To-Do Project

By Mohamud Mussa

Introduction

- First Class Degree in Computer Science.
- Background in SQL.
- Problem Solving.
- Creativity is everything to me.
- *Web Design, Graphic Design, Documentary Making. YouTube.*

Project Summary

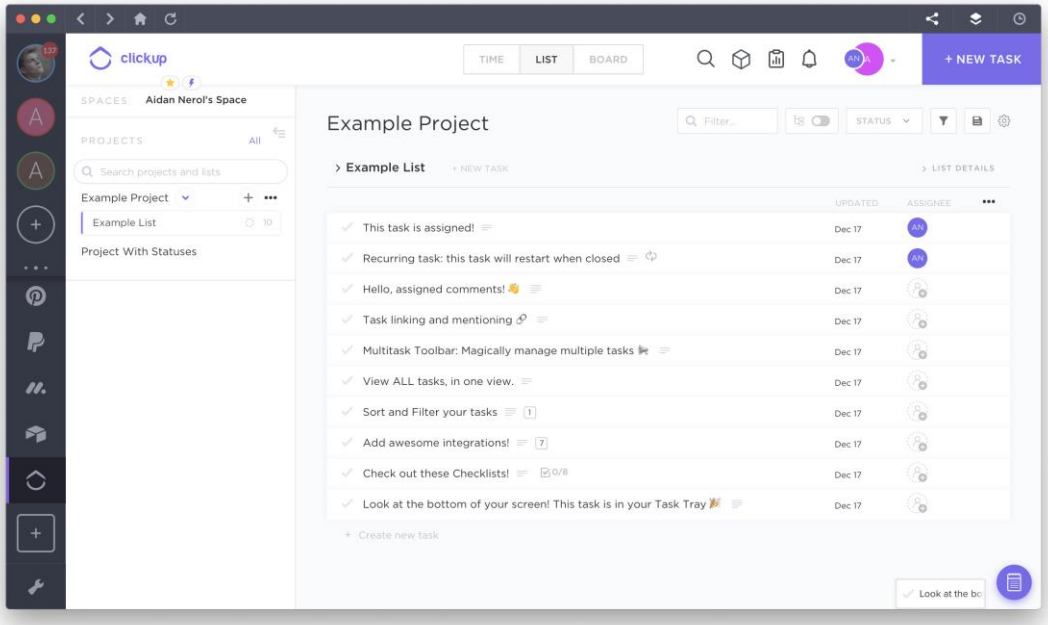
- To Create a full stack web application using the 3 tier architecture system.
- The Application is a To-Do List which Allows users to Create, Read, Update, Delete their tasks and interact with the website.

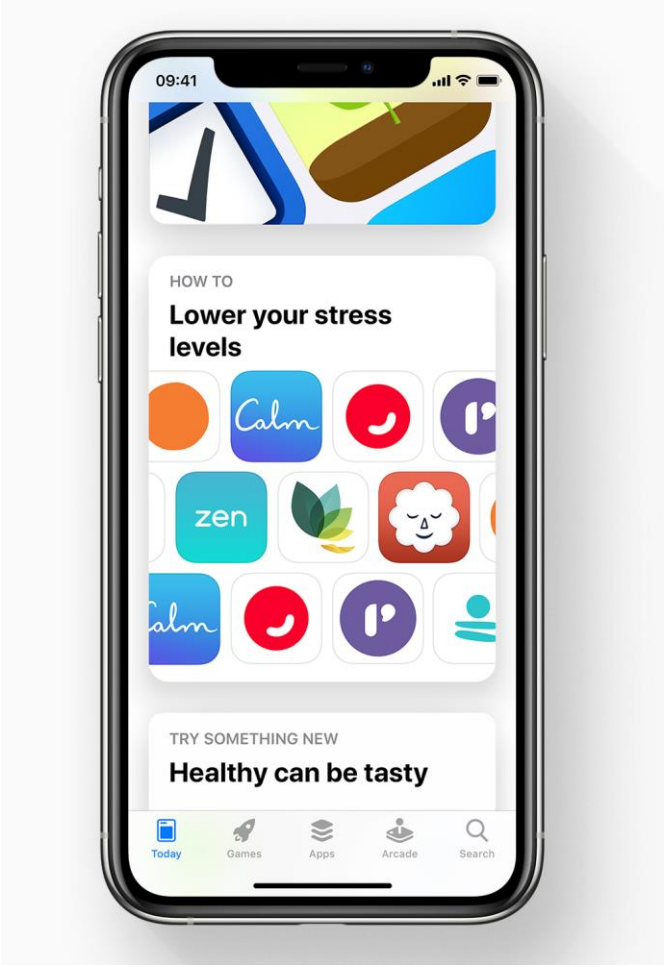
How did I approach the project specification?

1. Understanding the project
2. Developing a Risk Assessment from that understanding
3. Breaking down the MVP
4. Initial design process – ERD - UML
5. Writing the logical approach to the project down.
6. Attempt to tackle the project

Personalising the project

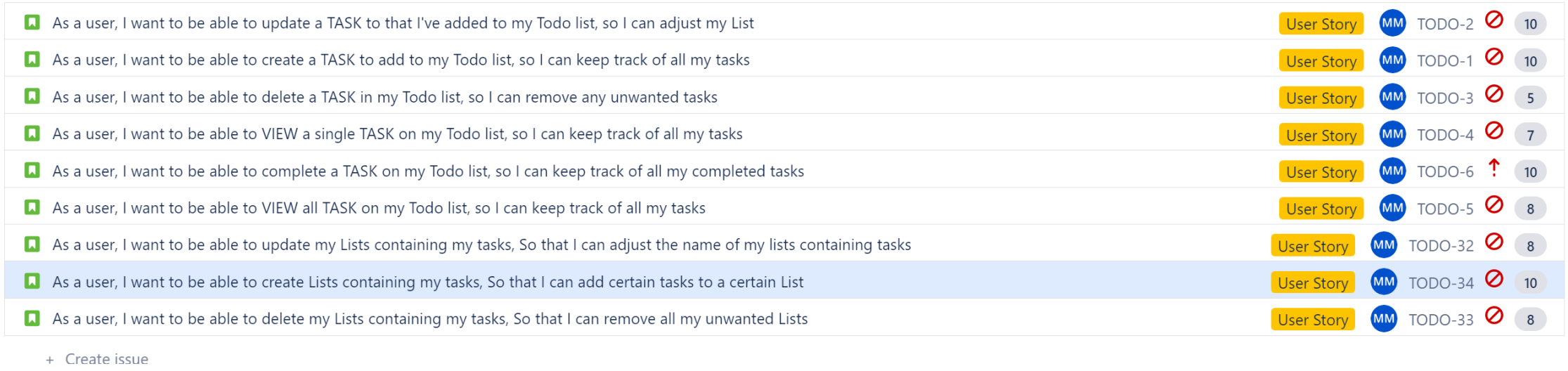
- Real world issue
- Tackling existing issues
- Avid user





Understanding

- Understanding the MVP specification and creating User stories and tasks based on them.



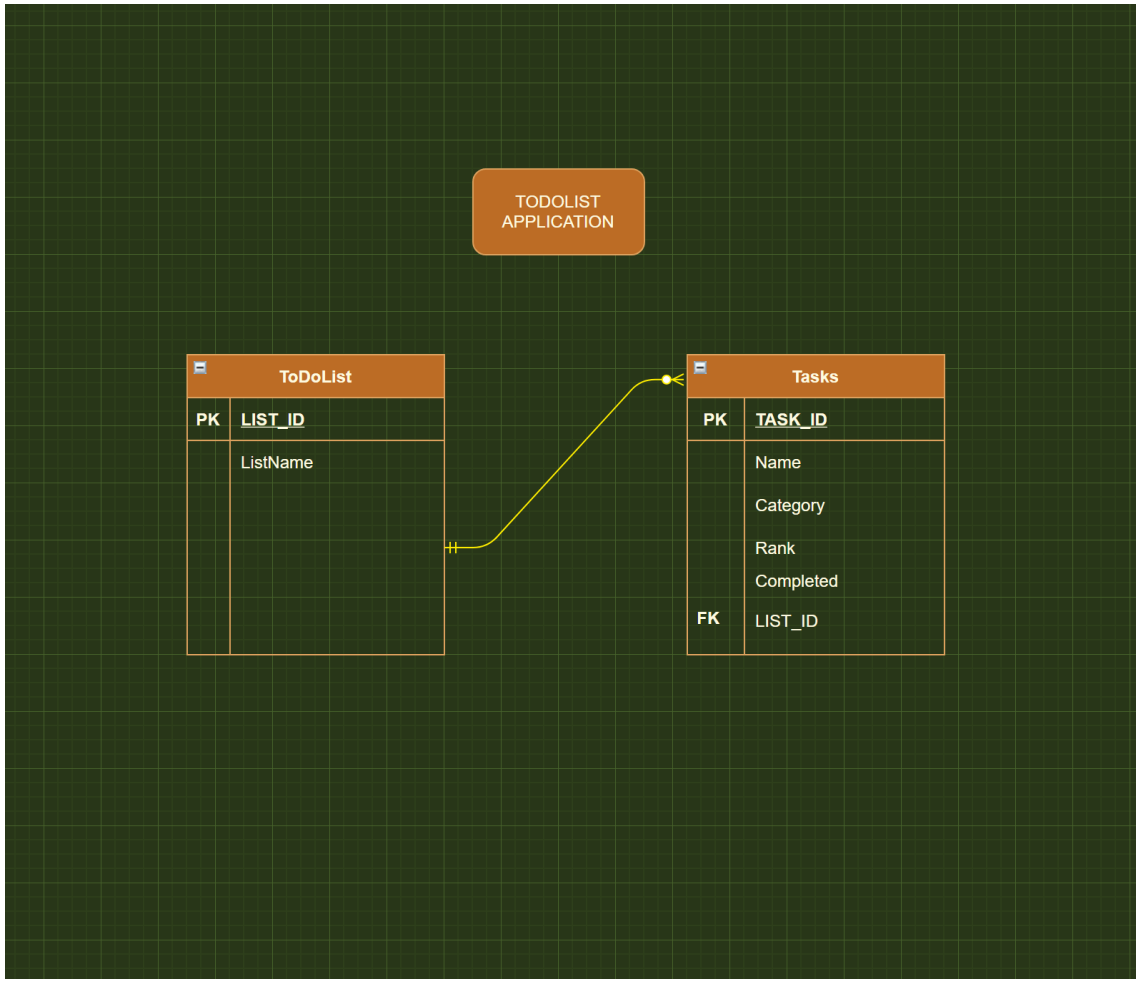
Risk Management

- Creating a Risk Assessment based on that.
- Looking at project with a new perspective based on previous project.
- Time management using Jira Software to better plan my Sprints.

Computer Crashing & Losing the data	My Laptop might crash for various reason.		/		/			This will impact the project as I would lose all progress made with this laptop.	N/A	I was given a QA community Laptop as well and this will allow me to switch from my personal Laptop to the QA one if necessary as my personal laptop is what I'm currently using. I would then fork my repository from GitHub and continue the work on there
Covid-19	Contract Covid-19 or a family member getting Covid-19		/			/		This can impact the project in many ways. One being that it would affect my ability to work on the project. If COVID-19 was to happen to a family member I'd need to take care of them whilst working on this project too.	A week	While self isolating I can still use my laptop although I might need time to recover.
Schedule Risk	Not completing the project on time	/					/	Wrong time estimation Resources are not tracked properly. All resources like staff, systems, skills of individuals, etc. Failure to identify complex functionalities and time required to develop those functionalities. Unexpected project scope expansions.	2 Weeks overall project sprint	On agile projects the team is heavily involved in planning and estimating through activities such as XP's planning game and Wideband Delphi workshops. By working in short increments the true velocity of the team quickly emerges and is visible to all stakeholders who are now more closely involved in the project. In short, the true progress is hard to hide and quickly revealed, giving feedback to the stakeholders.
Git Merge Conflicts	I could have a merge conflicts or overwrite by merging incorrectly.	/					/	This could impact the project majorly as I would have to start from where the overwrite took place. I would also need to resolve a conflict merge issue	A day / 2	Constantly update my GitHub by doing commits regularly. Also

Design

- Designed my ERD to better understand the structure of the database I wanted to create. Which would store users Tasks and their custom Lists containing tasks.



Application

- This would later be used to code my Domain Class in order to set up my Many To One relationships in the application.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@NotNull
private Long id;

@NotNull
private String name;

@NotNull
private String category;

@NotNull
private Integer rank;

@NotNull
private Boolean completed;

@ManyToOne
private ToDoDomain myToDo;
```

Sprint Plan

As a user, I want to be able to create a TASK to add to my Todo list, so I can keep track of all my tasks	User Story	MM	TODO-1
As a user, I want to be able to update a TASK to that I've added to my Todo list, so I can adjust my List	User Story	MM	TODO-2
As a user, I want to be able to delete a TASK in my Todo list, so I can remove any unwanted tasks	User Story	MM	TODO-3
As a user, I want to be able to VIEW a single TASK on my Todo list, so I can keep track of all my tasks	User Story	MM	TODO-4
As a user, I want to be able to VIEW all TASK on my Todo list, so I can keep track of all my tasks	User Story	MM	TODO-5
As a user, I want to be able to complete a TASK on my Todo list, so I can keep track of all my completed tasks	User Story	MM	TODO-6
INDEXT PAGE HTML	Front End	MM	TODO-29
INDEX PAGE CSS	Front End	MM	TODO-30
INDEX PAGE JS	Front End	MM	TODO-31
As a user, I want to have different Lists / Categories, so that I can put certain todos in their given categories	List Page	MM	TODO-9
Create TasksDomain	Tasks Page	MM	TODO-11
Create Application Config	Application	MM	TODO-10
Create ToDoListDomain	List Page		TODO-12
Create TasksDTO	Tasks Page	MM	TODO-13
Create ToDoListDTO	List Page		TODO-14
Create TasksController	Tasks Page	MM	TODO-15
Create ToDoListController	List Page		TODO-16
CreateTaskService	Tasks Page	MM	TODO-17
Create ToDoList Service	List Page		TODO-18
Create ToDoList Application	Application	MM	TODO-19
Tasks Integration Testing	Spring Testing	MM	TODO-20
Tasks Unit Testing	Spring Testing	MM	TODO-21
ToDoList Unit Testing	Spring Testing	MM	TODO-22
ToDoList Application testing	Spring Testing	MM	
ToDoList Controller testing	Spring Testing	MM	TODO-24

Technologies

The technologies I have learnt for this project;

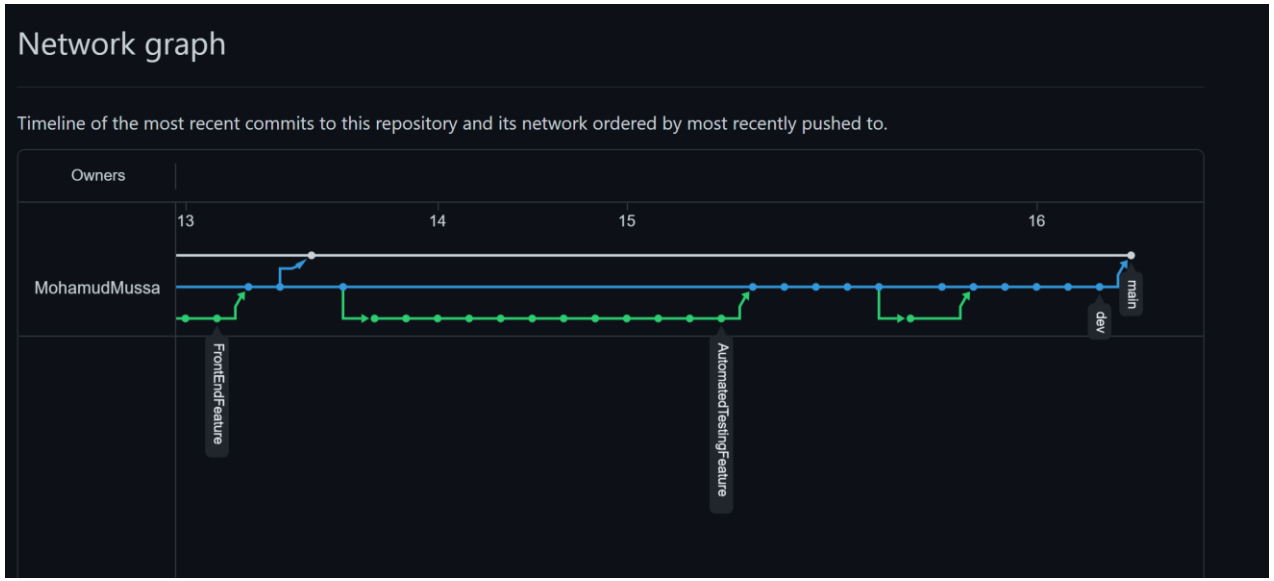
- Git – Version Control
- Markdown – Used to create README
- H2 is a relational database management system
- Spring Boot – Used to code the backend of the To-do Project (Java)
- HTML/CSS/JAVASCRIPT – was used to create the front end of the application.
- Jira – Used to create the two sprints and keep track of user stories and tasks
- Maven – Used to download Java libraries and Maven plugins & store them in a local cache
- Testing (Mockito / Junit / Selenium)
- Postman – API development view. Alongside SWAGGER
- SonarQube – to analyse code against best practice standards and then improve the code.

Continuous Integration

GitHub was used for Version Control and continuous integration. This was done by using feature / dev concept model. Pull Request and Merging into Master

- **Automated testing feature**
  #8 merged yesterday
- **Dev**
  #7 merged 3 days ago
- **Front end feature**
  #6 merged 3 days ago
- **Dev**
  #5 merged 7 days ago
- **Testing feature**
  #4 merged 7 days ago

Splitting branches by Epics



Markdown README

```
Coverage: 94%

# ToDo List Project by Mohamud Mussa

This Projected is a full-stack web application which allows users to Create, Upda
order to prevent coupling. The frontend of the website was coded in JavaScript, H
alongside with SQL for the database layer. An API was developed using Spring in o
tested with Unit, Automated and Integration testing. Agile methodology was used t
on schedule.


## Table of contents
* [Prerequisite](#Prerequisite)
* [Installing](#Installing)
* [Testing](#Testing)
* [Creating JAR file](#Creating_JAR_file)
* [Deployment](#Deployment)
* [Built With](#Built_With)
* [Versioning](#Versioning)
* [Authors](#Authors)
* [License](#License)
* [Acknowledgments](#Acknowledgments)
```

Coverage: 94%

ToDo List Project by Mohamud Mussa

This Projected is a full-stack web application which allows users to Create, Update and Delete their Tasks. The project was created with the three-tier architecture in mind in order to prevent coupling. The frontend of the website was coded in JavaScript, HTML & CSS, Using Visual Studio Code as an IDE. The backend was coded using Spring (Java) alongside with SQL for the database layer. An API was developed using Spring in order to communicate between the frontend and the backend components. This project is fully tested with Unit, Automated and Integration testing. Agile methodology was used throughout this project alongside Kanban boards to ensure all the criteria were met and finished on schedule.

Table of contents

Database

H2 Database

Relational database management system which was embedded into the To Do application.

```
INSERT INTO TO_DO_DOMAIN (list_Name)
VALUES
('Helping'),
('Shopping'),
('Work'),
('Work'),
('Shopping'),
('House Items');


INSERT INTO TASK_DOMAIN (name, category, rank, completed, my_To_Do_id)
VALUES
('Buy Banana', 'Shopping', 2, false,1),
('Football', 'Sport', 3, true,2),
('Buy Phone', 'Shopping', 2, false,3),
('Clean', 'To Do', 3, true,4),
('Run', 'Health', 2, false,5),
('Finish Report', 'Work', 3, true,6);
```

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM TASK_DOMAIN
```

SELECT * FROM TASK_DOMAIN;

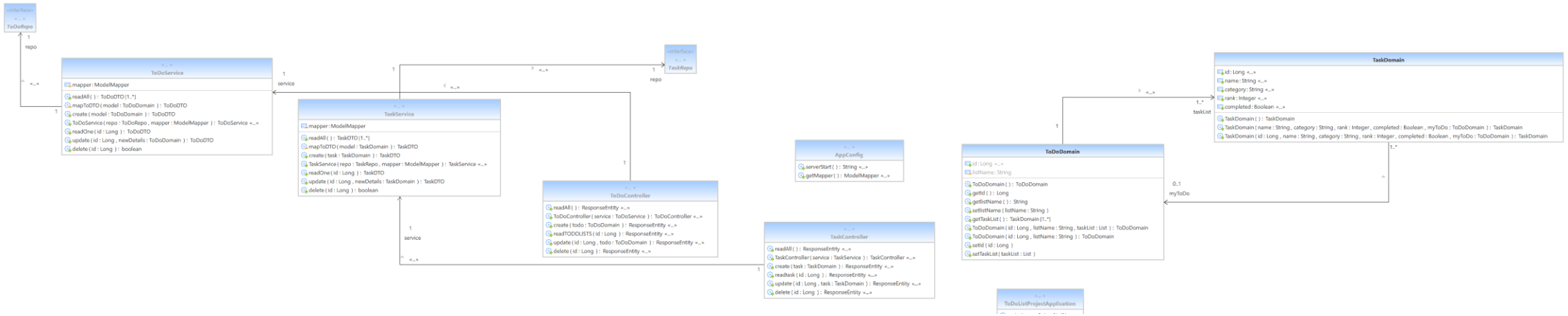
ID	CATEGORY	COMPLETED	NAME	RANK	MY_TO_DO_ID
1	Shopping	FALSE	Buy Banana	2	1
2	WORLD	FALSE	HELLP	1	1
3	Shopping	FALSE	Buy Phone	2	3
5	Health	FALSE	Run	2	5
6	Work	TRUE	Finish Report	3	6
7	QA	FALSE	Finish Presentation	5	1

(6 rows, 5 ms)

Edit

Back-end

Spring Boot



- API transfers the data between frontend and backend
- Controller - generates the http request
- Controller has a services - which stores the main logic of the application.
- Services calls on the repo which extracts from the database.
- Repo - DAO connection to database.

Domain

```java
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@NotNull
private Long id;

@NotNull
private String name;

@NotNull
private String category;

@NotNull
private Integer rank;

@NotNull
private Boolean completed;

@ManyToOne
private ToDoDomain myToDo;
```

DTO

```java
public class TaskDTO {

    private Long id;
    private String name;
    private String category;
    private Integer rank;
    private Boolean completed;
```

CONTROLLER

```java
@RestController
@RequestMapping("/task")
public class TaskController {

    private TaskService service;
```

SERVICE

```java
public List<TaskDTO> readAll() {
    List<TaskDomain> dbList = this.repo.findAll();
    List<TaskDTO> resultList = dbList.stream().map(this::mapToDTO).collect(Collectors.toList());
    return resultList;

}
```

Front-end

HTML

- Use of tables to organise the data received from the API.

- Simple
- Minimalistic
- User-friendly.

```
<table class="table table-dark table-hover">

    <thead>
        <tr>
            <th scope="col">ID</th>
            <th scope="col">To Do</th>
            <th scope="col">Category</th>
            <th scope="col">Rank</th>
            <th scope="col">Completed</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td id="taskID"></td>
            <td id="task"></td>
            <td id="category"></td>
            <td id="rank"></td>
            <td id="complete"></td>
        </tr>
        <tr>
```

JAVASCRIPT

- API request
- Creating A Task
- Using DOM through HTML
- Displaying the data consistently
- Refresh to update website seamlessly

```
//making my todolist
const idref = document.querySelector("#taskID");
const task = document.querySelector("#task");
const categoryRef = document.querySelector("#category");
const rankRef = document.querySelector("#rank");
const completeRef = document.querySelector("#complete");
```

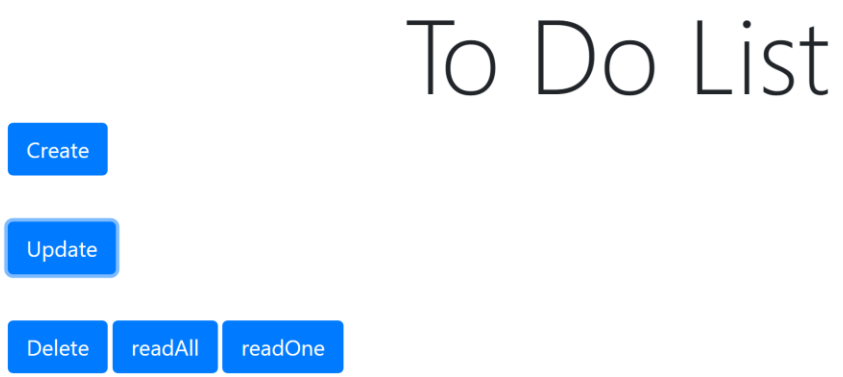
```
const createToDoList = () => {
    const taskname = stateTask.value;
    const category = stateCategory.value;
    const rank = stateRank.value;
    const completed = stateCompleted.value;
    const listvalue = stateListValue.value;

    let createdata = {
        name: taskname,
        category:category,
        rank: rank,
        completed: completed,
        myToDo: {id:listvalue}

    }

    fetch("http://localhost:8080/task/create", {
        method: "POST",
        body: JSON.stringify(createdata),
        headers: { "Content-Type": "application/json" }
    })
        .then(response => response.json())
        .then(info => {
            console.log(info);
            console.log("success");
            location.reload() //reloads the  info
        })
        .catch(err => console.error(`ERROR = ${err}`));
}
```

CSS

- Understanding design
- Minimalistic Design approach.

```
  @media (max-width:1919px) {
   .div {
     background-size: contain;
   }
  }

  * {

   background-color:■black;
   font-family: 'Courier New', Courier, monospace;
   text-align: center;
}

p, h3, h1, h2, h3, h4, h5, h6{
   color: ■white;
   text-align: center;
   background-color:■black;
}

.center-block {
   display: block;
   margin-left: auto;
   margin-right: auto;
 }
```

Initial Website



Demonstration

Testing

- Unit Testing

was used to check and test individual Units of code in order to validate whether code performs as expected

```
@Test
public void readOne() {

    TaskDomain test_task = new TaskDomain(1L, "Help Mum", "Helping", 5, false, null);
    TaskDTO test_dto = this.MockMapper.map(test_task, TaskDTO.class);

    //the rules
    Mockito.when(this.mockRepo.findById(test_task.getId())).thenReturn(Optional.of(test_task));

    //the task
    TaskDTO result = this.service.readOne(1L);


    Assertions.assertThat(result).isEqualTo(test_dto);
    //should only run once
    Mockito.verify(this.mockRepo, Mockito.times(1)).findById(1L);



}
```

Testing

- Integration Testing

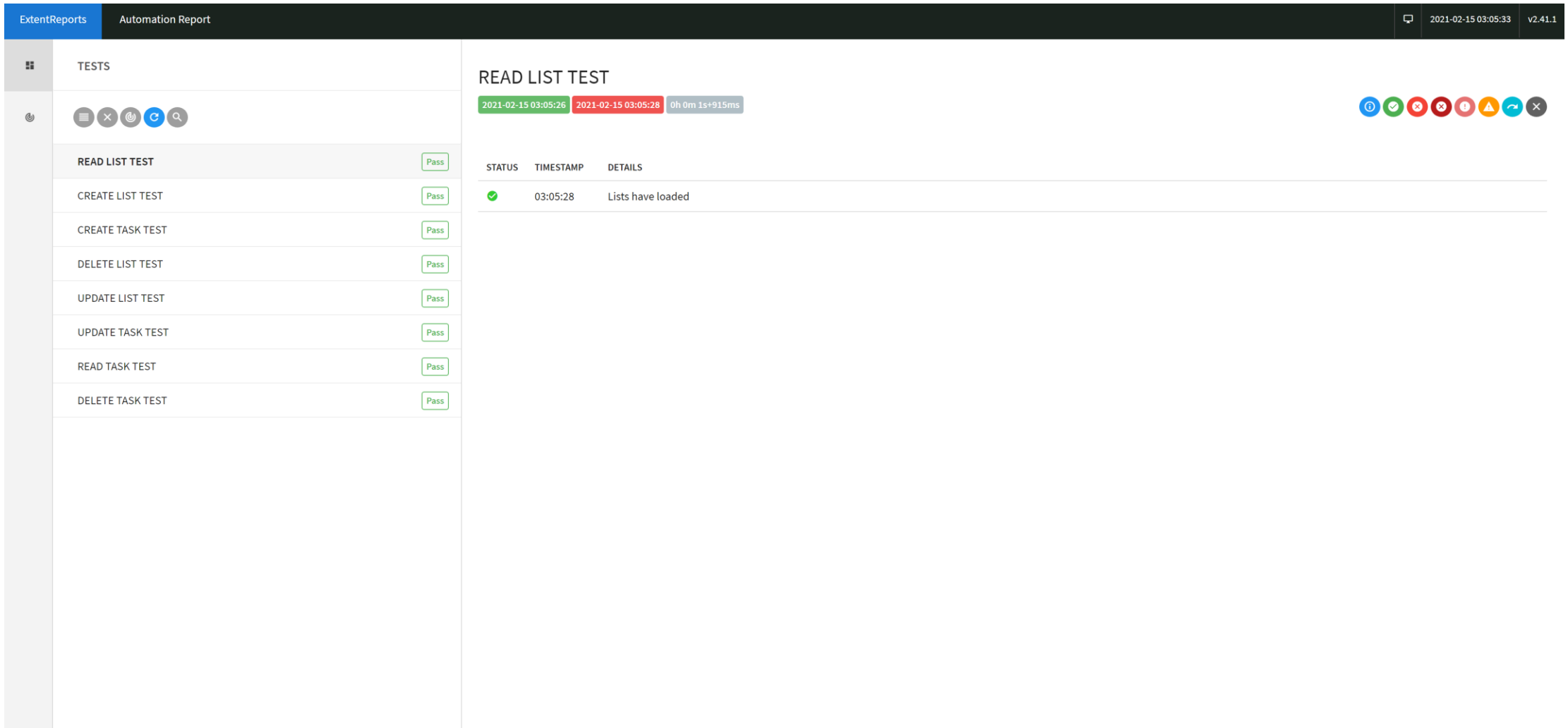
```
// read one task test
@Test
public void readTask() throws Exception {

    // EXPECTED INFO BACK WHICH WILL BE JSON
    TaskDTO expectedResults = new TaskDTO(1L, "Buy Banana", "Shopping", 2, false);

    // this sets up the request
    MockHttpServletRequestBuilder mockRequest = MockMvcRequestBuilders.request(HttpMethod.GET,
            "http://localhost:8080/task/read/" + 1L);

    // CHECK STATUS THAT YOU GET
    ResultMatcher matchStatus = MockMvcResultMatchers.status().isOk();

    // CHECK IF YOU GET ONE task
    ResultMatcher matchContent = MockMvcResultMatchers.content()
            .json(jsonifier.writeValueAsString(expectedResults));
    // PERFORM THE ABOVE
    this.mock.perform(mockRequest).andExpect(matchStatus).andExpect(matchContent);

}
```

Testing

- Automated Testing

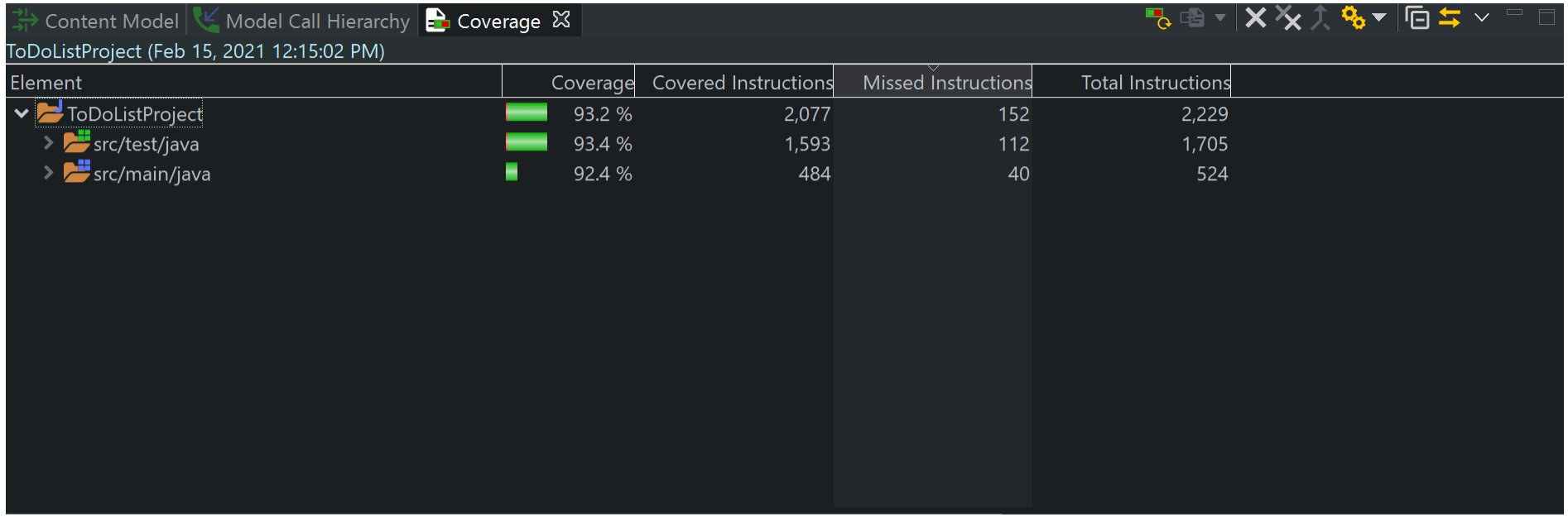
Was used to translate User Stories into automated tests. This would ensure that all user acceptance stories were working as per design.

```
//READ TASK
@Test
 void readAllTasks() throws InterruptedException {

    test = report.startTest("READ TASK TEST");

    //GIVEN THAT I CAN ACCES MY TO DO WEBSITE
    driver.get(URL);


    //WHEN THE WEBSITE LOADS UP

    Integer current = driver.findElements(By.xpath("//*[@id=\"taskID\"]/p")).size();


    //THEN I SHOULD BE ABLE TO SEE MY TASKS

    Integer expected = current;

    // ASSERTION


    if(expected.equals(current)) {
        test.log(LogStatus.PASS, "Tasks have loaded");
    } else {
        test.log(LogStatus.FAIL, "tasks have not loaded");
    }

    assertEquals(expected, current);
}
```


ExtentReports
Automation Report
2021-02-15 03:05:33
v2.41.1
TESTS
READ LIST TEST
Pass
CREATE LIST TEST
Pass
CREATE TASK TEST
Pass
DELETE LIST TEST
Pass
UPDATE LIST TEST
Pass
UPDATE TASK TEST
Pass
READ TASK TEST
Pass
DELETE TASK TEST
Pass
READ LIST TEST
2021-02-15 03:05:26
2021-02-15 03:05:28
0h 0m 1s+915ms
STATUS
TIMESTAMP
DETAILS
03:05:28
Lists have loaded

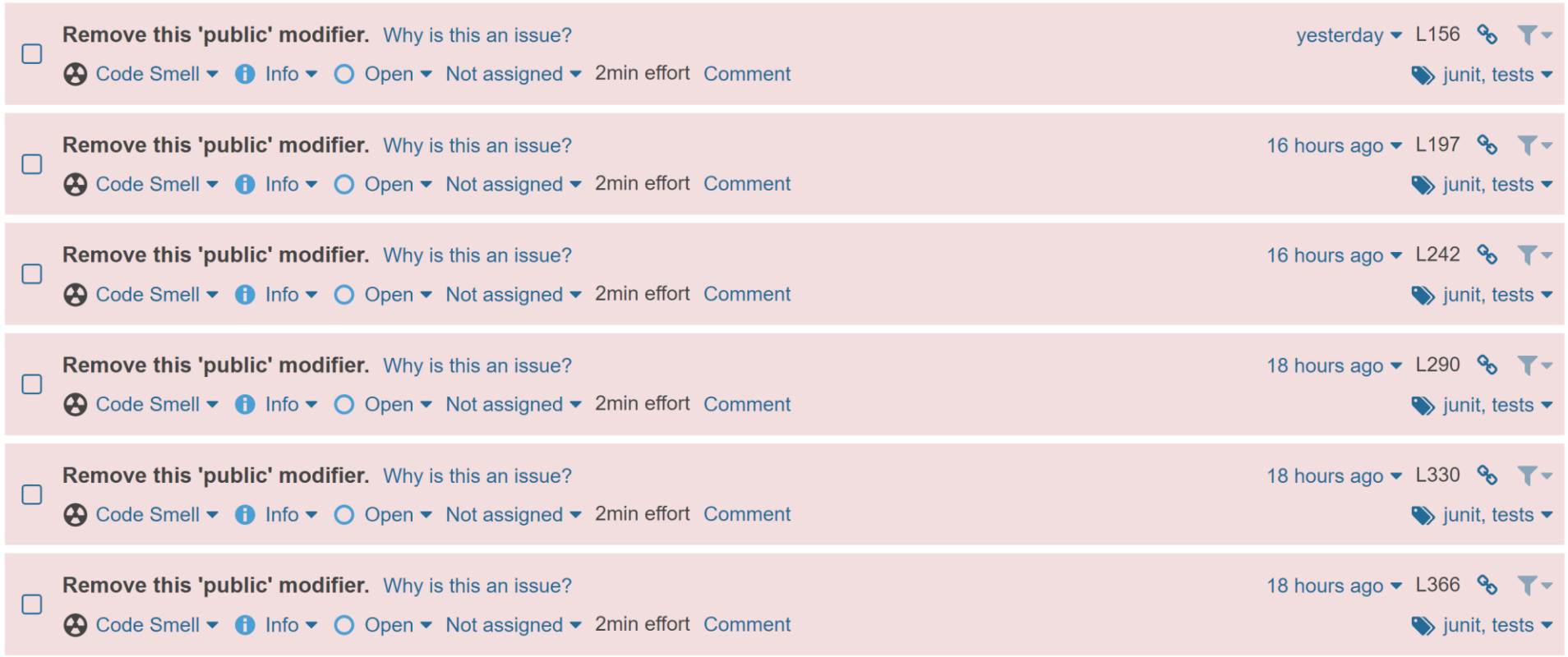
Testing

Project Coverage.

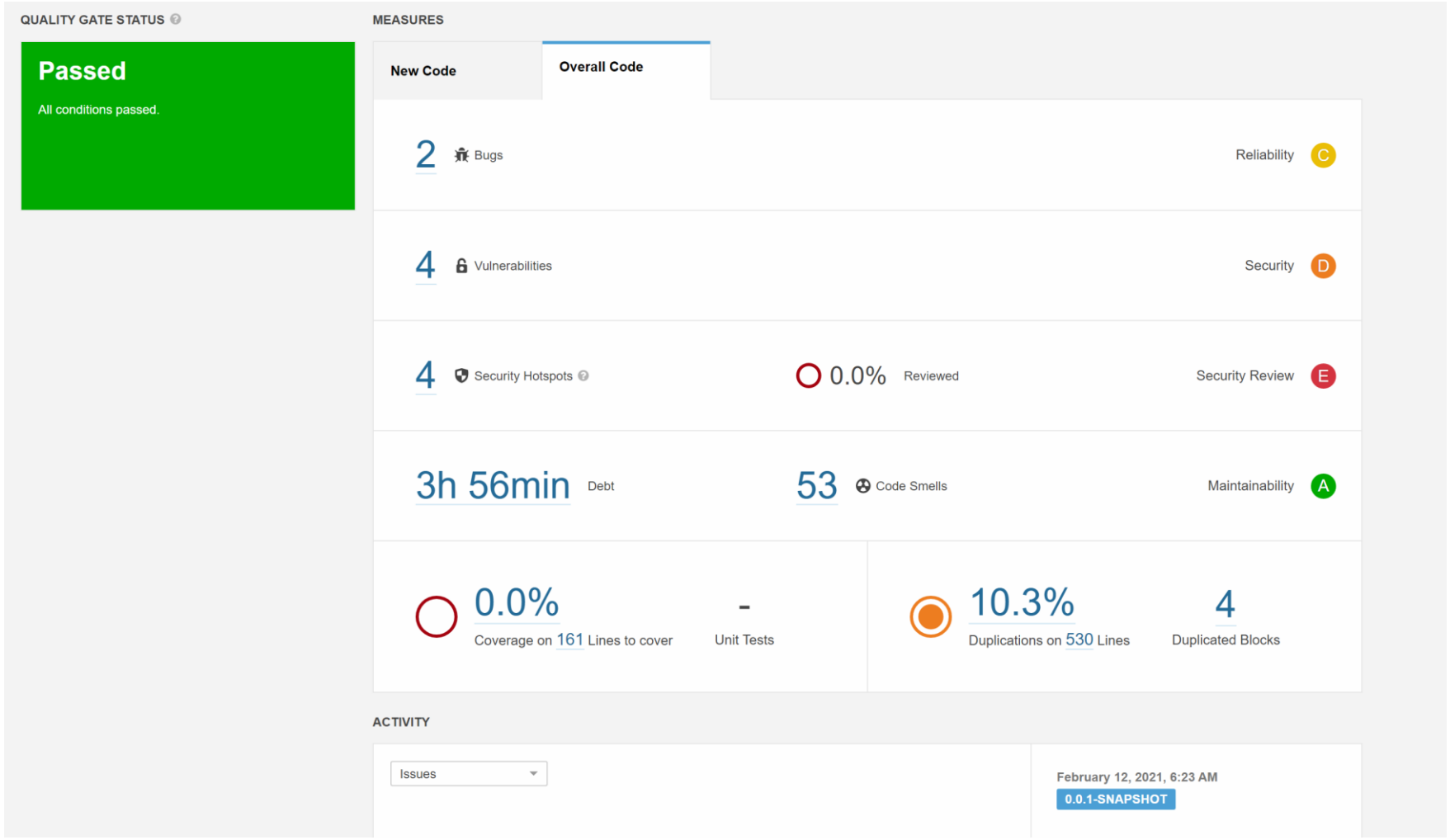


Improving the project

Sonar Cube was used to analyze the project code and test it against industry's standard. This was then used to improve the code



Improving the project

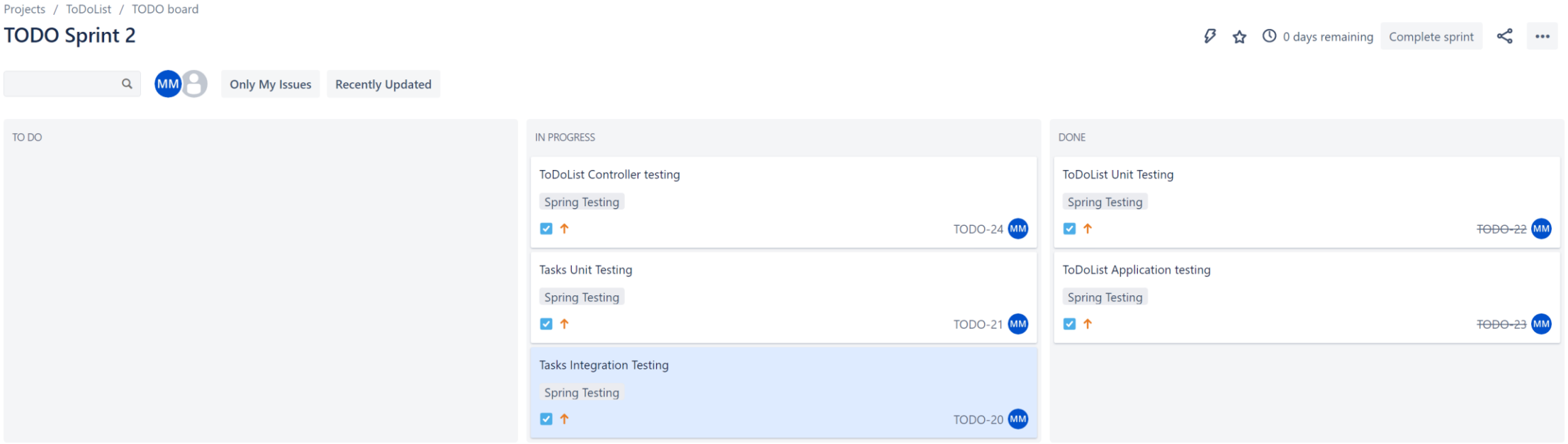


Test-driven development to further improve the application

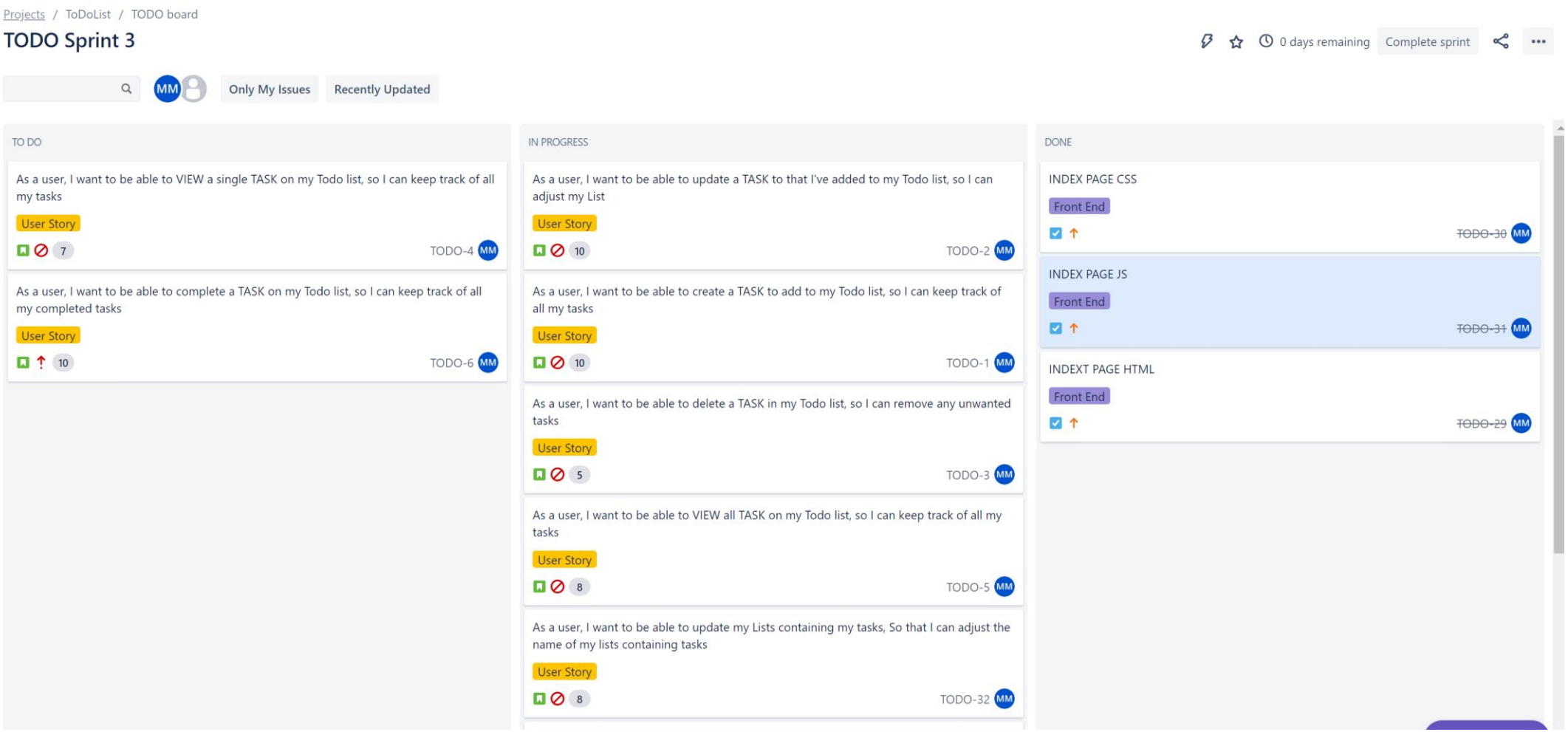
Sprint Review 1

TO DO	IN PROGRESS	DONE
Create ToDoList Service List Page TODO-18	Create ToDoListDTO List Page TODO-14	Create TasksDomain Tasks Page ~~TODO-11~~ MM
	Create ToDoListController List Page TODO-16	CreateTaskService Tasks Page ~~TODO-17~~ MM
	Create ToDoListDomain List Page TODO-12	Create ToDoList Application Application ~~TODO-19~~ MM
		Create TasksController Tasks Page ~~TODO-15~~ MM
		Create Application Config Application ~~TODO-10~~ MM
		Create TasksDTO Tasks Page ~~TODO-13~~ MM

Sprint Review 2



Sprint Review 3



Sprint retrospective

- What went right?
  - Testing and Time Management.
  - Completed all CRUD functionality for the Full Stack Application.
  - Built a fully functioning frontend with a focus on design.
  - Completed Testing for full application & Improved via SonarQube.
  - Met the MVP and then focused on improving.
- What went wrong?
  - Wanted to include more features on the website.
- What would we change?
  - Include a light mode on the website.
  - Improve print ability

Conclusion

- Project MVP was met successfully.
- Able to create a full stack application with functioning tests.
- Able to solve issues quickly and efficiently to ensure the deadline is met.
- Leant a lot and will continue to add features and testing in my spare time to improve my understanding of all the technologies I learnt.
- Built an understanding on how to take on a Project from ground up.
- Built an understanding of best practices throughout the project.

THANK YOU

Any Questions?