To Do List Application

BY PETER VAUGHAN-WILLIAMS

Introduction

- Who am I?

My name is Peter Vaughan-Williams; a QA SDET Trainee learning Java, HTML, CSS and JavaScript for the first time making this my first ever full stack project! The technologies and concepts used in this project were initially hard to grasp but with time and dedication, I've come to understand it more and become more familiar with all the technologies used.

How did I approach the specification?

Initially, I ran through the specification to understand what was expected of me and from this project, to understand the deadline and what technologies I would need to incorporate throughout this project. The main planning I had to make was with regards to the backend interacting with the front end of the project, making sure that all the java was set up and written correctly, similarly with all my fetch methods in my javascript ensuring that the CRUD functionality for both entities would actually work.

Concept

- How did I initially approach the problem?

Firstly, I identified what I needed to achieve through the use of User Stories on my Jira Scrum board, broke down the project into multiple little steps and assigned each step a sprint, so I could tackle the problem in 3 major sprints; a back end sprint, a front end sprint and a testing sprint. From this, I was able to figure out a starting point for the backend and continue from there.
Alongside the user stories, I also created an entity relationship diagram (ERD) and a unified modeling language diagram (UML) to understand how each part of the backend would interact with one another. From there I was able to continue in a similar fashion and approach the front end and testing sprints in a similar way.

Sprint Plan

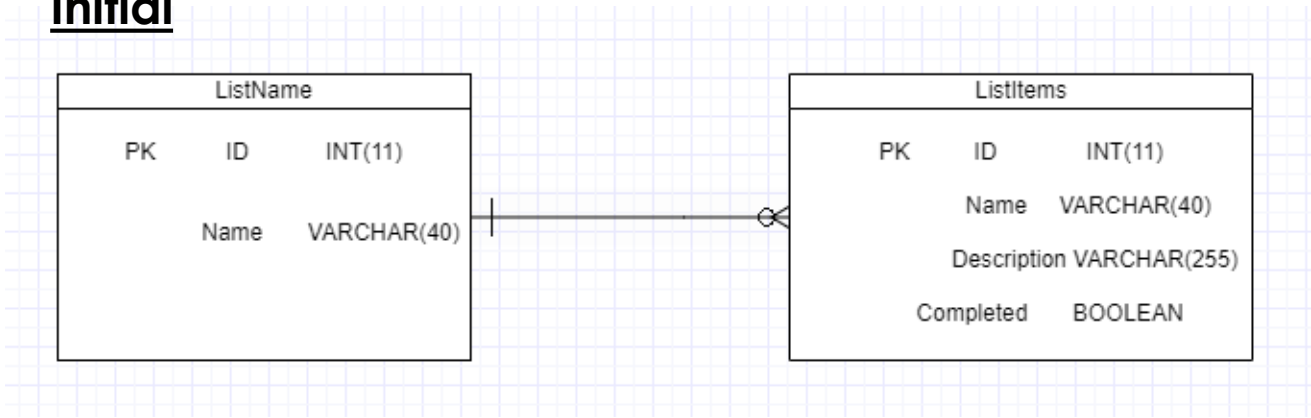
- What needed to be included?
- The project must have a Java back-end developed with Spring Boot allowing for the processing of dealing with the user's input and manipulating the SQL database.
- The project must have a HTML front-end developed webpage that presents the user with their Lists and Items that they can interact with.
- Must have respective JavaScript fetch commands for the CRUD functionality of both entities (ListName and ListItem)
- What did I hope to achieve?

I hoped to have a nicely developed front page utilizing bootstrap and CSS to make the user interface a nice experience for the user as well as custom queries/methods for extra functionality i.e. quickly merging two lists together.

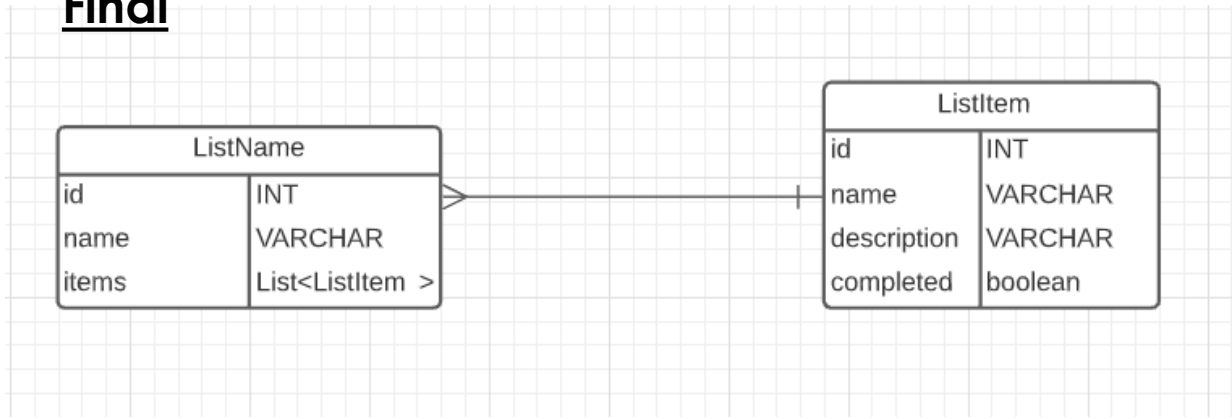
My Entity Relationship Diagram

This is the before and after of my Entity Relationship Diagram I created in order to understand the SQL relationship between the two entities I created in SpringBoot; where I would need to know the difference between the two entities, their respective parameters and the relationship they share.

Initial



Final



Consultant Journey

- What technologies have I learnt for this project?

Languages:
Backend written in: Java
Frontend written in: HTML, CSS, JS

Postman: The tool I used to interact with the API and send HTTP Requests to initially test functionality.

Lombok: A tool I used to generate functionality methods in classes

Spring Tool Suite 4: This was the IDE used to develop the backend, similar to Eclipse, but with Spring functionality

SonarQube: code smells

Jira: The project management platform used to plan the project.

Maven: Build tool used to inject dependencies, clean and package the project into .war

Mockito: Tool used to mock dependencies during testing

Visual Studio Code: The IDE used to develop the front end

Selenium: Tool used during

Junit: Tool used to write unit tests

Git: The Version Control System I used to keep track and update the project

Continuous Integration

- How did I approach version control?

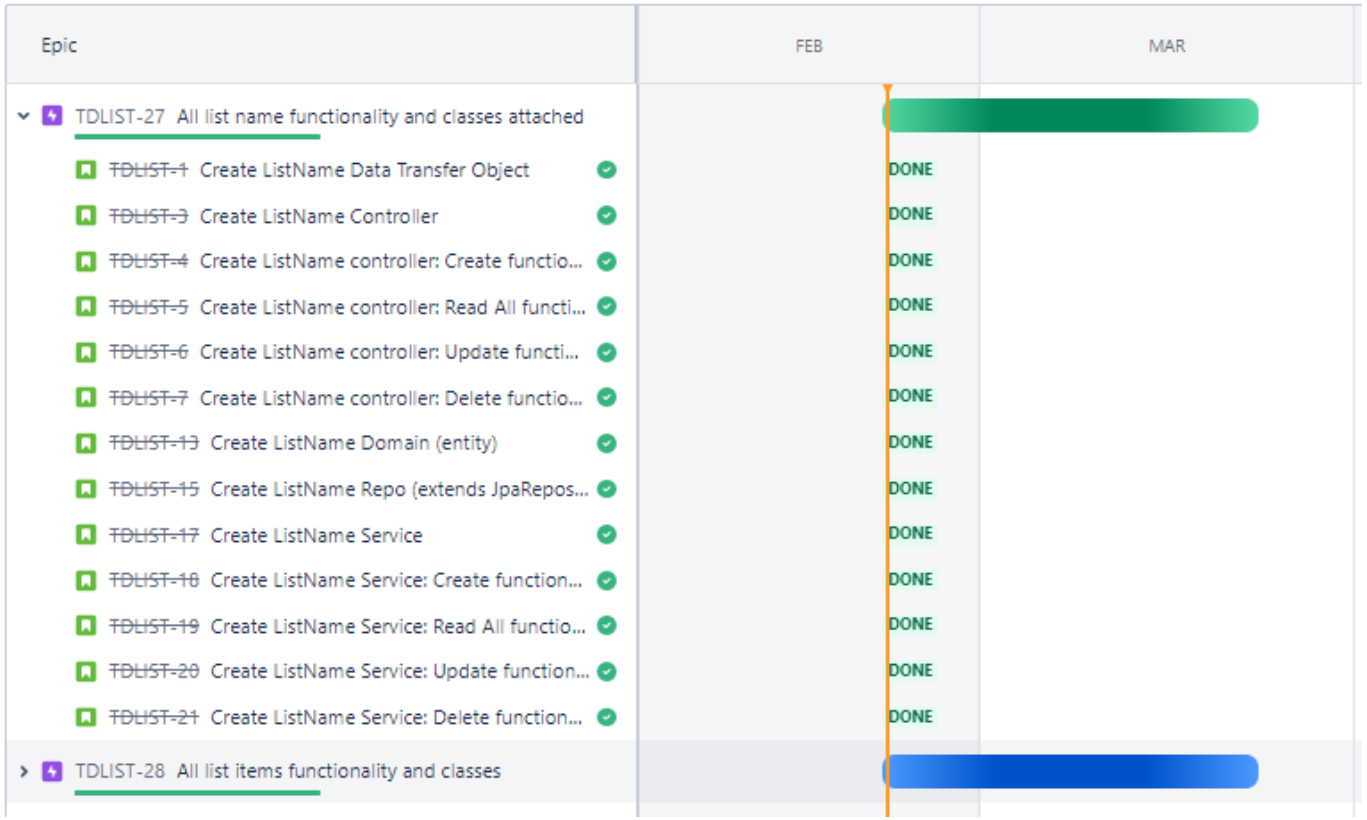
Throughout the development of this application, I ensured that new updates to the source code would not destroy the whole program by utilizing the Feature Branch model. Hence, starting with the original forked repo for the starting point of this project, I branched off of the main branch into my own, custom dev branch. Thereon, anytime I wanted to create new functionality based on the user stories I had created, I would branch off of those, name the branch based on the key of my Jira board user story (for link and integration purposes), and then merge it back in to the dev branch once I know I have made progress.

Example of the Version Control merging process.

- Git checkout dev -> git checkout TDL-25_xxx -> (make changes) -> git add .
- (etc)

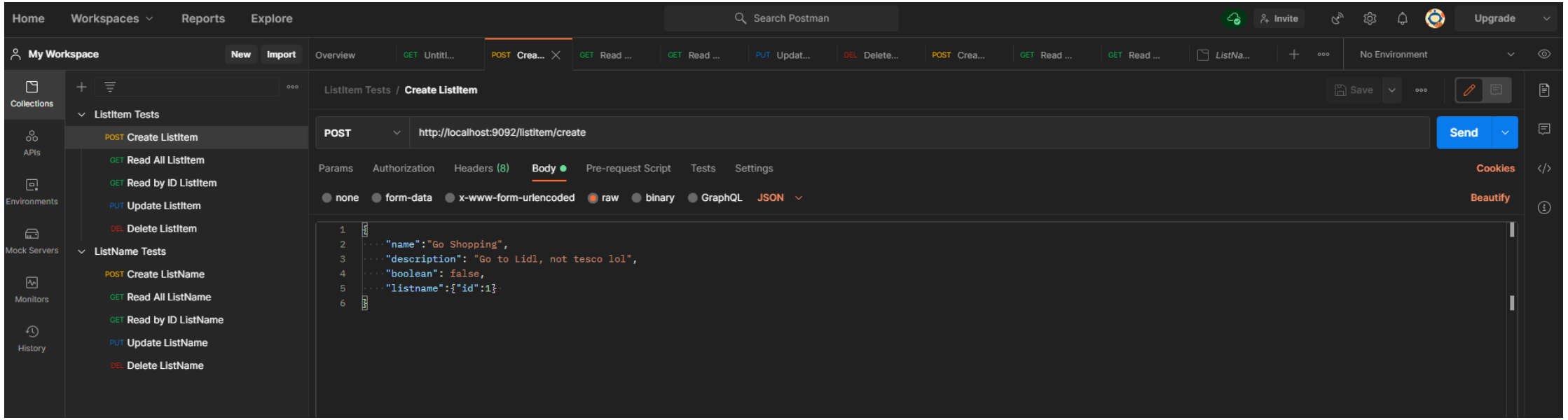
My Project Roadmap

Here I show the two epics I created for the entities. I linked my Jira with my GitHub repo for this project for naming purposes to attach each commit to their respective user story; utilizing the user story conventional structure i.e. as a (…), I want to (…), so that I can (…).



Interacting with the API using Postman

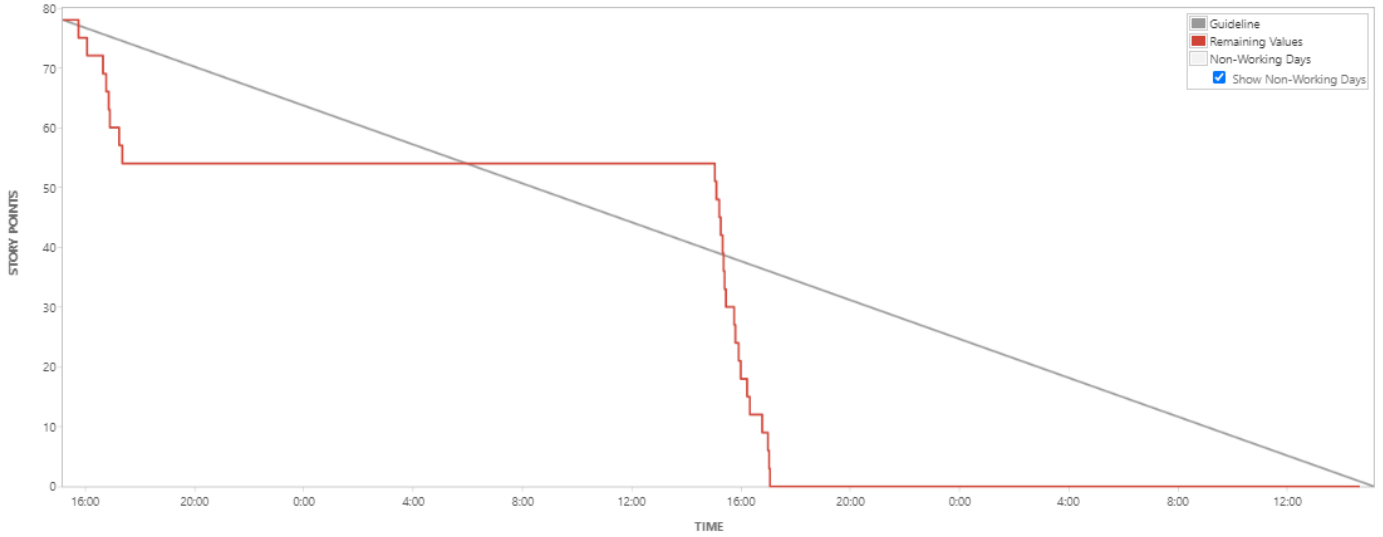
An illustration of how I communicated with my application using Postman to send HTTP Requests and the raw JSON body I send in a POST Request (create)



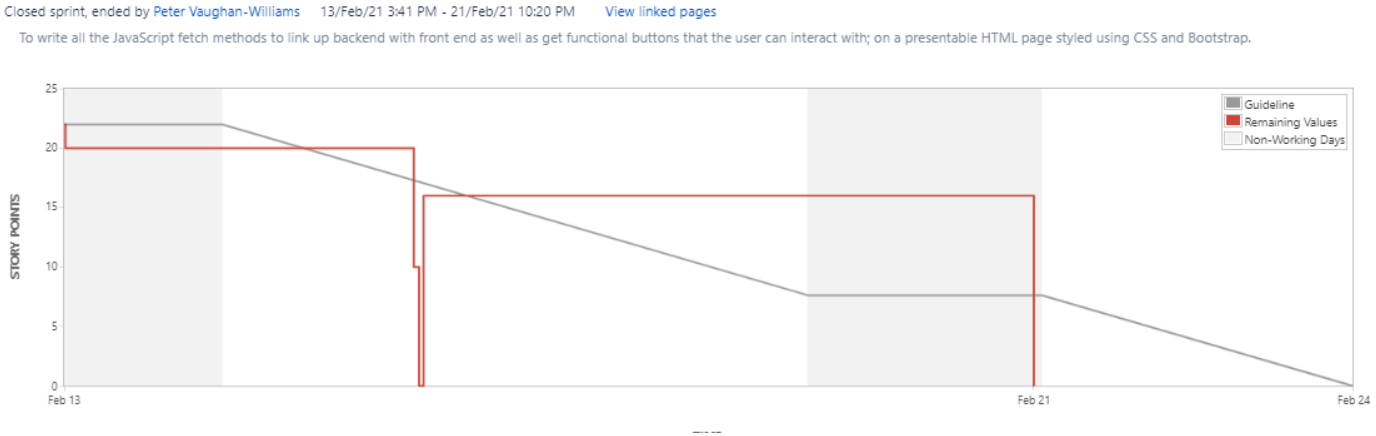
My Burndown Chart

Here are two of my Burndown Charts: My front-end sprint and my back-end sprint. You can see the rate at which I was able to complete the tasks/user stories I had created and assigned story points. Here's a link to my Jira board: https://pvaughan-williams.atlassian.net/secure/RapidBoard.jspa?rapidView=6&projectKey=TDLIST

Backend Sprint



Frontend Sprint



Testing

TDL-Project	84.6 %	2,795	510	3,305
src/main/java	60.9 %	741	476	1,217
src/test/java	98.4 %	2,054	34	2,088

As you can see, I tested for 84.6% of the total project.

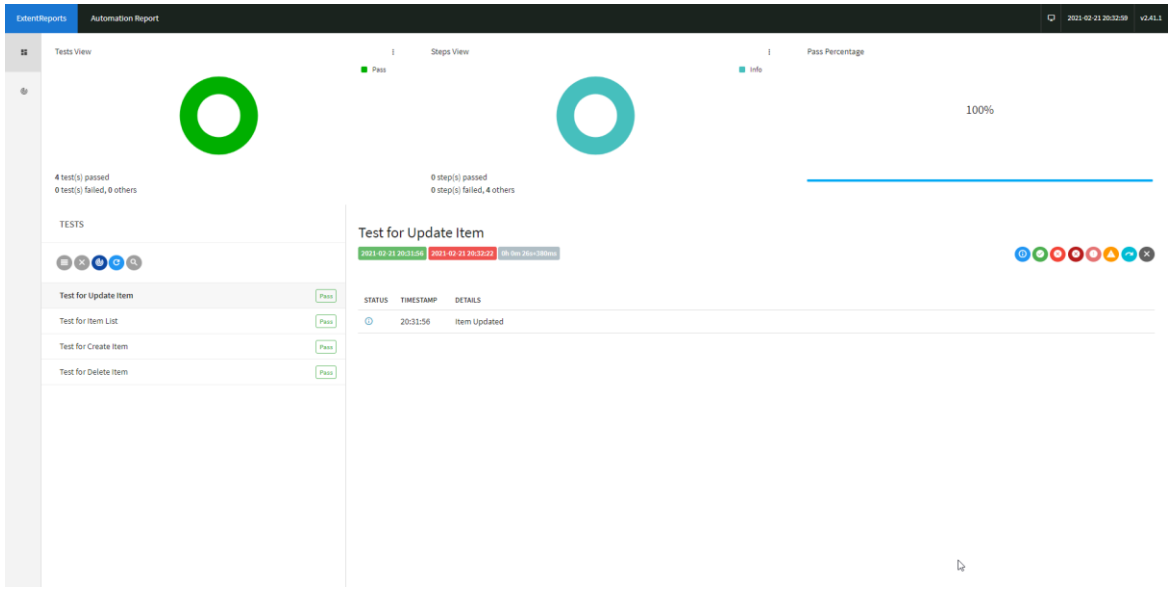
Console | Terminal | Coverage | JUnit

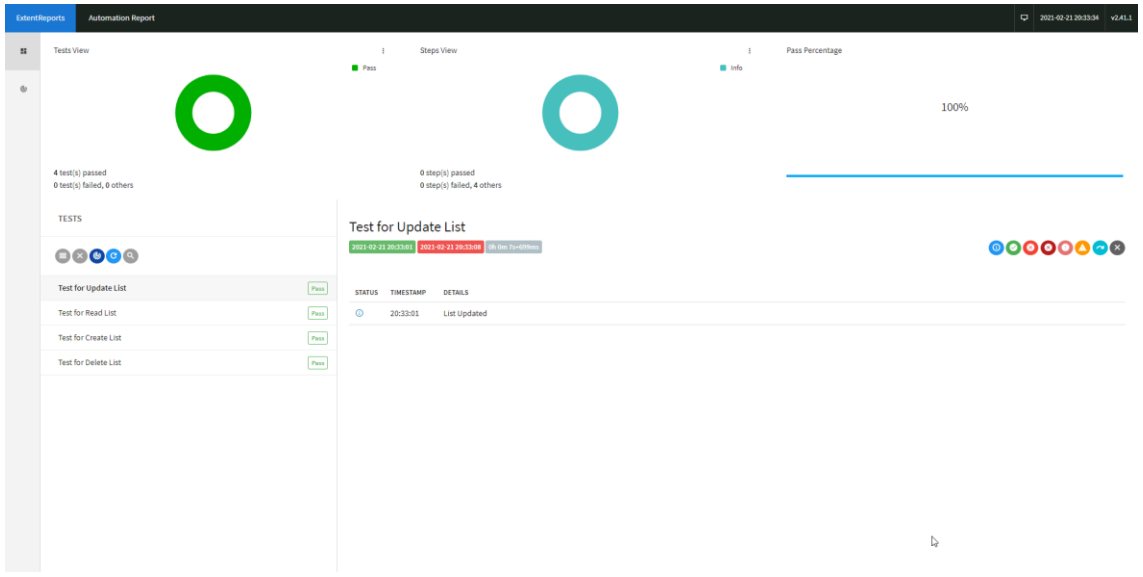
TDL-Project (21 Feb 2021 20:33:35)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src/main/java	60.9 %	741	476	1,217
com.qa.application.persistence.domain	48.9 %	251	262	513
com.qa.application.persistence.dto	47.7 %	177	194	371
com.qa.application.utils	84.6 %	44	8	52
com.qa.application	37.5 %	3	5	8
com.qa.application.service	97.4 %	150	4	154
com.qa.application.exceptions	50.0 %	3	3	6
com.qa.application.config	100.0 %	7	0	7
com.qa.application.controller	100.0 %	106	0	106
src/test/java	98.4 %	2,054	34	2,088

Testing 100% of the Config and Controller class, as well testing 97.4% of the Service class; making use of a combination of Unit and Integration testing to reach these numbers.

Extent Reports Generated





These are the extent reports I generated for each entity to ensure that I was able to display the tests passing, making sure that on the user side the application was functional.

Now for a Live Demonstration!

Sprint Review

- What did I complete?

I was able to complete the full CRUD functionality for both entities, with a backend developed in Spring written in Java, a working front end HTML page styled using CSS and Bootstrap, with buttons and modals interacted with using JavaScript.

- What got left behind?

I wanted to take some more time to perhaps create some custom methods for the project, as well as some more time on styling the HTML page to make the user interface even nicer for the user.

Sprint Retrospective

- What went well?

Building the backend and the framework for the HTML page went well, I found a systematic way of approaching both of those tasks and ensuring that I made a robust backend that could deal with the CRUD functionality of both entities as well as the creation of all the buttons and modals that would interact with the fetch methods I created using JavaScript.

- What got left behind?

At the end of the project, I figured out how to get text to display to the HTML page and was able to implement it for the create function for the first entity. With more time, I would like to implement it for the rest of the functionality in both entities. I would have also liked to implement the page object model in my acceptance testing as I had written my acceptance tests and gotten them running and passing, but not structured it in the way I would have wished to.

Conclusion

- Reflections on the project, future steps, any other relevant info:

I would like to mention at the end of this project that I felt I was able to manage my time effectively but you should still allocate more time to debugging around the end of the project time for any new features that you may have introduced later on in the project; as I didn't have enough time for it and wished I could have added on a little bit more on the front-end. I tried to rush more debugging at the end and felt a bit burned out at the end of the project, but I was able to complete it and feel proud of what I had accomplished.

Other points:

- Take more time with the setup of the sprints in order to have a less skewed burndown chart.

Thank you for your time!

Any questions?