Trinity College Dublin
The University of Dublin

IBM TODO App Project

Software Design Specification Document

Software Engineering Project (CSU33013 & CSU22013)

14th March 2022

Brian Whelan (3rd year)

Adriana Hrabowych (3rd year)

Tom Roberts (3rd year)

Nadia Abouelleil (3rd year)

Arshad Rehman Mohammed (2nd year)

James Merrins Pryce (2nd year)

Liam Reilly (2nd year)

1. Introduction

Overview - Purpose of system

The aim of this project is to create a containerized TODO application from scratch. The application will strive towards helping people organize their day to day tasks in the form of a TODO application. The application will list a user's tasks and allow the user to add or delete tasks in their list so that they never forget what they are meant to be working on.

The client, IBM, has expressed that the focus of the project should be less about the application itself, but the process of employing formal software development and design principles frequently used in the computer science industry. For example, using proper version control, working on both backend and frontend at once, testing every update, etc.

Scope

- A containerized, fully tested, TODO application which allows the user to add and delete tasks
- The app should provide a clear and easy to use UI
- Static software analysis and automated testing
- Automatic deployment to Red Hat OpenShift

Definitions, abbreviations

Standardization: This is a document or file format that is used by one or more software developers while working on the same program. Software standards enable the interoperability of programs written by different developers.

Self-Contained/Containerized: When a program/application is contained within a dedicated separate environment in an operating system that contains only what is needed by the program/application to run.

CI/CD: Continuous Integration/Continuous Deployment

CI/CD Pipeline: Automates the steps in the software development process. The pipeline builds code, executes tests (CI), and deploys a new version of the application in a safe manner (CD). Automated pipelines eliminate any manual errors, provide developers with standardized feedback loops, and allow for rapid product integration.

References

Red Hat OpenShift Developer Sandbox -
https://developers.redhat.com/developer-sandbox

Red Hat CodeReady Containers -
https://developers.redhat.com/products/codeready-containers/overview

Local development with OpenShift -
https://developers.redhat.com/blog/2019/09/05/red-hat-openshift-4-on-your-laptop-introducing-red-hat-codeready-containers

Deploying to OpenShift using GitHub Actions -
https://cloud.redhat.com/blog/deploying-to-openshift-using-github-actions

2. System Design

Design Overview

Some of the architectural decisions were made for us by the client in the project specification but a number of decisions had to be made by the team as we encountered certain issues. Through consultation with the client, it was advised to document these decisions to ensure that we were considering all possible options and justifying our final architectural design.

Below are the architectural decisions we encountered over the course of the project, the solutions we considered and the ultimate decision we made to solve the issue. These architectural decisions have been created using the architecture decision description template published in "Architecture Decisions: Demystifying Architecture" by Jeff Tyree and Art Akerman, Capital One Financial. These templates are active documents and will be updated as and when architectural decisions are made or changed over the course of the project.

Issue	The project requires collaboration and constant development of an application, and therefore it must be possible to easily make and revert changes in the codebase.
Decision	Use a version control system, namely Git, to track changes and versions of the codebase to enable backtracking.
Status	Decided
Group	Project Management
Assumptions	● We cannot guarantee asynchronous development amongst the team
Constraints	None

Positions	• Using SVN version control
Argument	Git version control is distributed, meaning that we can each have a local copy of the repository, make changes and commits and then push to the remote repository when we have a connection. This enables work to be done even when offline. In contrast, SVN is centralized, meaning commits cannot be made locally and require a connection.
Implications	The team will need to download Git onto their local machine to enable the creation and cloning of repositories.
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	• All of the team has some minor experience with SVN from 1st year programming module • Those who have experienced Git have found it much more user friendly and easy-to-use • Git is the most popular version control system and would be useful to have knowledge of going forward

Issue	The project requires a well-defined backlog to be maintained to ensure each team member is clear on what has been done, is being done and needs to be done at every stage.
Decision	Use a Kanban board, namely GitHub Project Boards, to define the project backlog, assign tasks, and monitor the progress of the project.
Status	Decided
Group	Project Management
Assumptions	• Managing the project backlog through scrum meetings alone is not viable
Constraints	None
Positions	• Using Trello
Argument	As we will be using GitHub, using GitHub Project Boards offers a seamless integration of the kanban board with the remote repository. Using a separate kanban board tool such as Trello would have required someone to maintain the board regularly as opposed to having the board update dynamically as tasks

	are completed within the repository.
Implications	The team will need to learn how to make use of the Project Boards feature by creating cards, assigning them and updating their progress.
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	● Some of the team has used Trello before, however GitHub Project Boards offers a more transparent kanban board feature and is integrated in the GitHub ecosystem

Issue	The project requires the automation of software development workflows, namely automating the integration and deployment of the application.
Decision	Use a platform, namely GitHub Actions, that enables the automation of the build, test and deployment pipeline
Status	Decided
Group	CI/CD Pipeline
Assumptions	None
Constraints	None
Positions	● Using Jenkins
Argument	As we will be using GitHub, GitHub Actions offers an integrated platform with the GitHub ecosystem. The learning curve for GitHub Actions also seems smaller than that of Jenkins which is key in the short time of the project.
Implications	The team will need to become familiar with creating workflows using GitHub Actions and customising them to the project requirements.
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None

Notes	● Very little knowledge of automation tools such as GitHub Actions or Jenkins amongst the team ● Learning curve of GitHub actions seems relatively smaller than that of Jenkins

Issue	The project requires that the application data be stored in a persistent database
Decision	Use MongoDB to store and retrieve application data
Status	Pending
Group	Backend
Assumptions	● The data must be easily accessible
Constraints	None
Positions	● Using MySQL ● Using SQLite ● Using PostgreSQL
Argument	
Implications	The backend team will need to become familiar with the MongoDB API for storage and retrieval of data, as well as configuring the database as required.
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	

Issue	The project will be completed over a number of months and as such it is essential that all additions can be verified to work as intended to prevent any unnecessary blockages.
Decision	Use a JavaScript unit testing framework, namely Jest, to write unit tests for new functions and features.
Status	Decided
Group	Backend
Assumptions	● Building a substantial application is not feasible if bugs cannot be easily found and rectified

Constraints	None
Positions	● Using Mocha ● Using Cypress ● Using Webdriver
Argument	Jest was designed by Facebook developers to work with React and so it fits seamlessly with our React application. Jest is also relatively simple and requires no pre-configuration unlike Mocha. Jest is also specifically designed for unit testing as opposed to more comprehensive testing frameworks like Cypress and Webdriver which offer a variety of additional testing mechanisms which are not required for the purposes of this project.
Implications	The backend team will need to become familiar with the syntax and write Jest unit tests when developing new features and functions.
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	● The team all has some knowledge of unit testing and have all used JUnit to test Java applications ● Using JavaScript itself is new to the majority of the team aso using a simple unit testing framework would be preferable ● As we are using React, it seemed obvious to use Jest given how closely linked the two are

Issue	The project will require a frontend UI for the application that is both functional and visually appealing.
Decision	Use JavaScript and moreover use the React library to develop the application frontend
Status	Decided
Group	Frontend
Assumptions	None
Constraints	None
Positions	● Using vanilla JavaScript ● Using Vue.js

Argument	Using React removes many of the complexities of working vanilla JavaScript and greatly simplifies the development of the application. React is also more catered toward cross-platform as opposed to Vue.js and while this application will not be cross-platform for this project, given the nature of the application, it may be something to consider for future development.
Implications	The frontend team will need to familiarise itself with React and JavaScript in general.
Related Decisions	• Choice of unit testing framework
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	• JavaScript offers the most comprehensive and common tools and frameworks for developing frontend applications • Some of the team members have some basic understanding of JavaScript and some have some experience with React

Issue	The project require a backend framework to build the application
Decision	Use Node.js
Status	Decided
Group	Backend
Assumptions	None
Constraints	None
Positions	• Use Java
Argument	Using Node.js integrates well with our frontend React application and also makes integration with a database relatively straightforward.
Implications	The backend team will have to learn Node.js.
Related Decisions	• Choice of frontend • Choice of unit testing framework • Choice of database

Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	• The team has extensive knowledge of Java from college already • Using JavaScript and Node.js might be best for both learning and also for the purposes of the application • The JavaScript syntax is relatively similar to that of Java and so should be relatively familiar

Issue	The project requires some static analysis of the codebase to assess code quality, detect bugs and detect vulnerabilities
Decision	Use Sonarqube to perform static analysis of the codebase
Status	Pending
Group	CI/CD Pipeline
Assumptions	• The quality and security of code cannot be left solely to developers
Constraints	None
Positions	• Using Sonarqube • Using pre-commit • Using snyk
Argument	Sonarqube is an industry standard and probably the most valuable static analysis tool to use
Implications	The team will have to become familiar with Sonarqube and static analysis
Related Decisions	None
Related Requirements	None
Related Artifacts	None
Related Principles	None
Notes	• Sonarqube is an industry standard and probably the most valuable static analysis tool to use

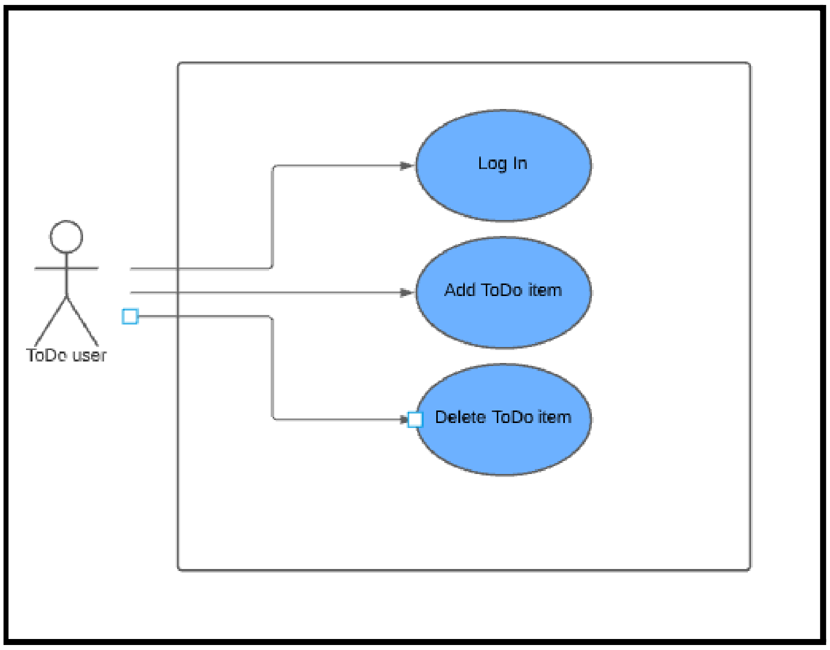
System Design Models

A. System Context

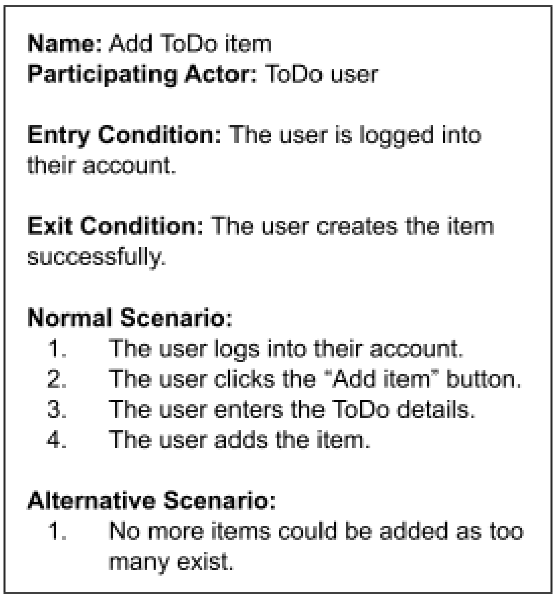
The system is designed to be a self contained application, able to run from a single download with no previous requisites. The project itself will interact with RedHat, Jest, a persistent MongoDB database, and other systems to aid in development and deploying.

B. Use cases (from Requirements)

<u>App Use Case Diagram</u>



<u>Use Case Descriptions</u>



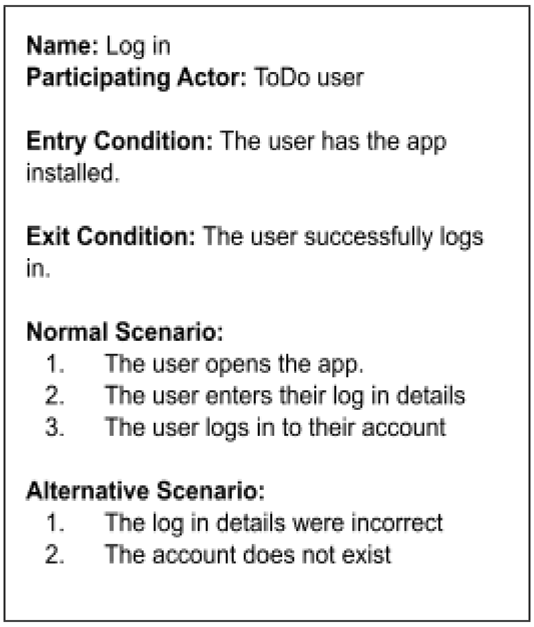
Name: Add ToDo item
Participating Actor: ToDo user

Entry Condition: The user is logged into their account.

Exit Condition: The user creates the item successfully.

Normal Scenario:
1. The user logs into their account.
2. The user clicks the "Add item" button.
3. The user enters the ToDo details.
4. The user adds the item.

Alternative Scenario:
1. No more items could be added as too many exist.



Name: Log in
Participating Actor: ToDo user

Entry Condition: The user has the app installed.

Exit Condition: The user successfully logs in.

Normal Scenario:
1. The user opens the app.
2. The user enters their log in details
3. The user logs in to their account

Alternative Scenario:
1. The log in details were incorrect
2. The account does not exist

Name: Delete ToDo item
Participating Actor: ToDo user

Entry Condition: The user is logged into their account and has at least one ToDo.

Exit Condition: The user deletes the item successfully.

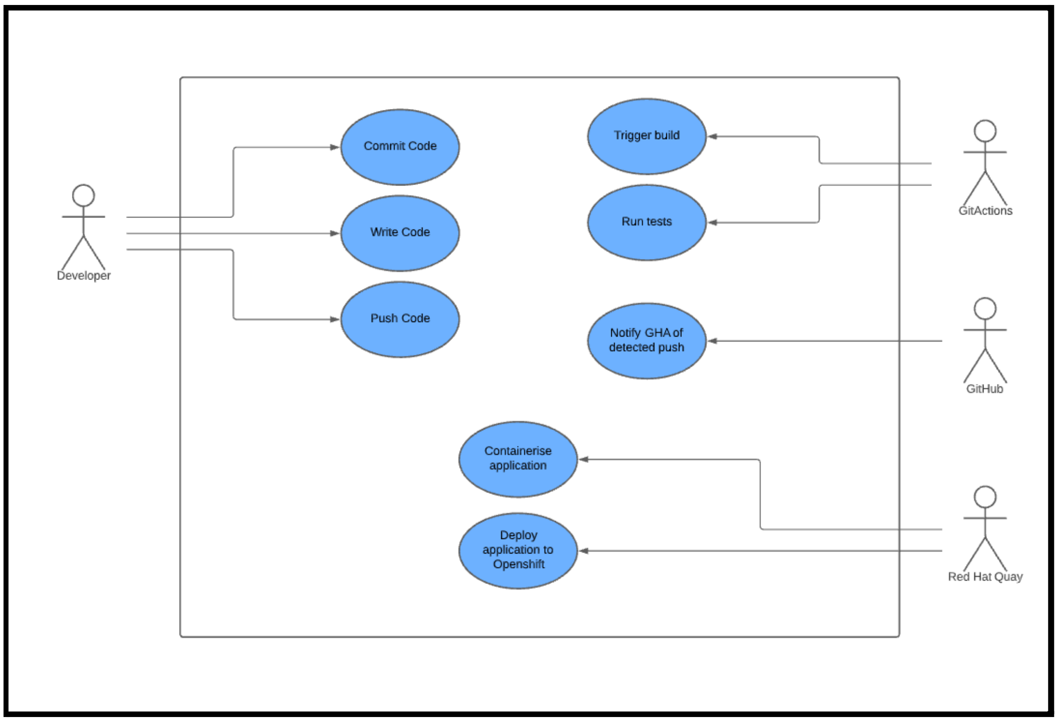
Normal Scenario:

1. The user logs into their account.
2. The user clicks the "Delete" button on the specified ToDo.
3. The user deletes the ToDo.

Alternative Scenario:

1. The user has no more ToDo items to delete

Pipeline Use Case Diagram



Use Case Descriptions

Name: Commit Code
Participating Actor: Developer

Entry Condition: The developer has made changes to the code base

Exit Condition: The user commits the changes successfully.

Normal Scenario:
1. The developer changes the codebase
2. The developer stages the changes.
3. The developer commits the changes.

Alternative Scenario:
1. The changes could not be committed

Name: Push Code
Participating Actor: Developer

Entry Condition: The developer has staged and committed changes to the code base

Exit Condition: The user pushes the changes to the remote repo successfully.

Normal Scenario:
1. The developer commits changes to the codebase
2. The developer pushes the changes to the remote repository

Alternative Scenario:
1. The local repository could not be pushed to the remote repository

Name: Trigger Build
Participating Actor: Github Actions

Entry Condition: The developer has pushed changes to the remote repository

Exit Condition: Github Actions successfully builds the new codebase.

Normal Scenario:
1. The developer pushes changes to the remote repository
2. Github Actions builds the new codebase.

Alternative Scenario:
1. Github Actions failed to build the code

Name: Run Tests
Participating Actor: Github Actions

Entry Condition: Github Actions has successfully built the pushed code

Exit Condition: The built code passes all tests

Normal Scenario:
1. Github Actions runs a test-suite against the new build.
2. The build passes all tests

Alternative Scenario:
1. The build fails to pass at least one of the tests.

Name: Notify Github Actions of a push
Participating Actor: Github

Entry Condition: The developer has pushed changes to the remote repository

Exit Condition: Github notifies github actions of the push.

Normal Scenario:
1. The developer pushes changes to the Github remote repository
2. Github notifies Github Actions of the push.

Alternative Scenario:
1. Github fails to notify Github Actions of the push

Name: Containerize application
Participating Actor: Red Hat Quay

Entry Condition: The code has been built by Github Actions and passed all of the tests

Exit Condition: The code is containerized successfully

Normal Scenario:
1. Github Actions successfully builds and tests the code
2. Red Hat Quay successfully containerizes the code

Alternative Scenario:
1. Red Hat Quay fails to containerize the code

Name: Deploy application to openshift
Participating Actor: Red Hat Quay

Entry Condition: The code has been containerized successfully.

Exit Condition: The code is deployed successfully to Openshift

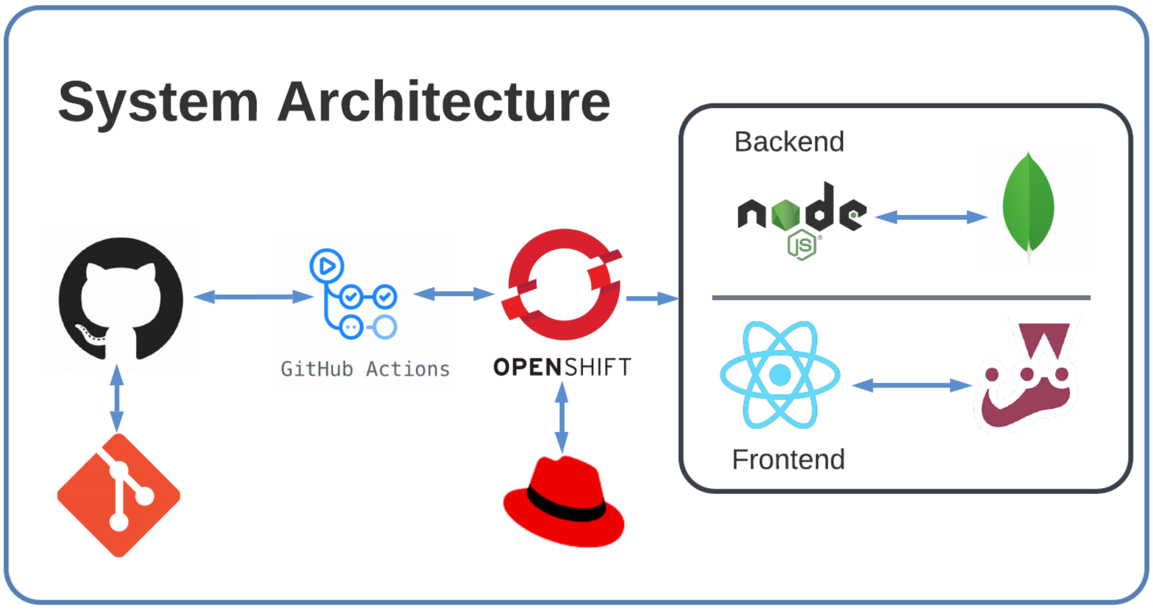
Normal Scenario:

1. The code is successfully containerized by Red Hat Quay
2. The code is successfully deployed to Openshift

Alternative Scenario:

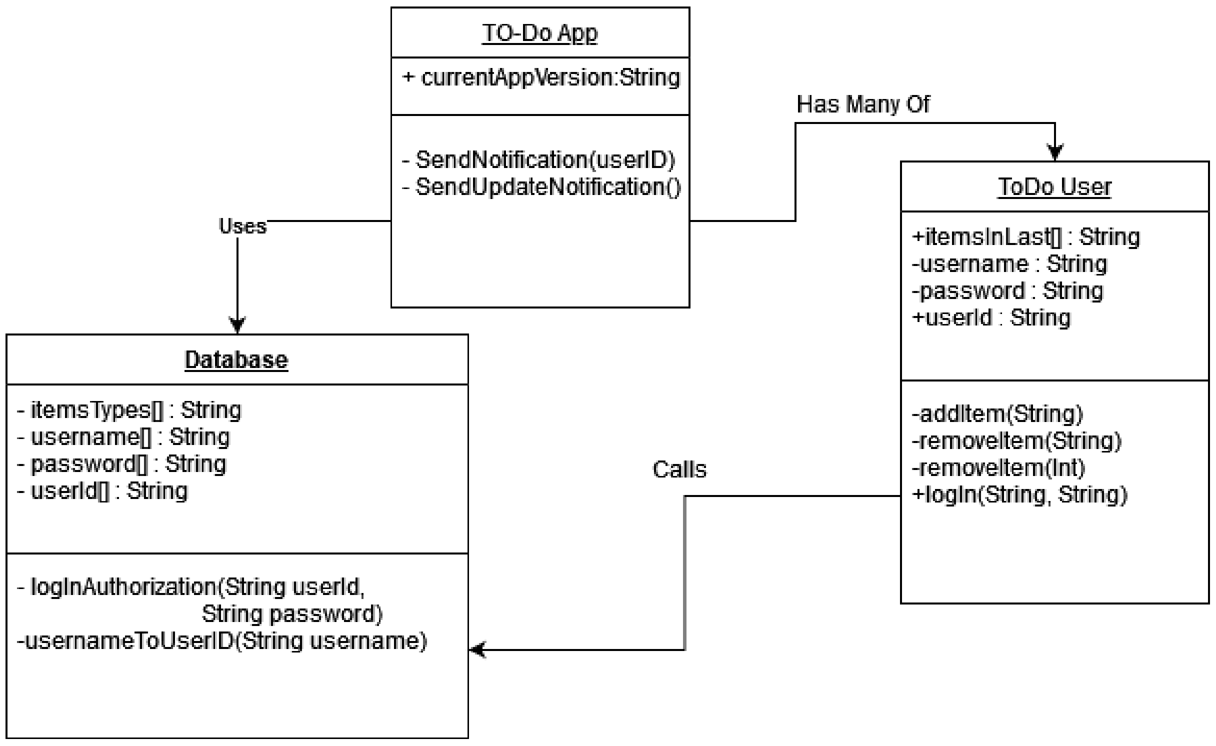
1. Red Hat Quay fails to deploy the code to Openshift

C. System Architecture



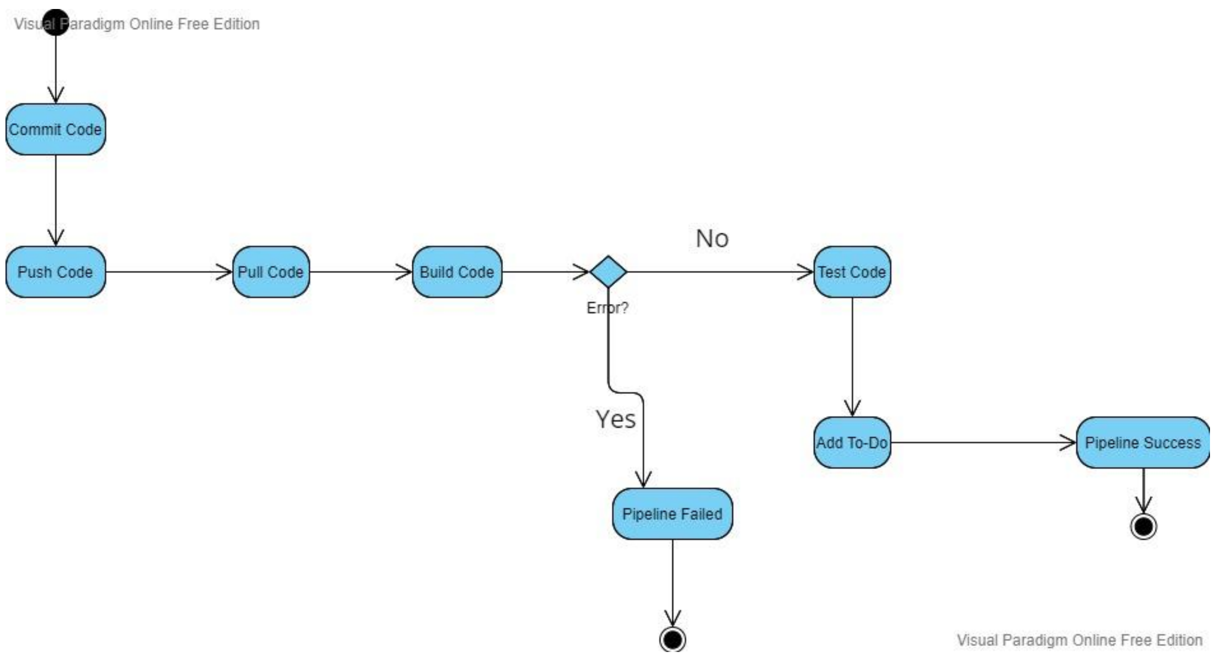
The system architecture diagram above describes how the different components and tools of our project work together to create our application. Our source code is stored in a Git repository on GitHub. GitHub Actions recognizes pushes to this repository and will deploy the application on OpenShift containers, with the image of these containers being stored on RedHat Quay The code for the application itself is written in Node.js for the backend, and React for the frontend. The backend is connected to a MongoDB database which stores our TODOs and Jest is connected to the frontend and tests the application.

D. Class Diagrams

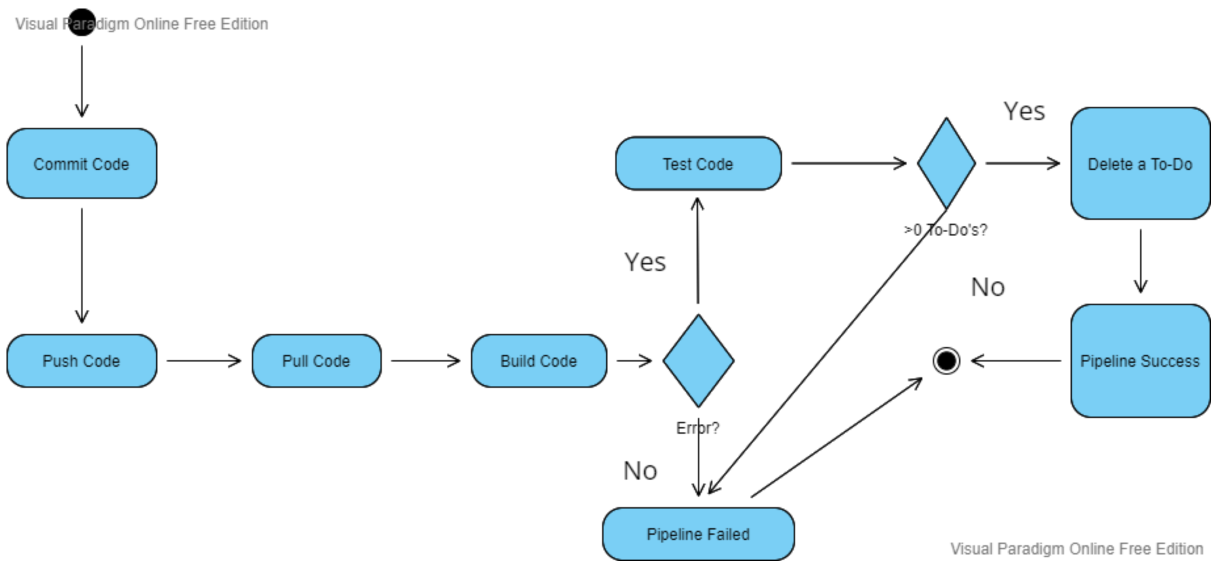


E. Sequence Diagrams

This sequence diagram shows the steps taken in order to add a TODO.

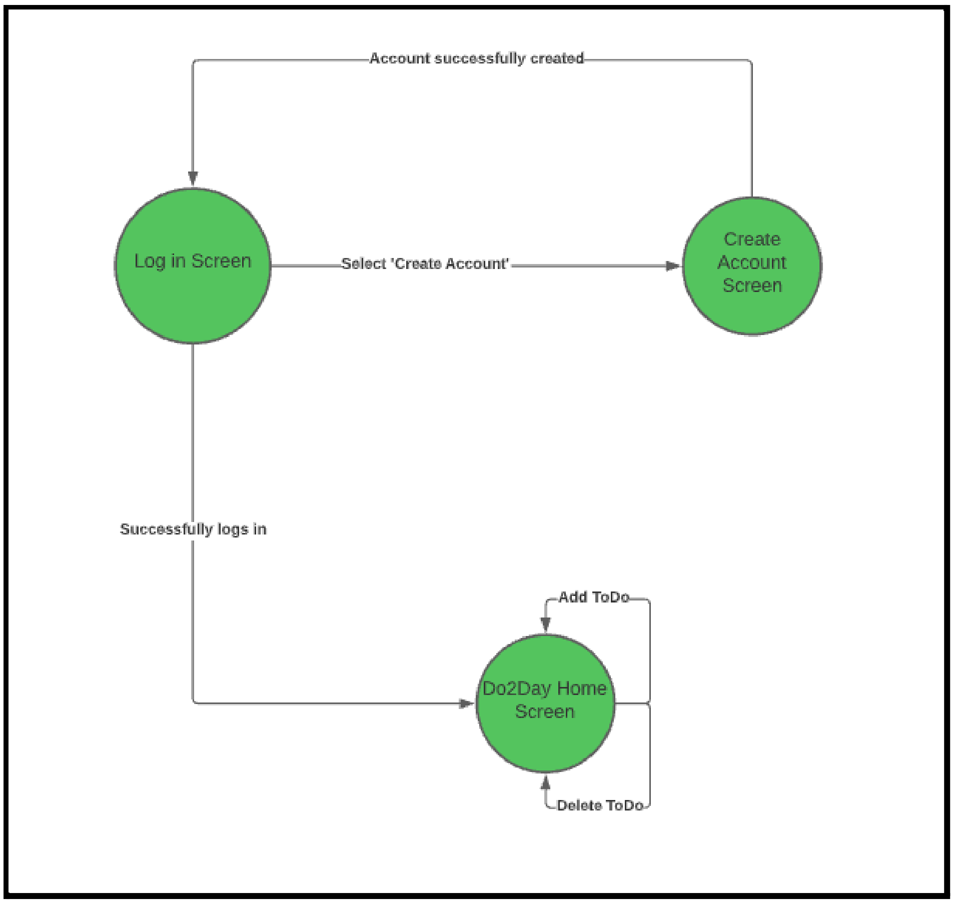


This sequence shows the steps taken to delete aTODO.



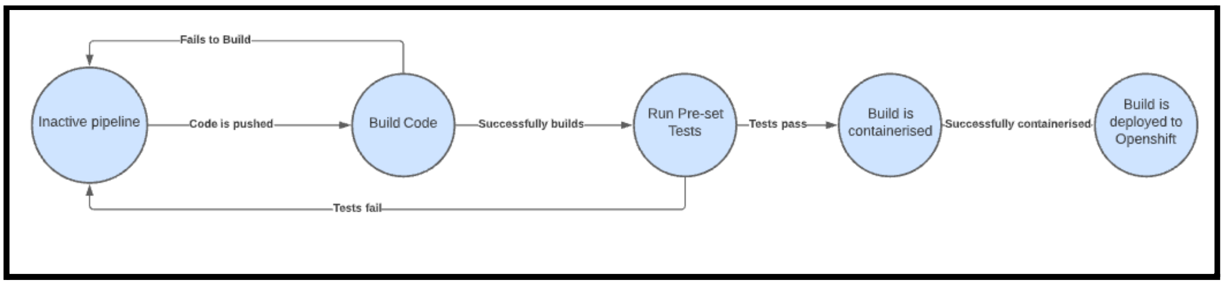
F. State Diagrams

App State Diagram



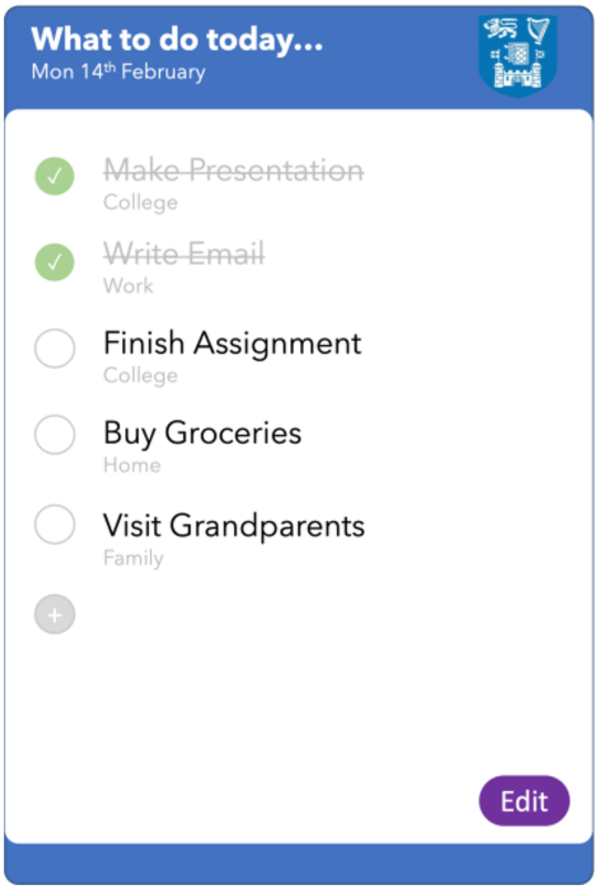
This diagram shows the states that the application can be in.

Pipeline State Diagram

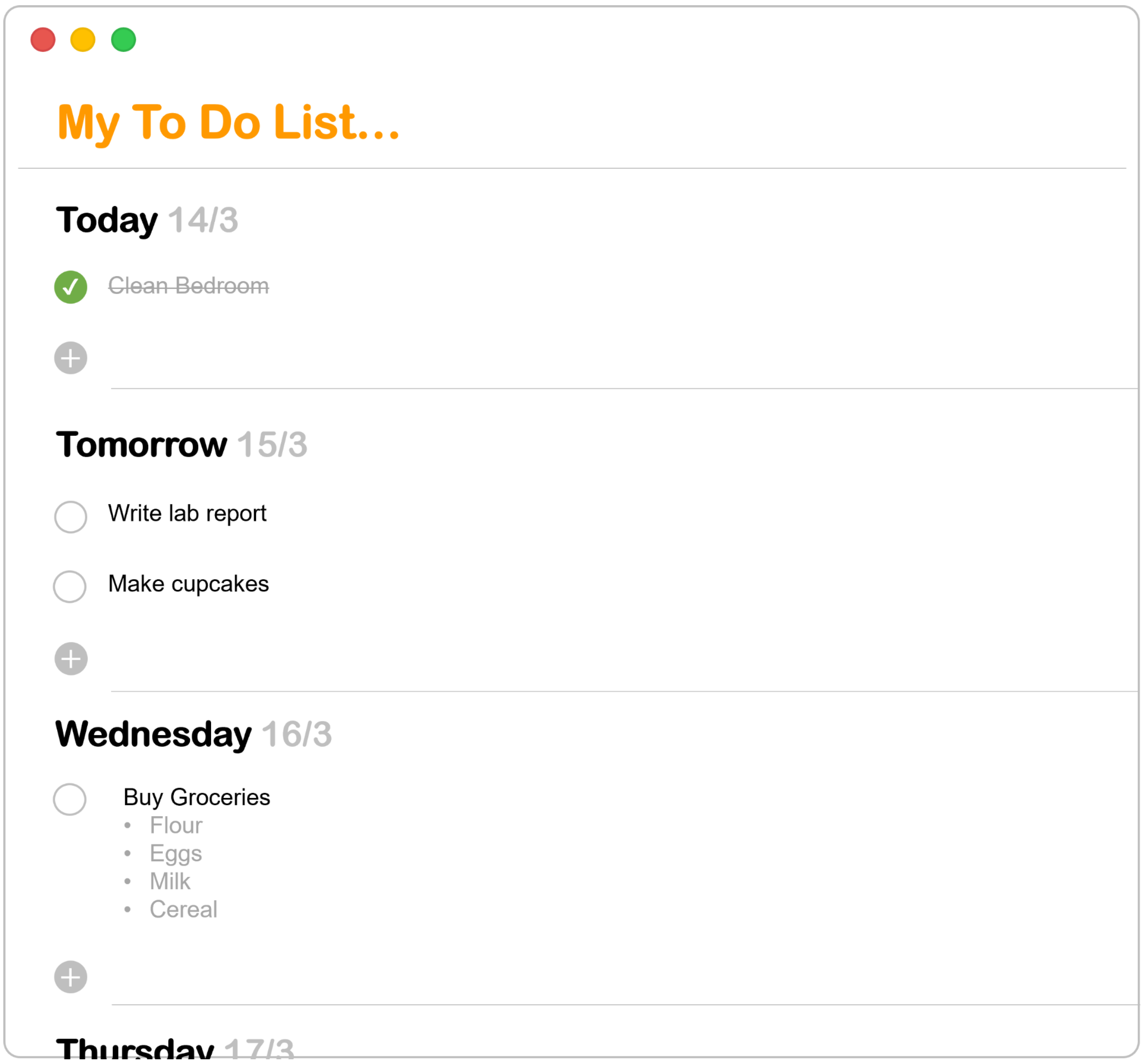


This shows the steps the pipeline goes through in its execution.

G. Other relevant models



UI Mock-up 1


My To Do List…

Today 14/3

✓ ~~Clean Bedroom~~

Tomorrow 15/3

○ Write lab report

○ Make cupcakes

Wednesday 16/3

○ Buy Groceries
- Flour
- Eggs
- Milk
- Cereal

Thursday 17/3

UI Mock-up 2