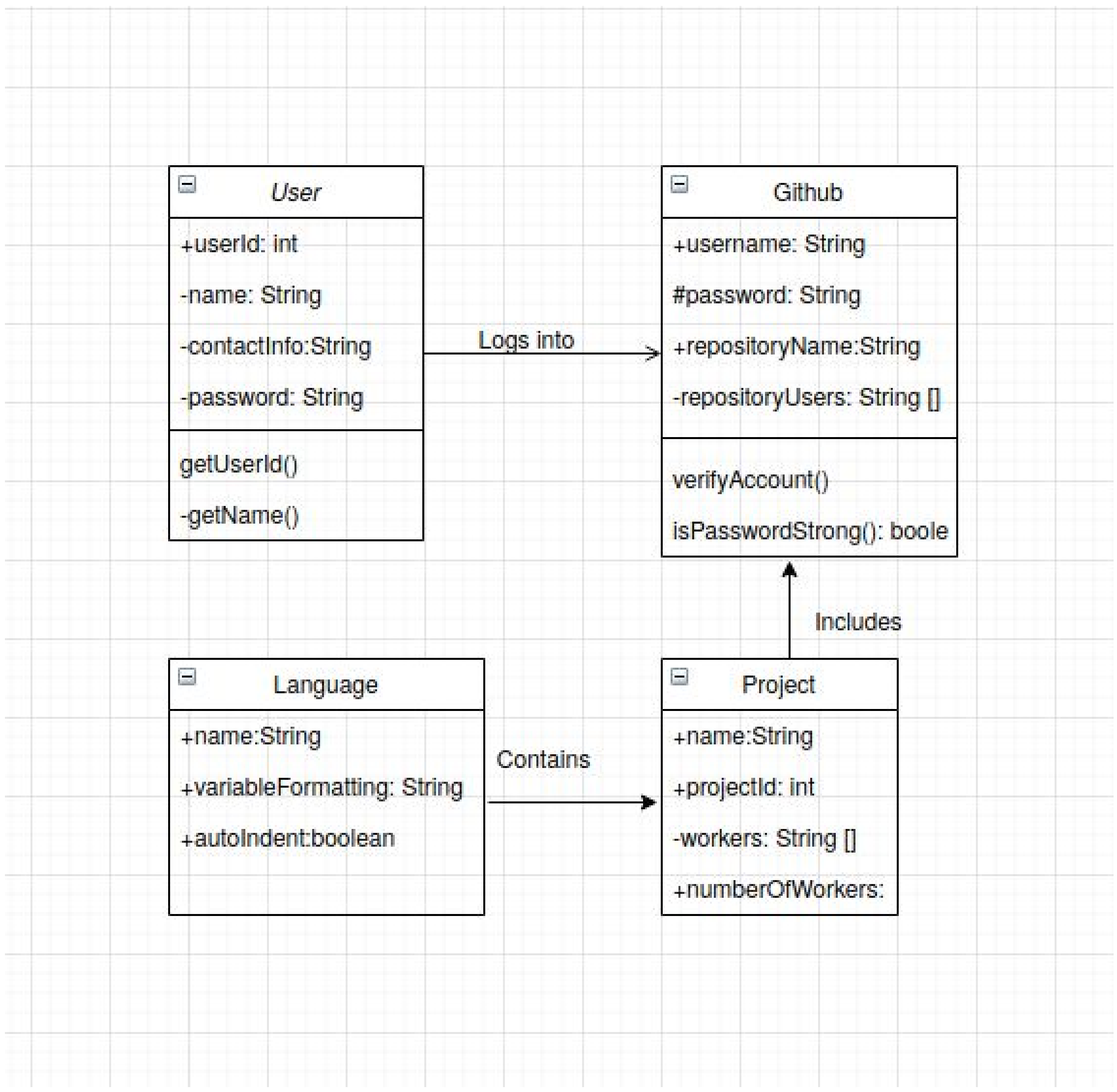
Normal Scenario:

- The code has passed all previous stages in the pipeline.
- The code is deployed to a specified environment.

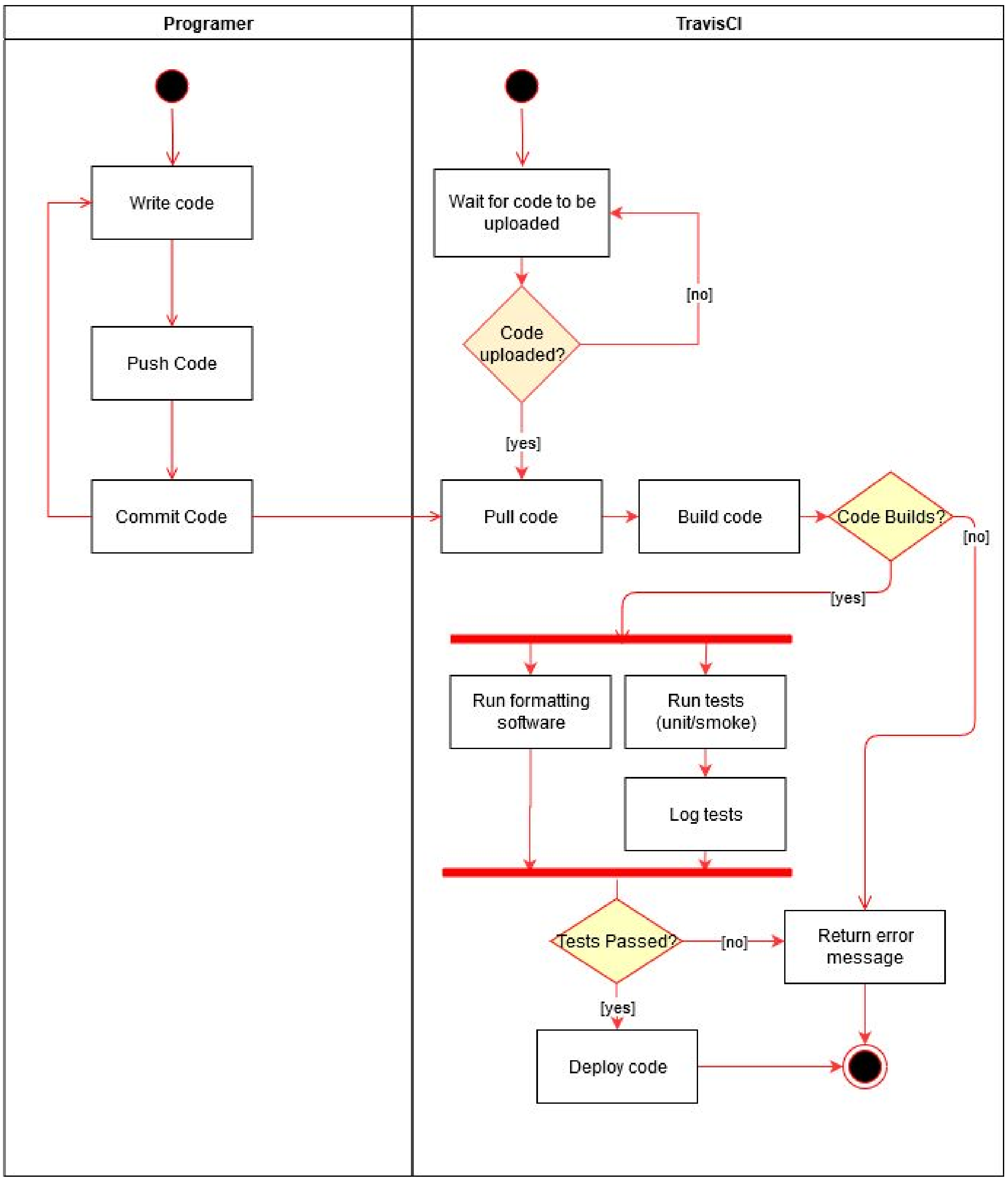
Error Scenario:

- The code is deployed to the wrong server.

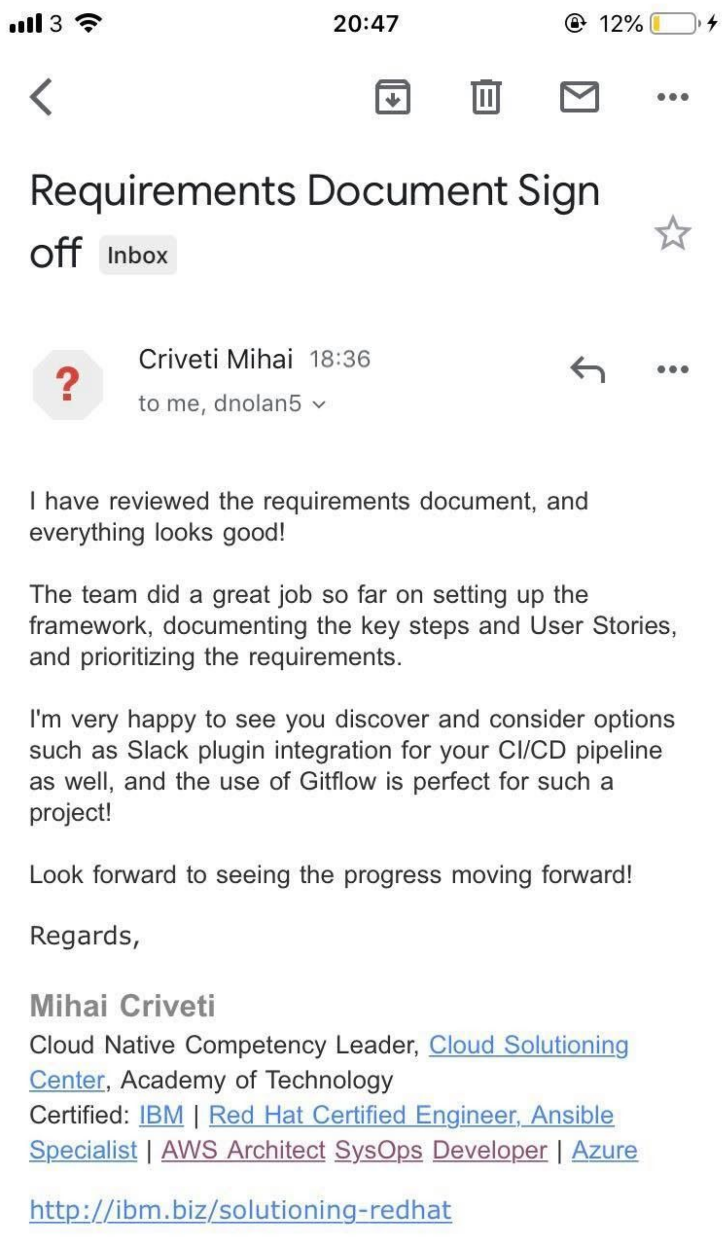
3.4.3 Object Model



3.4.4 Dynamic Model:



Client Sign off


3 20:47 12%

Requirements Document Sign off Inbox

Criveti Mihai 18:36

to me, dnolan5

I have reviewed the requirements document, and everything looks good!

The team did a great job so far on setting up the framework, documenting the key steps and User Stories, and prioritizing the requirements.

I'm very happy to see you discover and consider options such as Slack plugin integration for your CI/CD pipeline as well, and the use of Gitflow is perfect for such a project!

Look forward to seeing the progress moving forward!

Regards,

Mihai Criveti
Cloud Native Competency Leader, Cloud Solutioning Center, Academy of Technology
Certified: IBM | Red Hat Certified Engineer, Ansible Specialist | AWS Architect SysOps Developer | Azure

http://ibm.biz/solutioning-redhat

Appendix Two: Software Design Specification

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/

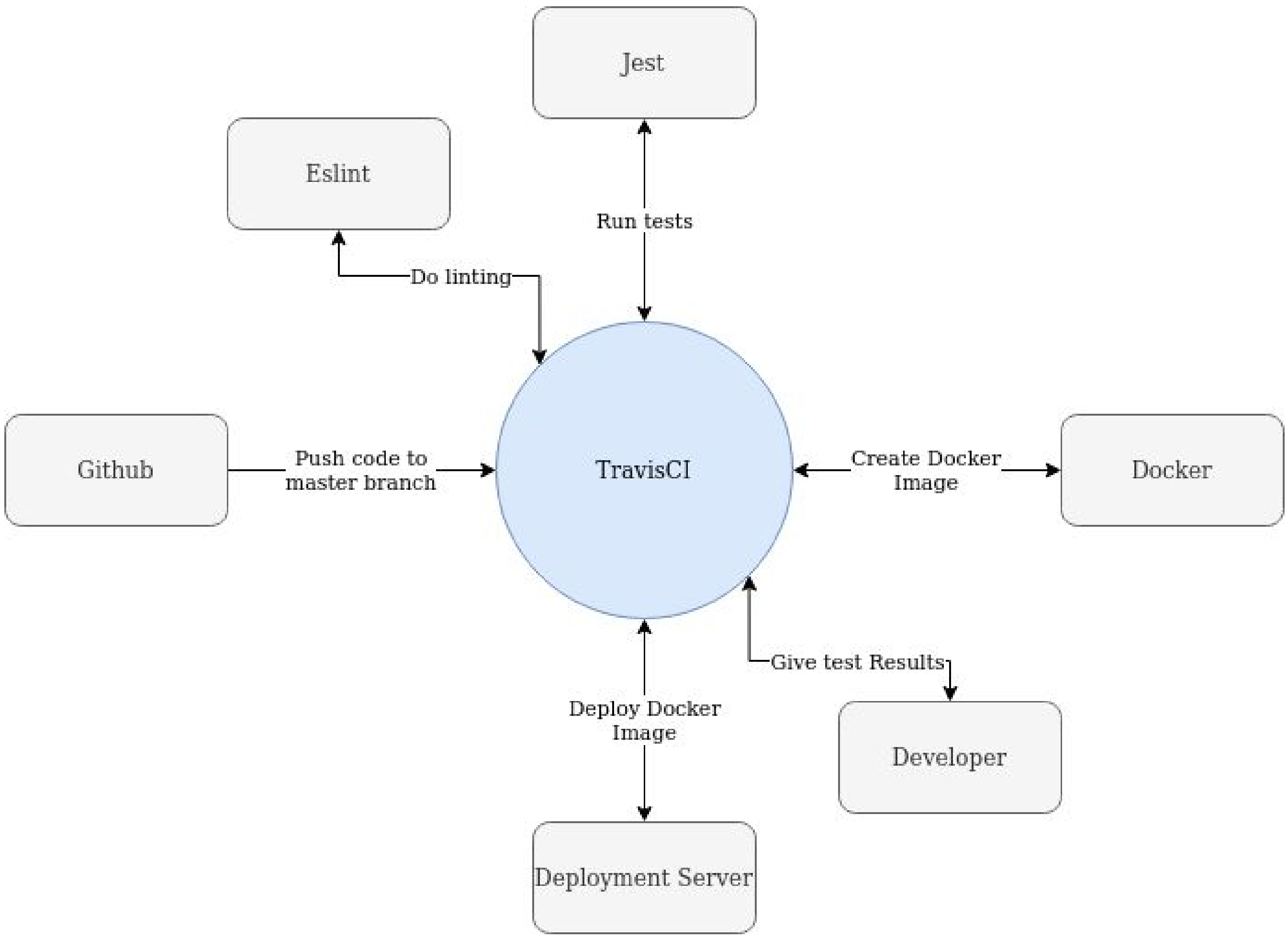
2. System Design

2.1. Design Overview

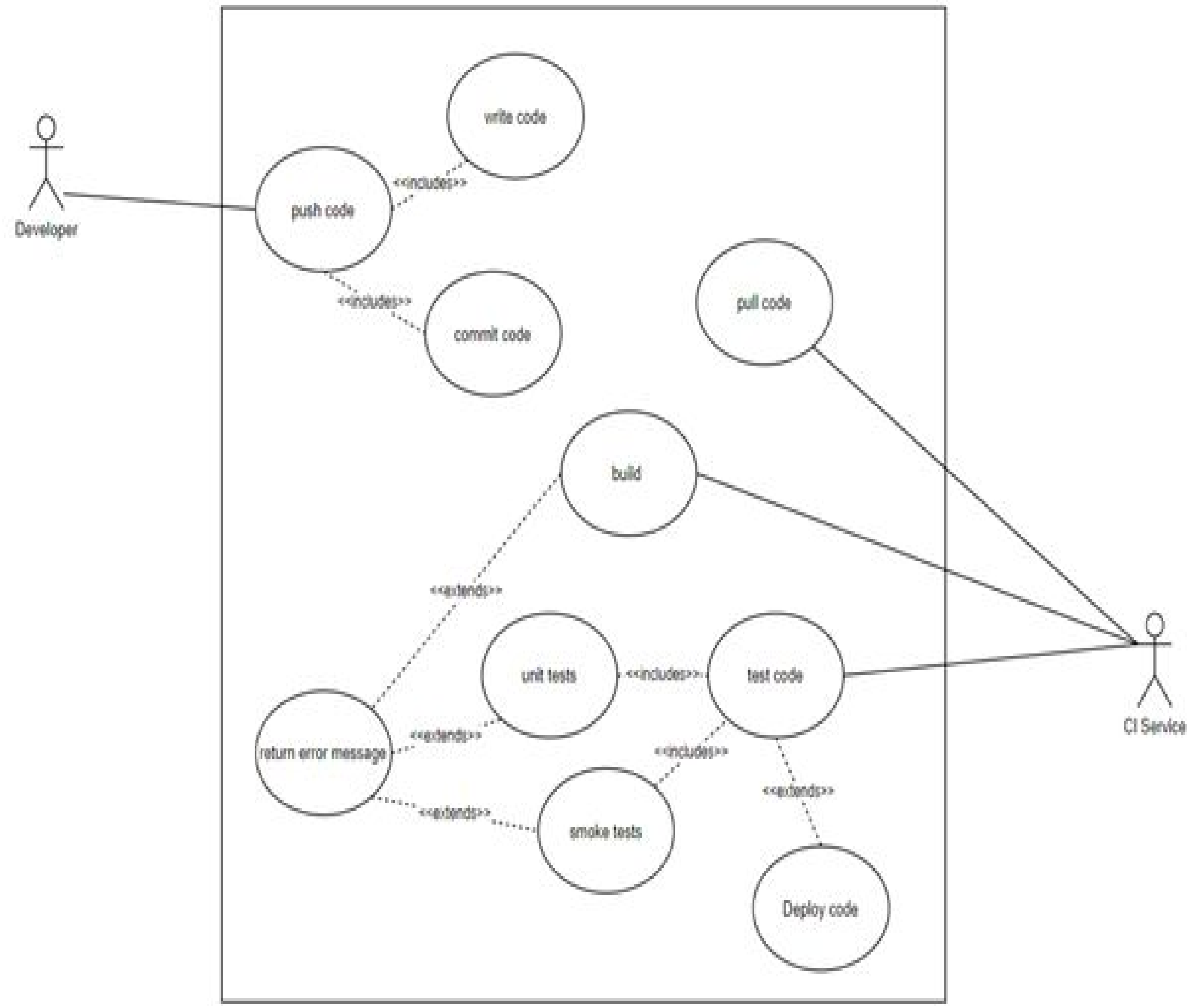
A complete CI/CD pipeline and a simple application to demonstrate the use of this pipeline. The simple application in question will be a reactJS app with an Express backend. This will be hosted using Heroku. Version-control will be handled with Github and changes to the master branch will trigger an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

2.2. System Design Models

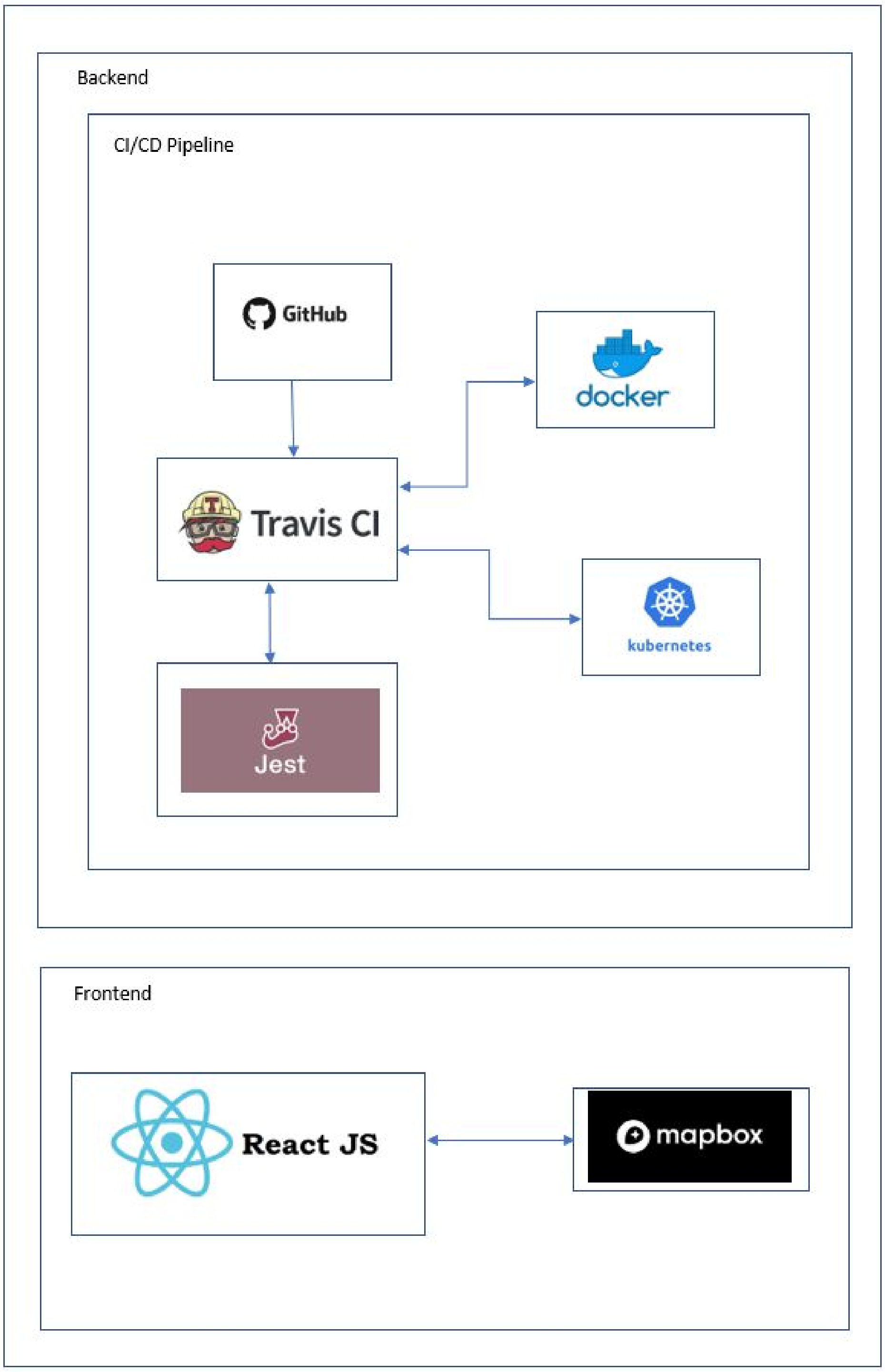
2.2.1.System Context



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.

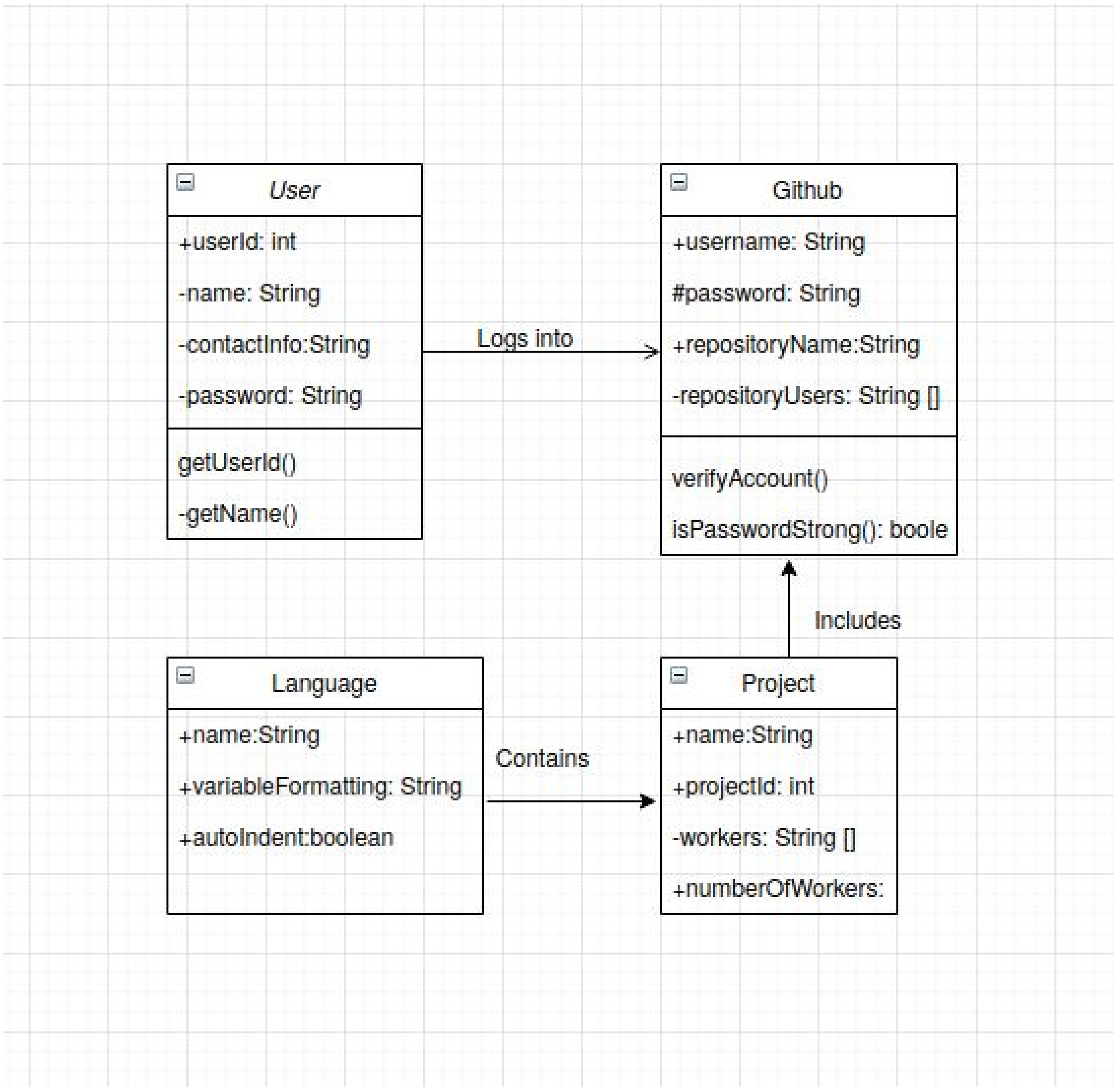
Architectural Decisions:

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/

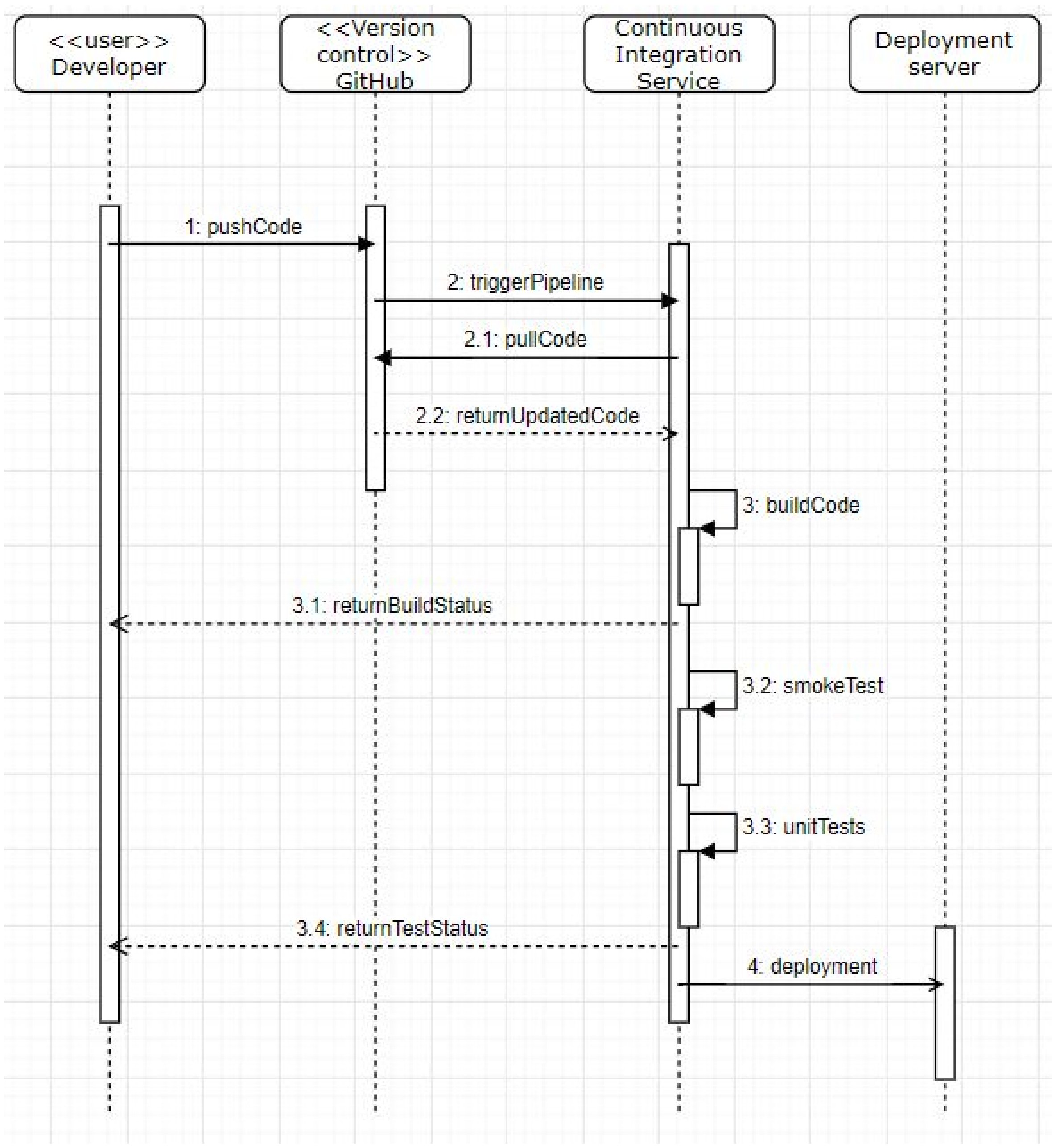
Architectural Decision:	Use GitHub as the main Version-control platform.
Issue or Problem Statement:	There are several different CI solutions to choose from
Motivation:	Support the Continuous Integration of the application.
Alternatives:	TortoiseSVN, Apache Subversion
Justification:	We chose GitHub because 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/

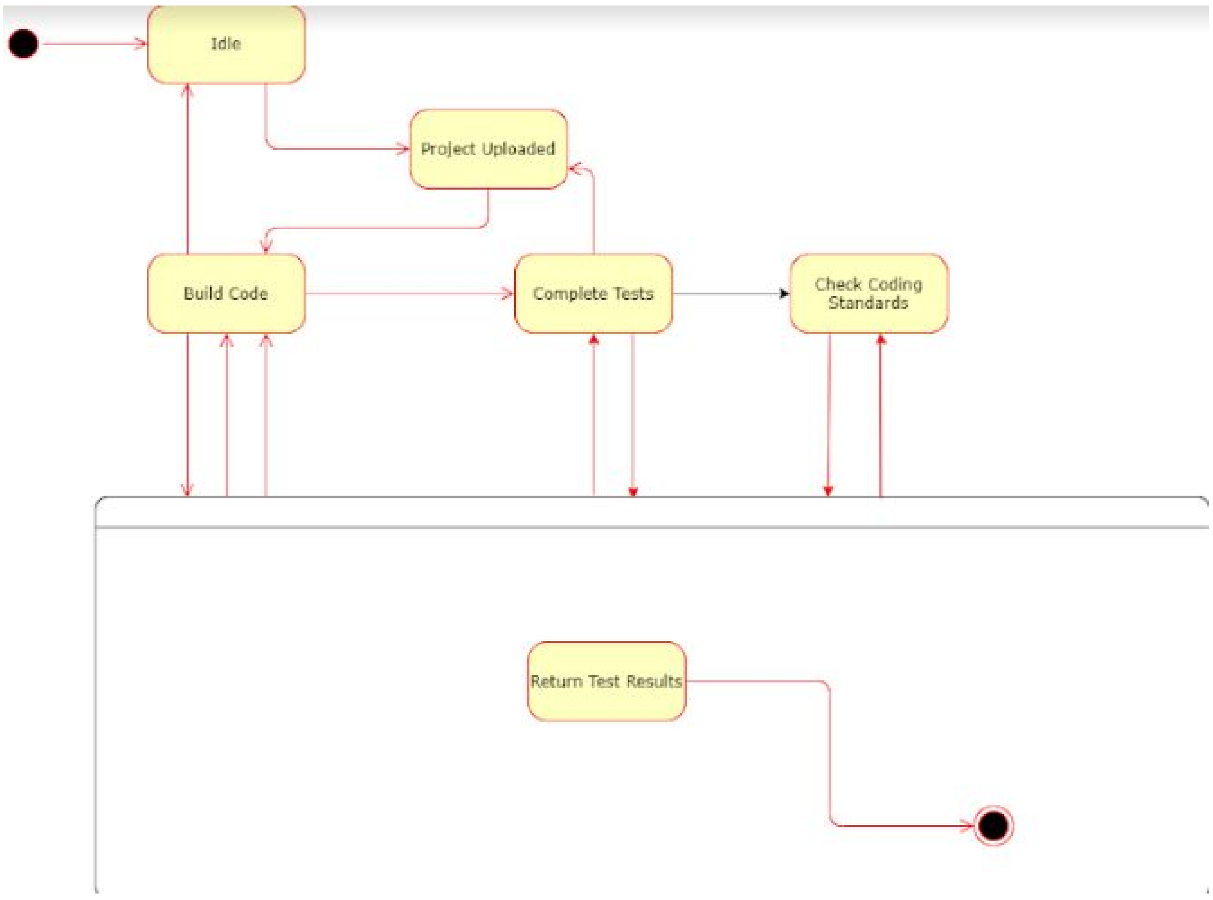
2.2.4.Class Diagrams



2.2.5.Sequence Diagrams

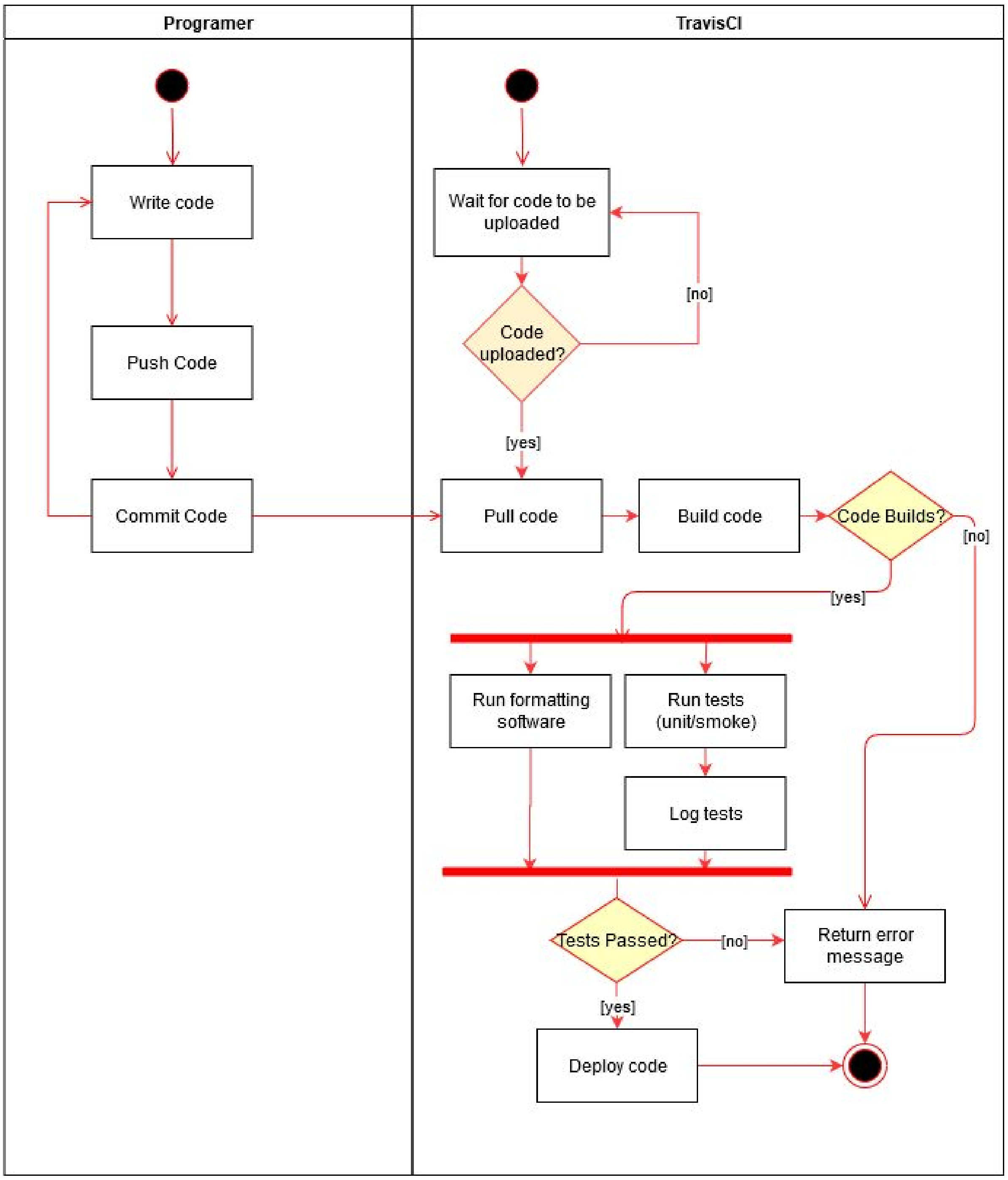


2.2.6.State Diagrams



2.2.7.Other relevant models:

Dynamic Model:


Programer
TravisCI
Write code
Push Code
Commit Code
Wait for code to be uploaded
Code uploaded?
[no]
[yes]
Pull code
Build code
Code Builds?
[no]
[yes]
Run formatting software
Run tests (unit/smoke)
Log tests
Tests Passed?
[no]
[yes]
Return error message
Deploy code