TO-DO LIST WEB APPLICATION

LEIGHTON MANNING

CONCEPT

- Second project, focusing on the MVP and what was a must.
- Decided to plan the backend so that I could design the front end accordingly.
- Had a look at a couple of ideas online.

SPRINT PLAN

- Knew there needed to be two parts so I decided that I'd split the sprints into two as mentioned before with back-end first.
- By the end of the sprints I was hoping to achieve a fully working to-do list creator with both front and back end.

CONSULTANT JOURNEY

- Technologies used before.
- Java, Git, JUNIT+Mockito, Maven

- New technologies used this time
- Spring, HTML, CSS, JavaScript, Selenium, Sonarqube, Postman/SwaggerUI

CONTINUOUS INTEGRATION

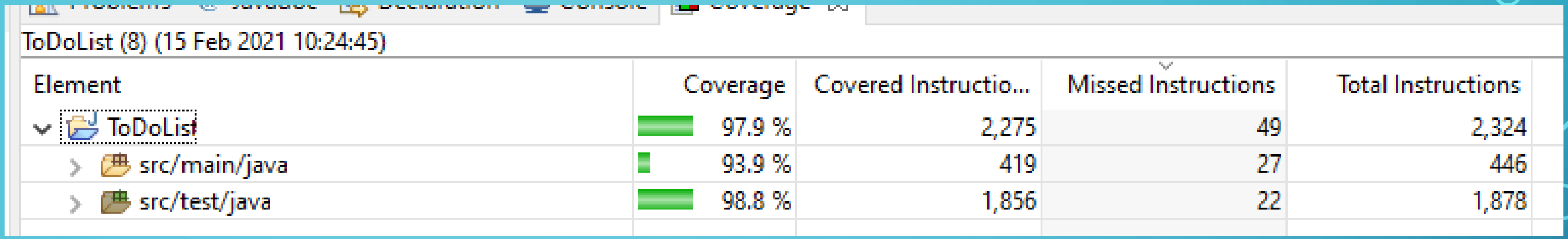
- Using Git for my version control using git bash
- Utilised the feature branch model, using main dev and feature-”concept”.

- Once features are complete merged back into dev. Finally into main.

- Was good to be able to split front/back/testing into different branches.

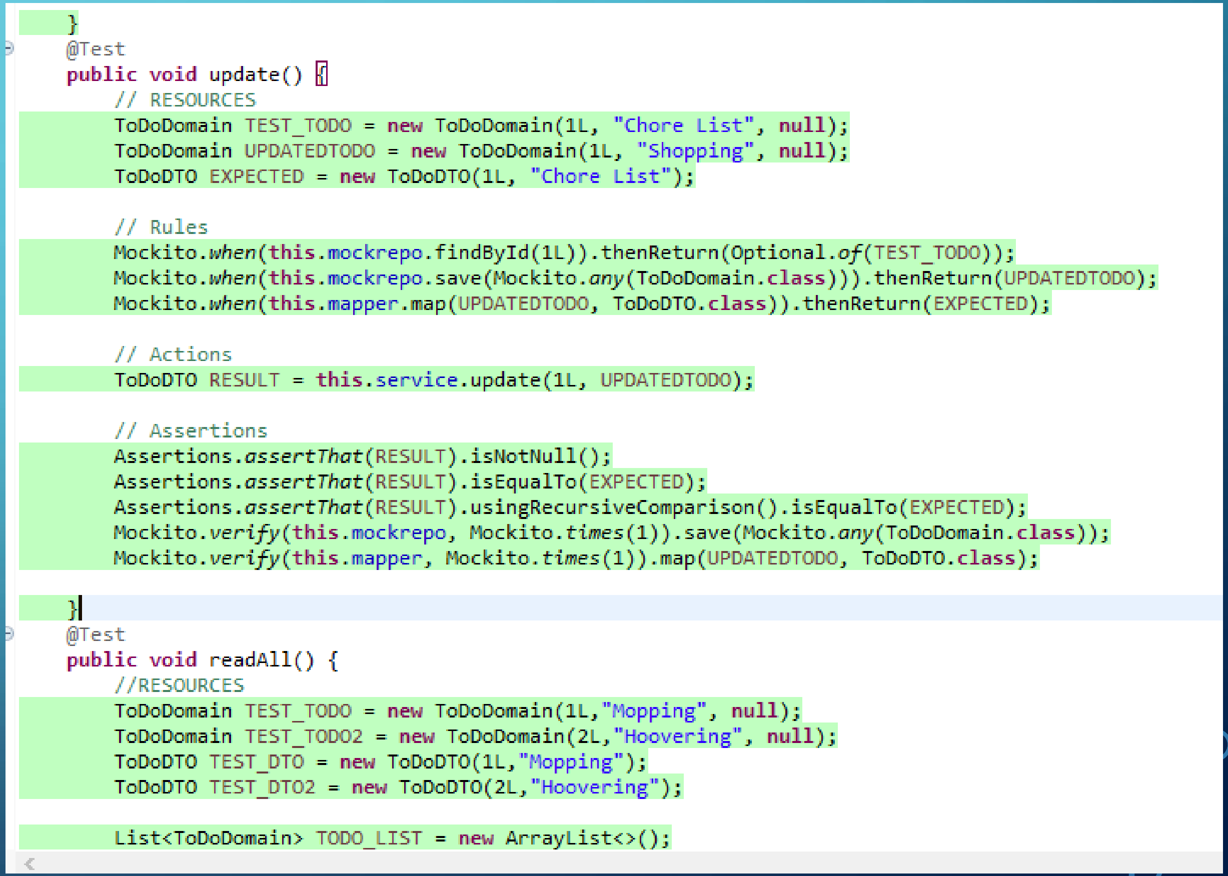
TESTING

- **JUNIT TESTING.**
- Testing my service classes for Tasks and ToDos.
- Focus point for me this time.



ToDoList (8) (15 Feb 2021 10:24:45)

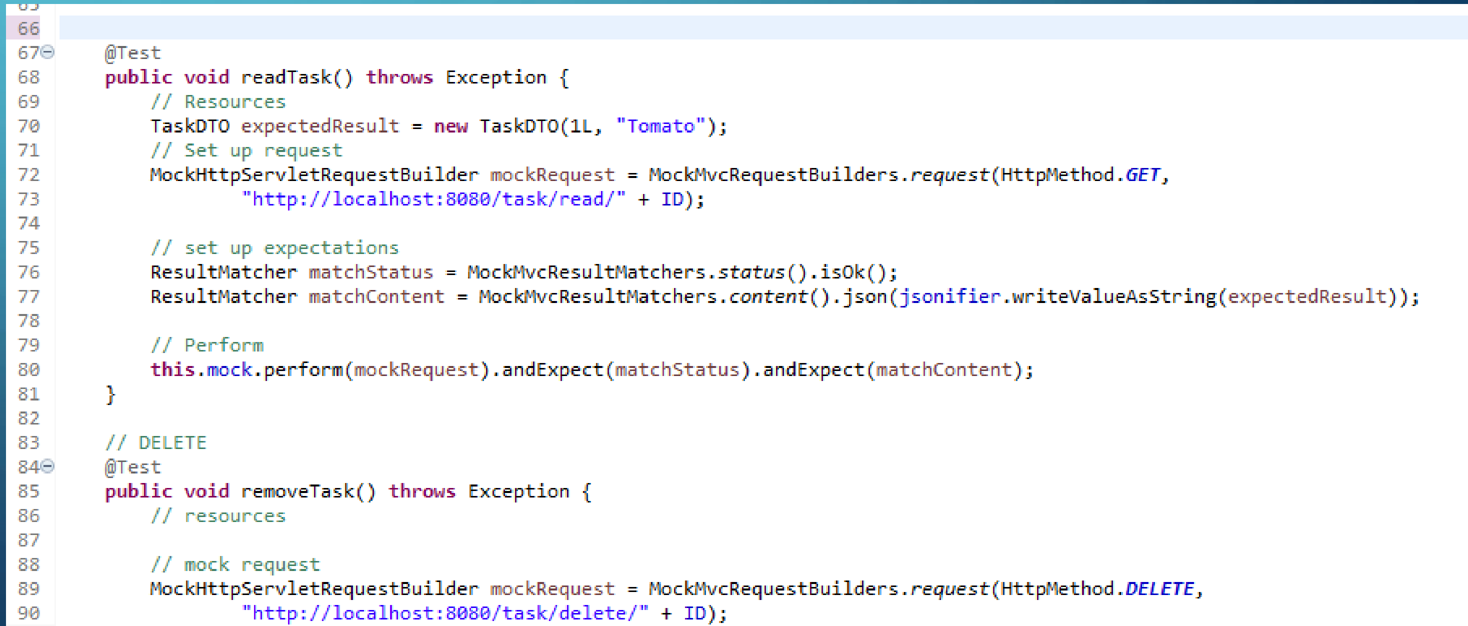
Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
ToDoList	97.9 %	2,275	49	2,324
src/main/java	93.9 %	419	27	446
src/test/java	98.8 %	1,856	22	1,878



```
    }
    @Test
    public void update() {
        // RESOURCES
        ToDoDomain TEST_TODO = new ToDoDomain(1L, "Chore List", null);
        ToDoDomain UPDATEDTODO = new ToDoDomain(1L, "Shopping", null);
        ToDoDTO EXPECTED = new ToDoDTO(1L, "Chore List");

        // Rules
        Mockito.when(this.mockrepo.findById(1L)).thenReturn(Optional.of(TEST_TODO));
        Mockito.when(this.mockrepo.save(Mockito.any(ToDoDomain.class))).thenReturn(UPDATEDTODO);
        Mockito.when(this.mapper.map(UPDATEDTODO, ToDoDTO.class)).thenReturn(EXPECTED);

        // Actions
        ToDoDTO RESULT = this.service.update(1L, UPDATEDTODO);

        // Assertions
        Assertions.assertThat(RESULT).isNotNull();
        Assertions.assertThat(RESULT).isEqualTo(EXPECTED);
        Assertions.assertThat(RESULT).usingRecursiveComparison().isEqualTo(EXPECTED);
        Mockito.verify(this.mockrepo, Mockito.times(1)).save(Mockito.any(ToDoDomain.class));
        Mockito.verify(this.mapper, Mockito.times(1)).map(UPDATEDTODO, ToDoDTO.class);

    }
    @Test
    public void readAll() {
        //RESOURCES
        ToDoDomain TEST_TODO = new ToDoDomain(1L,"Mopping", null);
        ToDoDomain TEST_TODO2 = new ToDoDomain(2L,"Hoovering", null);
        ToDoDTO TEST_DTO = new ToDoDTO(1L,"Mopping");
        ToDoDTO TEST_DTO2 = new ToDoDTO(2L,"Hoovering");

        List<ToDoDomain> TODO_LIST = new ArrayList<>();
```

TESTING CONTINUED

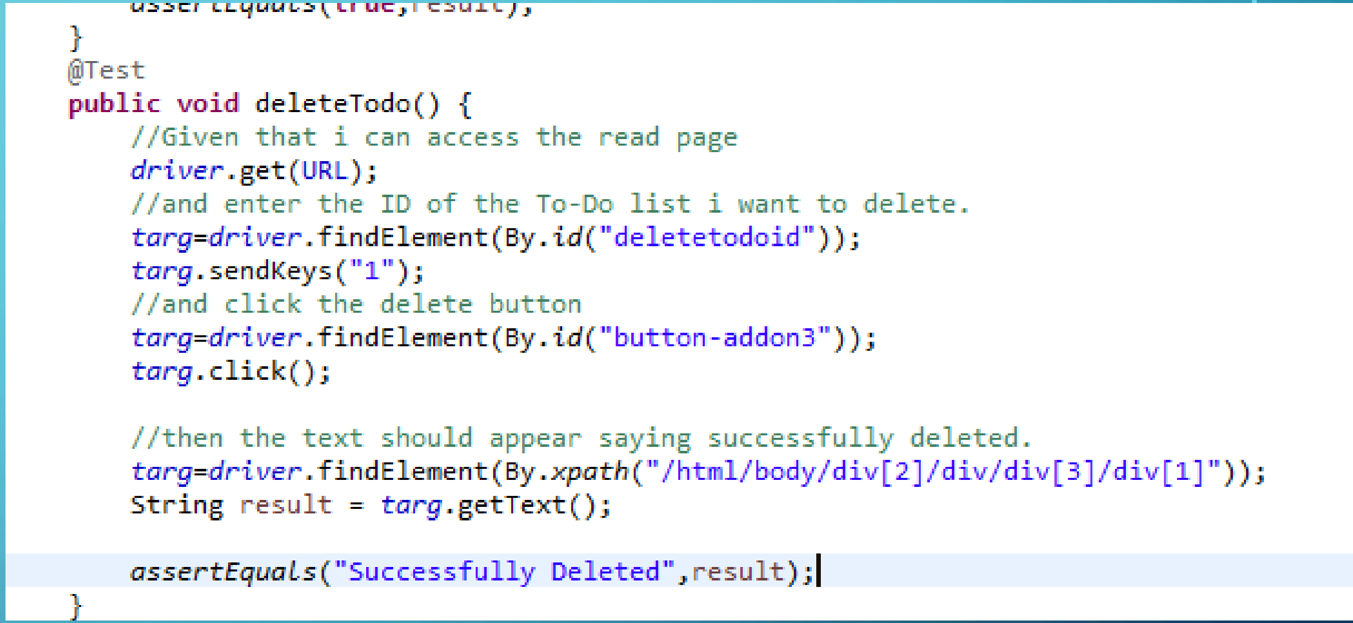
- **Integration Testing**
- To prove that each integration of the application is functioning as expected
- Done on the Controller classes.
- Using Junit+Mockito



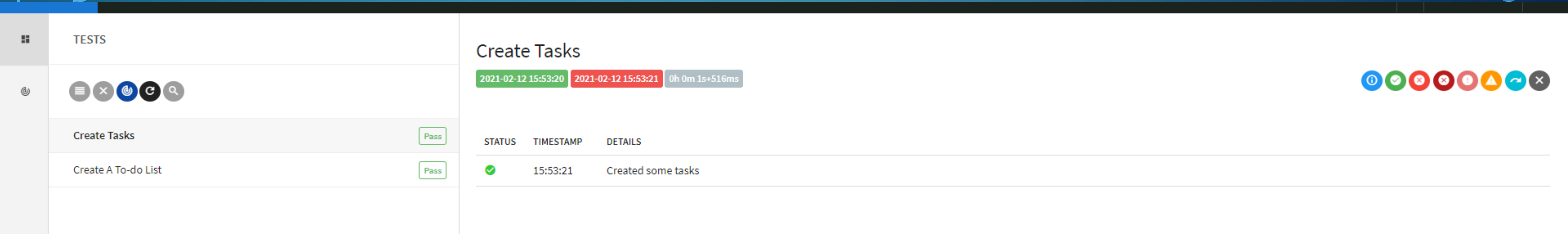
```
@Test
public void readTask() throws Exception {
    // Resources
    TaskDTO expectedResult = new TaskDTO(1L, "Tomato");
    // Set up request
    MockHttpServletRequestBuilder mockRequest = MockMvcRequestBuilders.request(HttpMethod.GET,
            "http://localhost:8080/task/read/" + ID);

    // set up expectations
    ResultMatcher matchStatus = MockMvcResultMatchers.status().isOk();
    ResultMatcher matchContent = MockMvcResultMatchers.content().json(jsonifier.writeValueAsString(expectedResult));

    // Perform
    this.mock.perform(mockRequest).andExpect(matchStatus).andExpect(matchContent);
}

// DELETE
@Test
public void removeTask() throws Exception {
    // resources

    // mock request
    MockHttpServletRequestBuilder mockRequest = MockMvcRequestBuilders.request(HttpMethod.DELETE,
            "http://localhost:8080/task/delete/" + ID);
```

TESTING CONTINUED

- **Selenium**
- Testing my front end using selenium to automate the web browser
- Testing the front-end works as expected
- Added an extent report.



```
        assertEquals(true,result);
    }
    @Test
    public void deleteTodo() {
        //Given that i can access the read page
        driver.get(URL);
        //and enter the ID of the To-Do list i want to delete.
        targ=driver.findElement(By.id("deletetodoid"));
        targ.sendKeys("1");
        //and click the delete button
        targ=driver.findElement(By.id("button-addon3"));
        targ.click();

        //then the text should appear saying successfully deleted.
        targ=driver.findElement(By.xpath("/html/body/div[2]/div/div[3]/div[1]"));
        String result = targ.getText();

        assertEquals("Successfully Deleted",result);
    }
```



TESTS

Create Tasks	Pass
Create A To-do List	Pass

Create Tasks

2021-02-12 15:53:20 2021-02-12 15:53:21 0h 0m 1s+516ms

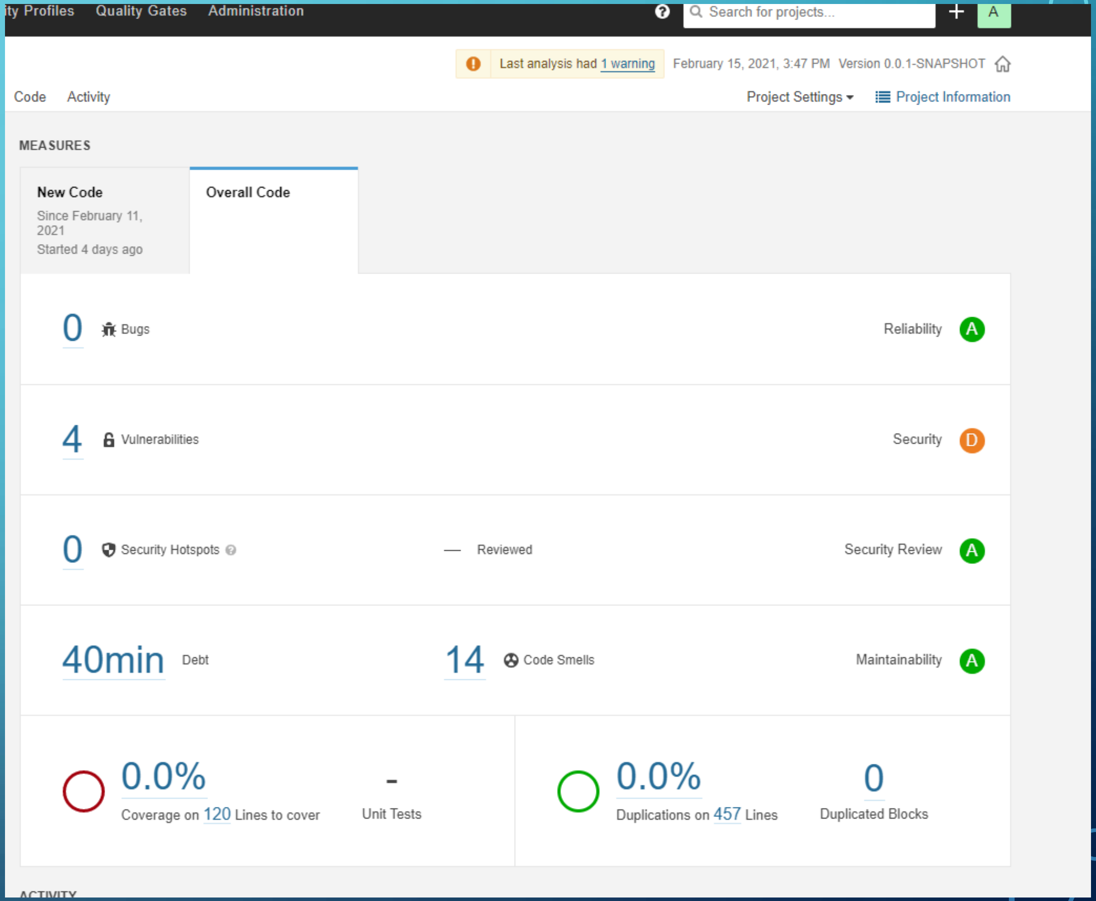
STATUS	TIMESTAMP	DETAILS
✓	15:53:21	Created some tasks

SONARQUBE

- Used for static analysis of my code.
- Makes sure that bugs and security issues are caught

- Using best practice

- Worked to remove code smells and bugs.



Last analysis had 1 warning February 15, 2021, 3:47 PM Version 0.0.1-SNAPSHOT

MEASURES

New Code
Since February 11, 2021
Started 4 days ago

Overall Code

0 Bugs — Reliability A

4 Vulnerabilities — Security D

0 Security Hotspots — Reviewed — Security Review A

40min Debt — 14 Code Smells — Maintainability A

0.0% Coverage on 120 Lines to cover — - Unit Tests

0.0% Duplications on 457 Lines — 0 Duplicated Blocks

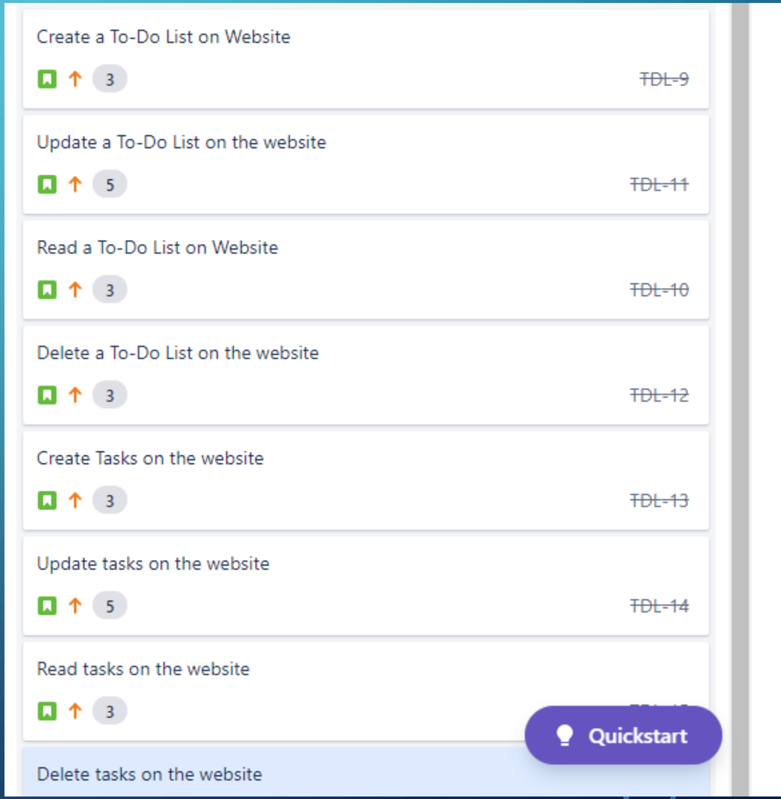
ACTIVITY

DEMONSTRATION

- I’m going to run through a couple of user stories.

SPRINT REVIEW

- Using Jira
- I completed all the user stories I set for the project.
- Worked on the backend first, then moved to front end as planned
- Estimated using story points
- Wanted to customise the front-end more. More Javascript.



Create a To-Do List on Website
3 ~~TDL-9~~

Update a To-Do List on the website
5 ~~TDL-11~~

Read a To-Do List on Website
3 ~~TDL-10~~

Delete a To-Do List on the website
3 ~~TDL-12~~

Create Tasks on the website
3 ~~TDL-13~~

Update tasks on the website
5 ~~TDL-14~~

Read tasks on the website
3

Delete tasks on the website

SPRINT RETROSPECTIVE

- Both sprints went well. Completed in time and they focused on what my program MUST have via MoSCoW
- Some of the story points I gave the tasks where a little on the low side and took longer than expected.
- Overall I managed my time well, will work on improving user story estimations.

CONCLUSION

- I think overall the project went well. Using lots of new technologies but also building on the knowledge that I learnt from the previous project.
- Future steps would be to continue developing my skills using both front end and backend technologies going forward. Using these towards future projects in the academy and for myself.

QUESTIONS

Thanks for listening