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:

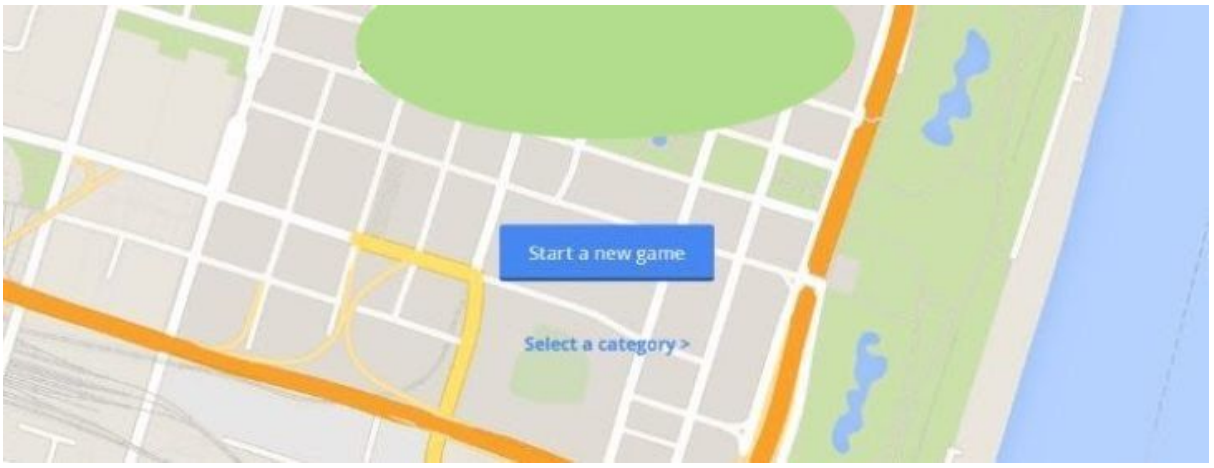
1) Integrating notifications from the pipeline with some sort of messaging, for example Slack.
2) Augment the pipeline with a local stage, by using Git Hooks to enforce coding standards before any code can be added to Github.
3) Add a test deployment environment, so that end-to-end testing can happen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

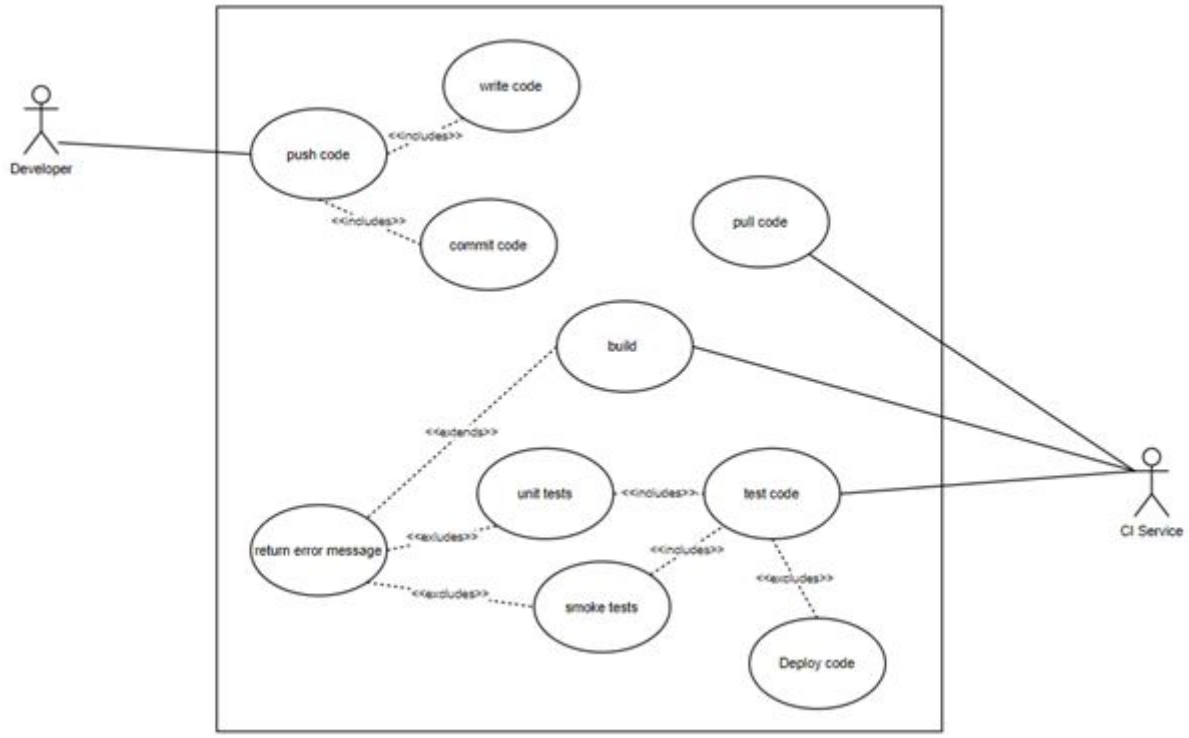


- Upon starting a new game, the user will be presented with a question in a pop-up box.

- 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.

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

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.

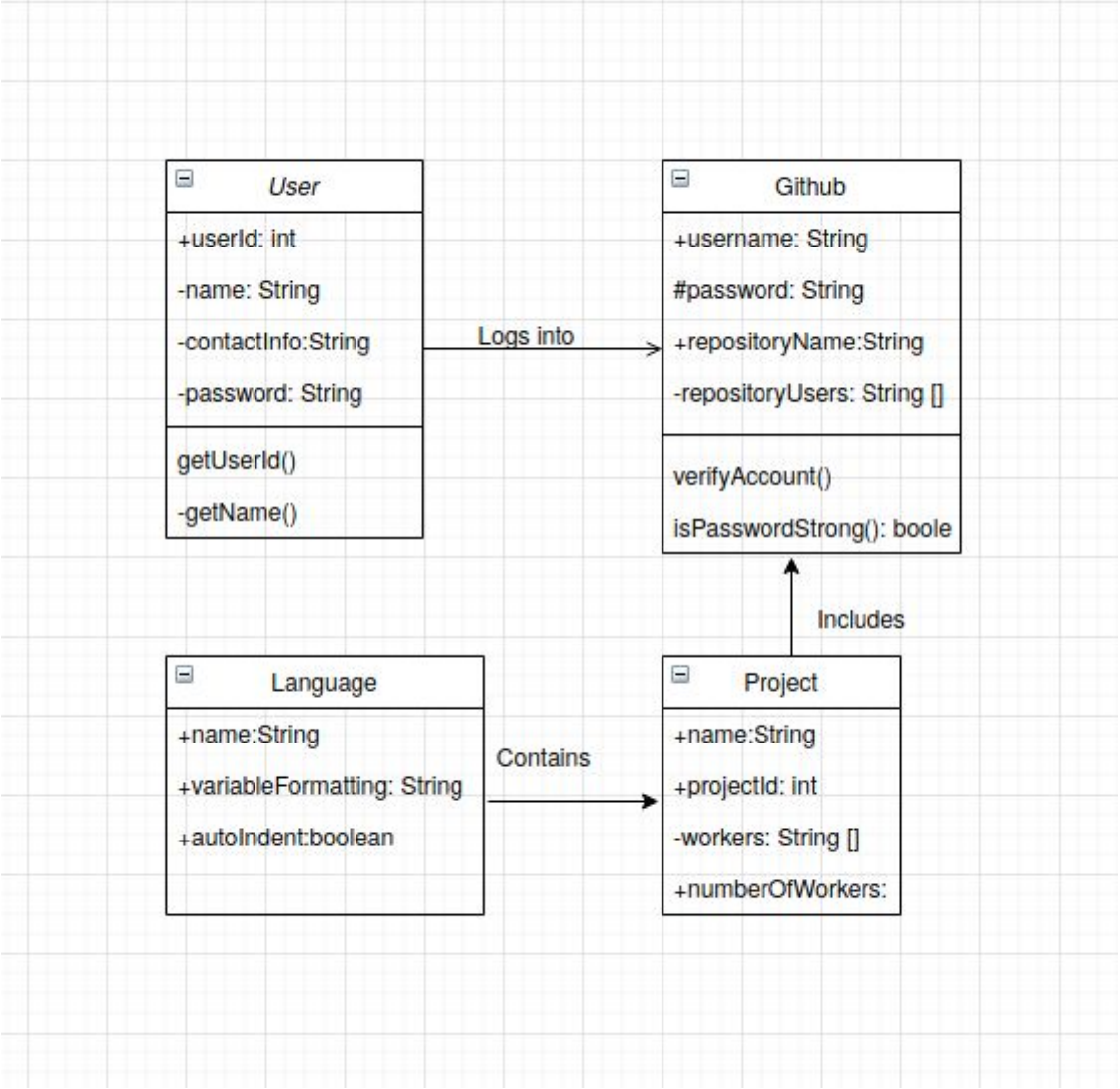
Normal Scenario:

- 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: