

To Do List Application.

Benjamin Simon

Introductions

Start of the IMS

Create ERD

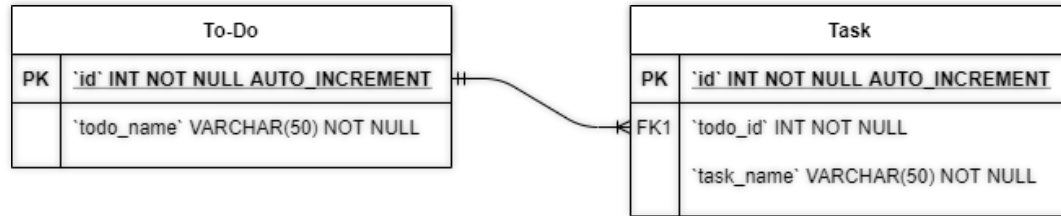
Create UML

Make an Initial Risk Assessment

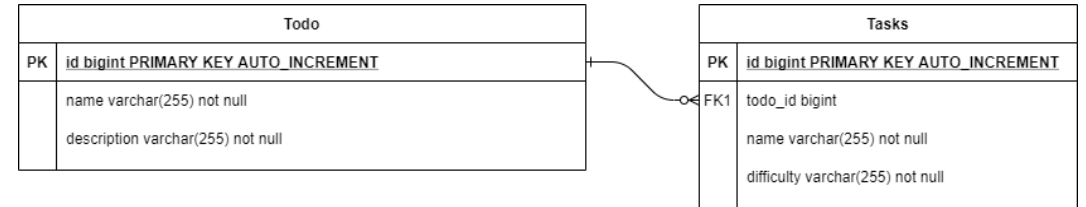
Make a MoSCoW

Set Up Jira and Github

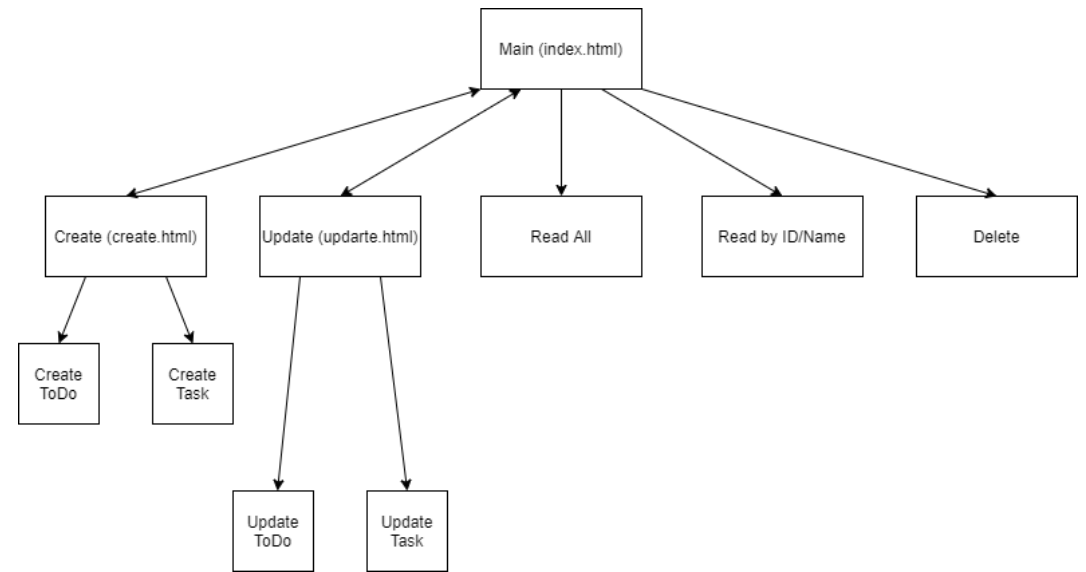
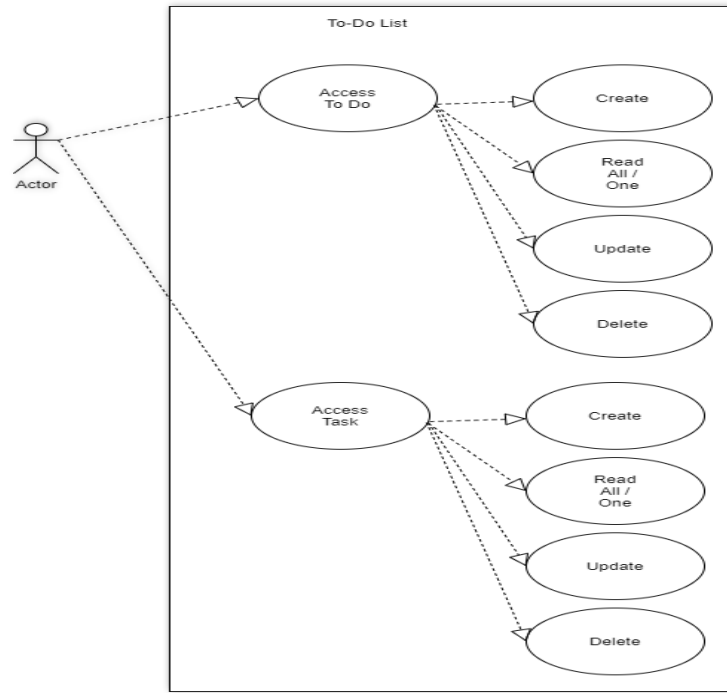
First ERD



Final ERD



ERD



UML

Risk Assessment

Risk register 1 – Initial scores and actions

Ref	Risk Description	Cause	Risk Event	Likelihood (1-5)	Impact (1-5)	Risk Rating	Action
1	Lack of time	Improper use of time and lack of time management	Unable to complete assigned project and meeting all criteria.	4	5	15	Plan daily and weekly tasks. Set sprint and time goals.
2	Lack of experience	Not having proper work experience.	Not completing project at industry standards.	3	5	10	Ask questions when needed, complete given task and do further research.
3	Self-Management	Overwhelming and burnout.	Unable to finish project, loss of motivation and affecting health.	4	4	12	Take breaks, do what you can, set realistic tasks.
4	Worldwide Disruption	Covid-19.	Illness, change of routine, remote work.	5	3	9	Social distancing, make sure have all equipment's, join online calls.
5	Unreliable application	Not having enough test results.	Project is not fully reliable.	3	4	12	Test all methods, ask for help.

MoSCoW

Must have:

- Code with a fully integrated version control System using GitHub and Git.
- A project management board that contains user stories, acceptance criteria and task that will need to complete the project.
- A risk assessment that outlines all potential risk as well and risk that are faced during the production.
- A relational database that contains a to-do and task tables.
- A functional “back-end” that follows the best industry practices and design, using all languages and materials covered during the course.
- A functional “front-end” that follows the best industry practices and design, using all languages and materials covered during the course.
- A build of the application that fully functions.
- An industry standard of 80% test coverage, using unit, integration, and user-acceptance testing.
- A well thought out presentation that covers everything about the project as well as materials covered during the training.

Should have:

- Good use of comments throughout the code to get an understanding of what is happening.
- Constantly commit project to GitHub to have backups, that contains clear messages to know what has been committed.
- A clean and structured folder.
- A user-friendly UI.

Could have:

- A test coverage that goes above 95%.
- A daily diary that covers everything that has been applied and created for the project.

Wont have:

- User log-in credentials.
- Using an actual domain so people can access from anywhere.

Consultant journey

Technologies

Spring Tool Suite(JAVA)

Visual Studio Code (HTML, CSS, JavaScript)

SQL

JUnit

Mockito

Selenium

SonarQube

Postman

H2-Console

Web Console

Maven

Management

Jira

MoSCoW

Risk Assessment

Integration Testing

```
// Create test
@Test
void createTest() throws Exception {
    TaskDTO testDTO = mapToDTO(new Task("Water", "Easy"));
    String testDTOAsJSON = this.jsonifier.writeValueAsString(testDTO);

    RequestBuilder request = post(URI + "/create").contentType(MediaType.APPLICATION_JSON).content(testDTOAsJSON);

    ResultMatcher checkStatus = status().isCreated();

    TaskDTO testSavedDTO = mapToDTO(new Task("Water", "Easy"));
    testSavedDTO.setId(4L);
    String testSavedDTOAsJSON = this.jsonifier.writeValueAsString(testSavedDTO);

    ResultMatcher checkBody = content().json(testSavedDTOAsJSON);

    this.mvc.perform(request).andExpect(checkStatus).andExpect(checkBody);

    this.mvc.perform(post(URI + "/create").contentType(MediaType.APPLICATION_JSON).content(testDTOAsJSON))
        .andExpect(status().isCreated()).andExpect(content().json(testSavedDTOAsJSON));
}

// Update test
@Test
void updateTest() throws Exception {
    TaskDTO testDTO = mapToDTO(new Task("Gas", "Hard"));
    String testDTOAsJSON = this.jsonifier.writeValueAsString(testDTO);

    RequestBuilder request = put(URI + "/update/1").contentType(MediaType.APPLICATION_JSON).content(testDTOAsJSON);

    ResultMatcher checkStatus = status().isAccepted();

    TaskDTO testSavedDTO = mapToDTO(new Task("Gas", "Hard"));
    testSavedDTO.setId(1L);
    String testSavedDTOAsJSON = this.jsonifier.writeValueAsString(testSavedDTO);
}
```


Unit Testing

```
// Create
@Test
void createTest() throws Exception {
    when(this.service.create(TEST_TASK_1)).thenReturn(this.mapToDto(TEST_TASK_1));
    assertThat(new ResponseEntity<TaskDTO>(this.mapToDto(TEST_TASK_1), HttpStatus.CREATED))
        .isEqualTo(this.controller.create(TEST_TASK_1));
    verify(this.service, atLeastOnce()).create(TEST_TASK_1);
}

// Read one
@Test
void readOneTest() throws Exception {
    when(this.service.readOne(TEST_TASK_1.getId())).thenReturn(this.mapToDto(TEST_TASK_1));
    assertThat(new ResponseEntity<TaskDTO>(this.mapToDto(TEST_TASK_1), HttpStatus.OK))
        .isEqualTo(this.controller.readOne(TEST_TASK_1.getId()));
    verify(this.service, atLeastOnce()).readOne(TEST_TASK_1.getId());
}

// Read all
@Test
void readAllTest() throws Exception {
    List<TaskDTO> dtos = LISTOFCARS.stream().map(this::mapToDto).collect(Collectors.toList());
    when(this.service.readAll()).thenReturn(dtos);
    assertThat(this.controller.read()).isEqualTo(new ResponseEntity<>(dtos, HttpStatus.OK));
    verify(this.service, atLeastOnce()).readAll();
}
```

Selenium

```
@Test
public void testHome() throws InterruptedException {

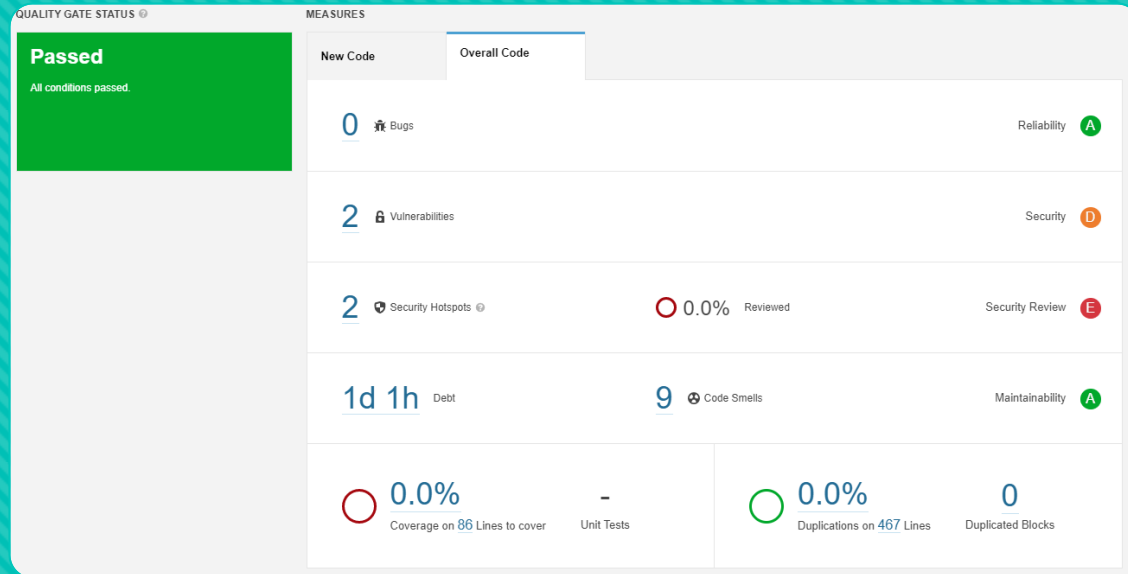
    assertEquals("Home", driver.getTitle());
}

@Test
public void testCreatePage() throws InterruptedException {
    WebElement createBut = driver.findElement(By.id("createButton"));
    createBut.click();
    assertEquals("Create", driver.getTitle());
}

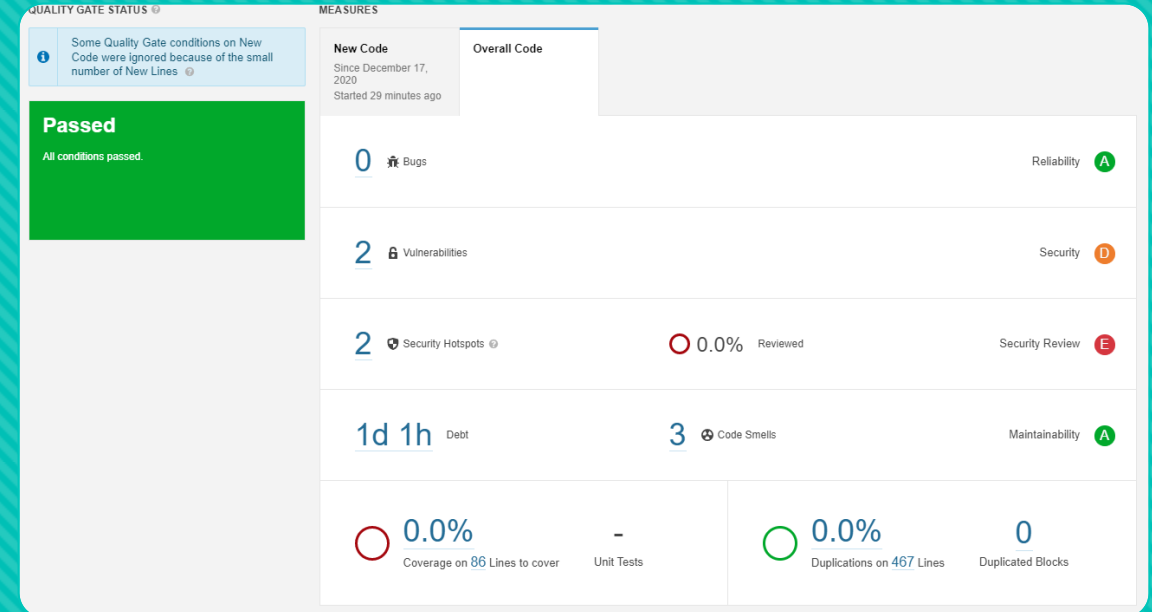
@Test
public void testUpdatePage() throws InterruptedException {
    WebElement updateBut = driver.findElement(By.id("updateButton"));
    updateBut.click();
    assertEquals("Update", driver.getTitle());
}

@Test
public void testCreateTodo() throws InterruptedException {
    WebElement createBut = driver.findElement(By.id("createButton"));
    createBut.click();
    WebElement todoName = driver.findElement(By.id("floatingName"));
    String tName = "Bake cake";
    todoName.sendKeys(tName);
    WebElement todoDes = driver.findElement(By.id("floatingDescription"));
    String tDes = "Make by tomorrow";
    todoDes.sendKeys(tDes);
    WebElement subBut = driver.findElement(By.id("submitButton"));
    subBut.click();
    driver.switchTo().alert().dismiss();
}
```



















Before



After




SonarQube

▼  ToDoTaskQA		99.5 %	2,354	13	2,367
▼  src/main/java		96.9 %	412	13	425
>  com.example.demo.exception		0.0 %	0	6	6
>  com.example.demo		37.5 %	3	5	8
>  com.example.demo.service		98.9 %	184	2	186
>  com.example.demo.config		100.0 %	10	0	10
>  com.example.demo.persistence.domain		100.0 %	48	0	48
>  com.example.demo.rest		100.0 %	118	0	118
>  com.example.demo.util		100.0 %	49	0	49
>  src/test/java		100.0 %	1,942	0	1,942

Overall Testing

User Stories

 ~~708-38~~ As a user, I would like to create tasks, so that I know how to complete my todo.


Create Task

Task Name

Task Difficulty

Add to Todo

User Stories

 ~~TQB-29~~ As a user, I will like to read all my to-do list, so that I know what i need to do.

ID	Name	Description	Task	Difficulty
1	Bake a cake	best	Flour	Hard
1	Bake a cake	best	egg	easy
2	Build a house	best	Bricks	Hard

User Stories

~~708-33~~ As as user, I will like to update my to-do list, so that i can make changes.

Update To Do

To Do ID

To Do Name

To Do Description

Sprint Review

Completed

Created the UML, ERD and Risk Assessment.

Set Up a Jira and GitHub.

Have a well designed and functioning front end.

Have fully functioning back end and CRUD functionalities.

Reach above 80% testing, this includes unit, integration, selenium and SonarQube.

Struggle

Front end Development.

JavaScript.

Sprint Retrospective

What Went Well

Time management

Testing

GCP

Eclipse Java Programming

What could be improved

Jira

GitHub

Conclusion

- Expanded knowledge on Java and JavaScript.
- Learned how to work on Spring Tool, Postman and Visual Studio Code.
- Learned how to test using Junit, Mockito, Selenium and SonarQube.
- Best practice of Time Management.



**ANY
QUESTIONS?**