Inventory Management System

20DecSDET2 Cameron Guthrie

Introduction

➔ **How did I approach the specification?**

Fundamentals

Minimum Viable Product

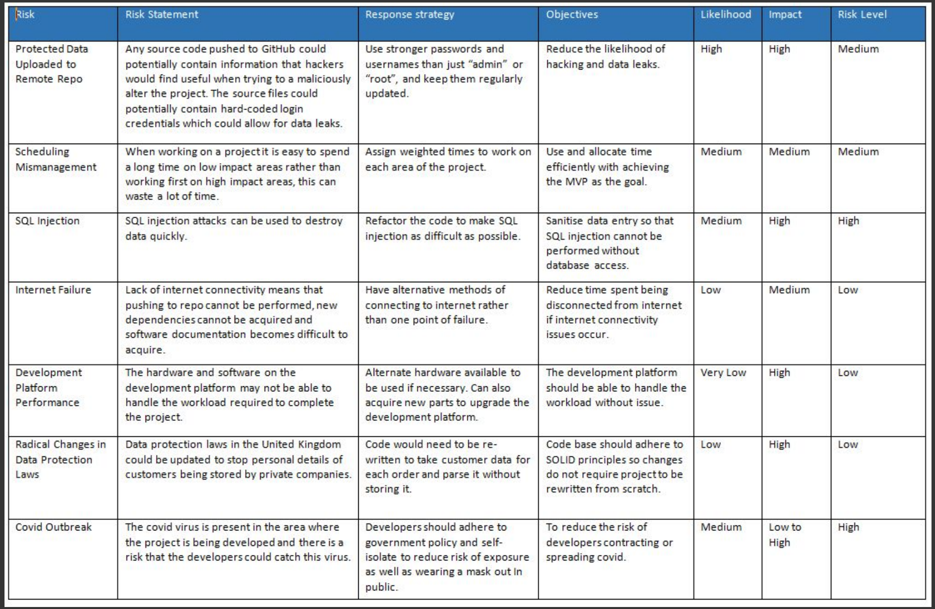
The Scope

Some thoughts

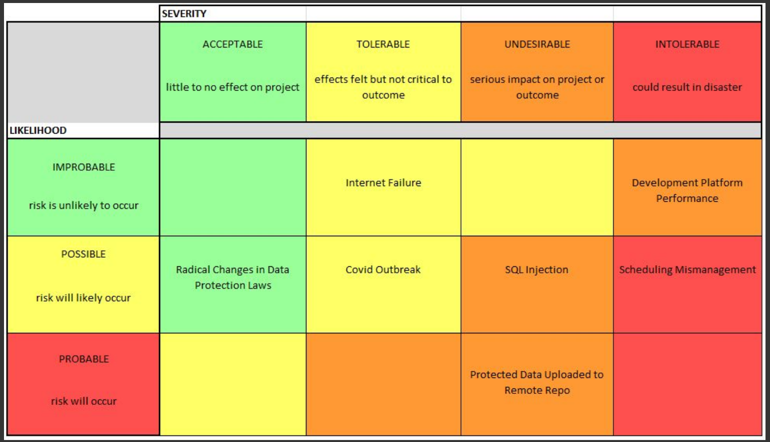
I was incredibly **excited** to work on this project**.**

But also **worried** about the documentation!

Risk



Risk	Risk Statement	Response strategy	Objectives	Likelihood	Impact	Risk Level
Protected Data Uploaded to Remote Repo	Any source code pushed to GitHub could potentially contain information that hackers would find useful when trying to a maliciously alter the project. The source files could potentially contain hard-coded login credentials which could allow for data leaks.	Use stronger passwords and usernames than just "admin" or "root", and keep them regularly updated.	Reduce the likelihood of hacking and data leaks.	High	High	Medium
Scheduling Mismanagement	When working on a project it is easy to spend a long time on low impact areas rather than working first on high impact areas, this can waste a lot of time.	Assign weighted times to work on each area of the project.	Use and allocate time efficiently with achieving the MVP as the goal.	Medium	Medium	Medium
SQL Injection	SQL injection attacks can be used to destroy data quickly.	Refactor the code to make SQL injection as difficult as possible.	Sanitise data entry so that SQL injection cannot be performed without database access.	Medium	High	High
Internet Failure	Lack of internet connectivity means that pushing to repo cannot be performed, new dependencies cannot be acquired and software documentation becomes difficult to acquire.	Have alternative methods of connecting to internet rather than one point of failure.	Reduce time spent being disconnected from internet if internet connectivity issues occur.	Low	Medium	Low
Development Platform Performance	The hardware and software on the development platform may not be able to handle the workload required to complete the project.	Alternate hardware available to be used if necessary. Can also acquire new parts to upgrade the development platform.	The development platform should be able to handle the workload without issue.	Very Low	High	Low
Radical Changes in Data Protection Laws	Data protection laws in the United Kingdom could be updated to stop personal details of customers being stored by private companies.	Code would need to be re-written to take customer data for each order and parse it without storing it.	Code base should adhere to SOLID principles so changes do not require project to be rewritten from scratch.	Low	High	Low
Covid Outbreak	The covid virus is present in the area where the project is being developed and there is a risk that the developers could catch this virus.	Developers should adhere to government policy and self-isolate to reduce risk of exposure as well as wearing a mask out In public.	To reduce the risk of developers contracting or spreading covid.	Medium	Low to High	High

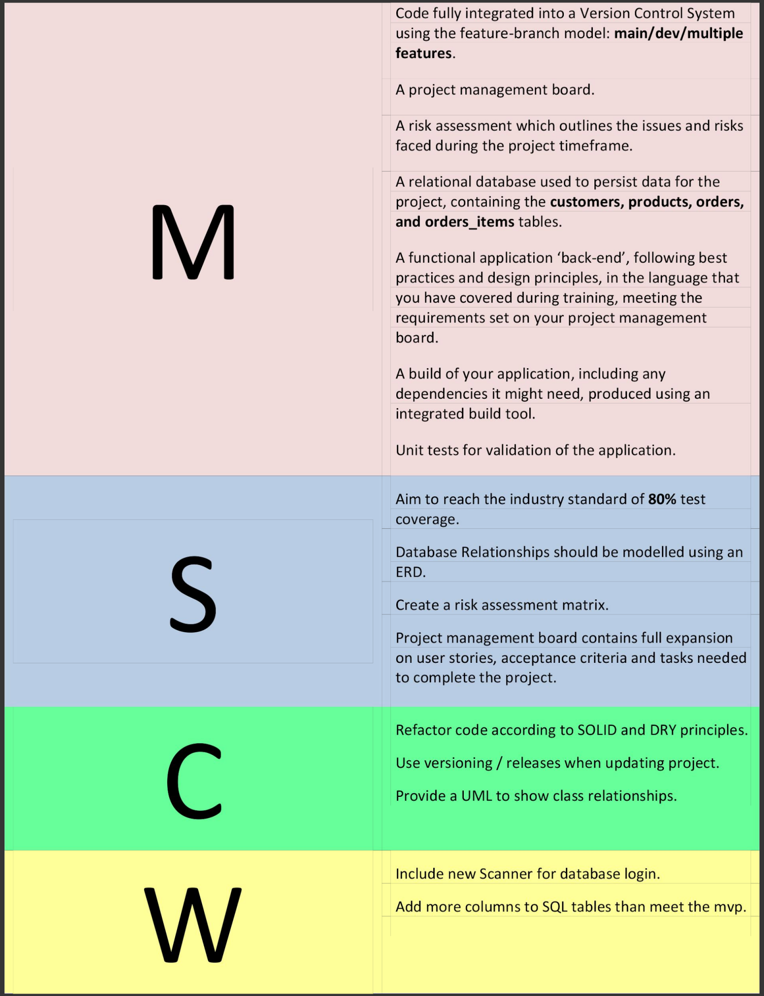


	SEVERITY			
	ACCEPTABLE little to no effect on project	TOLERABLE effects felt but not critical to outcome	UNDESIRABLE serious impact on project or outcome	INTOLERABLE could result in disaster
LIKELIHOOD				
IMPROBABLE risk is unlikely to occur		Internet Failure		Development Platform Performance
POSSIBLE risk will likely occur	Radical Changes in Data Protection Laws	Covid Outbreak	SQL Injection	Scheduling Mismanagement
PROBABLE risk will occur			Protected Data Uploaded to Remote Repo	

MoSCoW

An interesting method which can help visualise project goals.

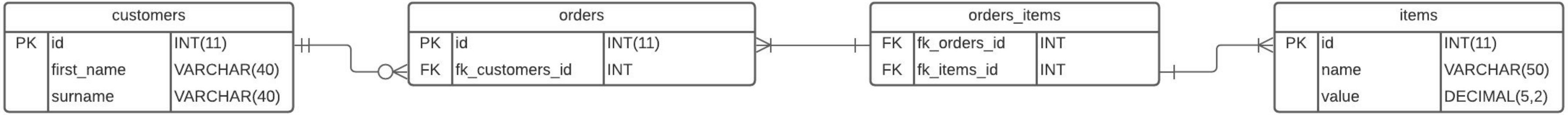
- ➔ **Must and Should**
  Cover the Domain and Scope
- ➔ **Could and Wont**
  Are for stretch goals and extras.



M	Code fully integrated into a Version Control System using the feature-branch model: **main/dev/multiple features**. A project management board. A risk assessment which outlines the issues and risks faced during the project timeframe. A relational database used to persist data for the project, containing the **customers, products, orders, and orders_items** tables. A functional application ‘back-end’, following best practices and design principles, in the language that you have covered during training, meeting the requirements set on your project management board. A build of your application, including any dependencies it might need, produced using an integrated build tool. Unit tests for validation of the application.
S	Aim to reach the industry standard of **80%** test coverage. Database Relationships should be modelled using an ERD. Create a risk assessment matrix. Project management board contains full expansion on user stories, acceptance criteria and tasks needed to complete the project.
C	Refactor code according to SOLID and DRY principles. Use versioning / releases when updating project. Provide a UML to show class relationships.
W	Include new Scanner for database login. Add more columns to SQL tables than meet the mvp.

Entity Relationship Diagram

Predicted



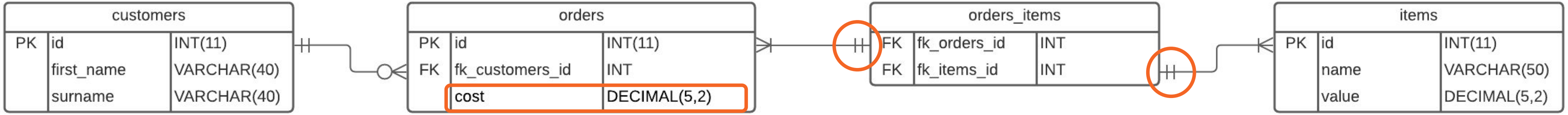
customers		
PK	id	INT(11)
	first_name	VARCHAR(40)
	surname	VARCHAR(40)

orders		
PK	id	INT(11)
FK	fk_customers_id	INT

orders_items		
FK	fk_orders_id	INT
FK	fk_items_id	INT

items		
PK	id	INT(11)
	name	VARCHAR(50)
	value	DECIMAL(5,2)

v0.3 / Final



customers		
PK	id	INT(11)
	first_name	VARCHAR(40)
	surname	VARCHAR(40)

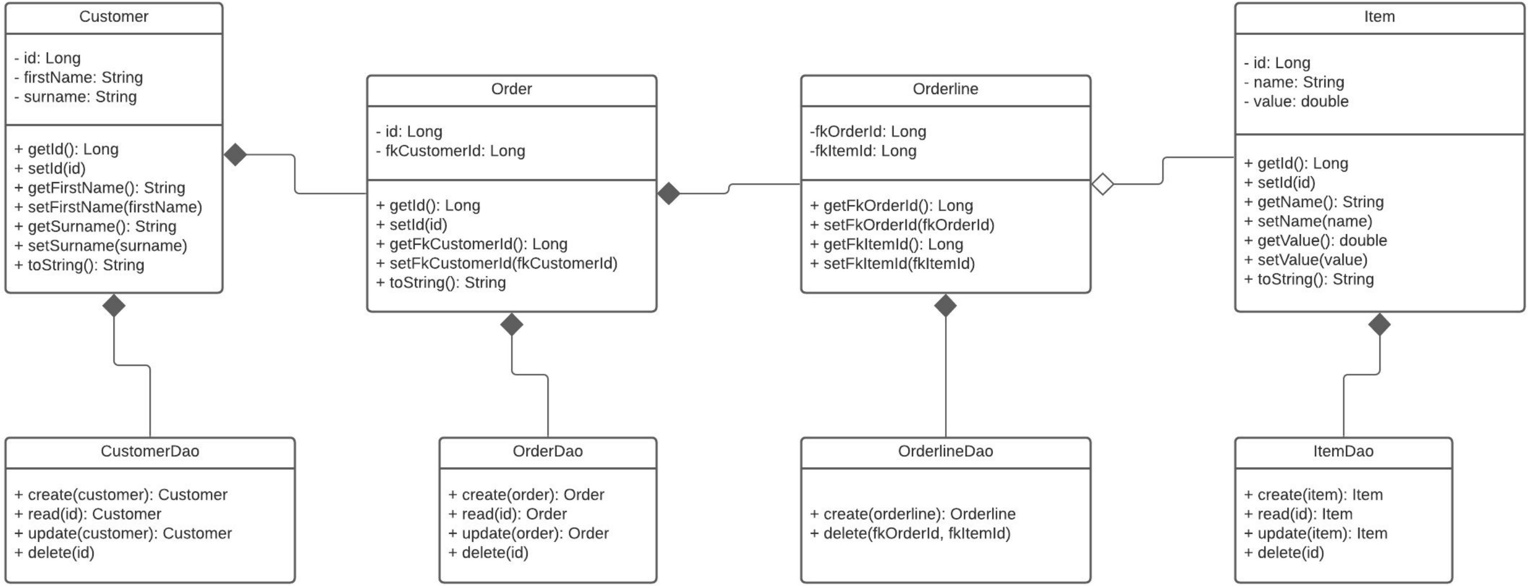
orders		
PK	id	INT(11)
FK	fk_customers_id	INT
	cost	DECIMAL(5,2)

orders_items		
FK	fk_orders_id	INT
FK	fk_items_id	INT

items		
PK	id	INT(11)
	name	VARCHAR(50)
	value	DECIMAL(5,2)

Unified Modeling Language Class Diagram

Predicted



Customer

- id: Long
- firstName: String
- surname: String

+ getId(): Long
+ setId(id)
+ getFirstName(): String
+ setFirstName(firstName)
+ getSurname(): String
+ setSurname(surname)
+ toString(): String

Order

- id: Long
- fkCustomerId: Long

+ getId(): Long
+ setId(id)
+ getFkCustomerId(): Long
+ setFkCustomerId(fkCustomerId)
+ toString(): String

Orderline

-fkOrderId: Long
-fkItemId: Long

+ getFkOrderId(): Long
+ setFkOrderId(fkOrderId)
+ getFkItemId(): Long
+ setFkItemId(fkItemId)

Item

- id: Long
- name: String
- value: double

+ getId(): Long
+ setId(id)
+ getName(): String
+ setName(name)
+ getValue(): double
+ setValue(value)
+ toString(): String

CustomerDao

+ create(customer): Customer
+ read(id): Customer
+ update(customer): Customer
+ delete(id)

OrderDao

+ create(order): Order
+ read(id): Order
+ update(order): Order
+ delete(id)

OrderlineDao

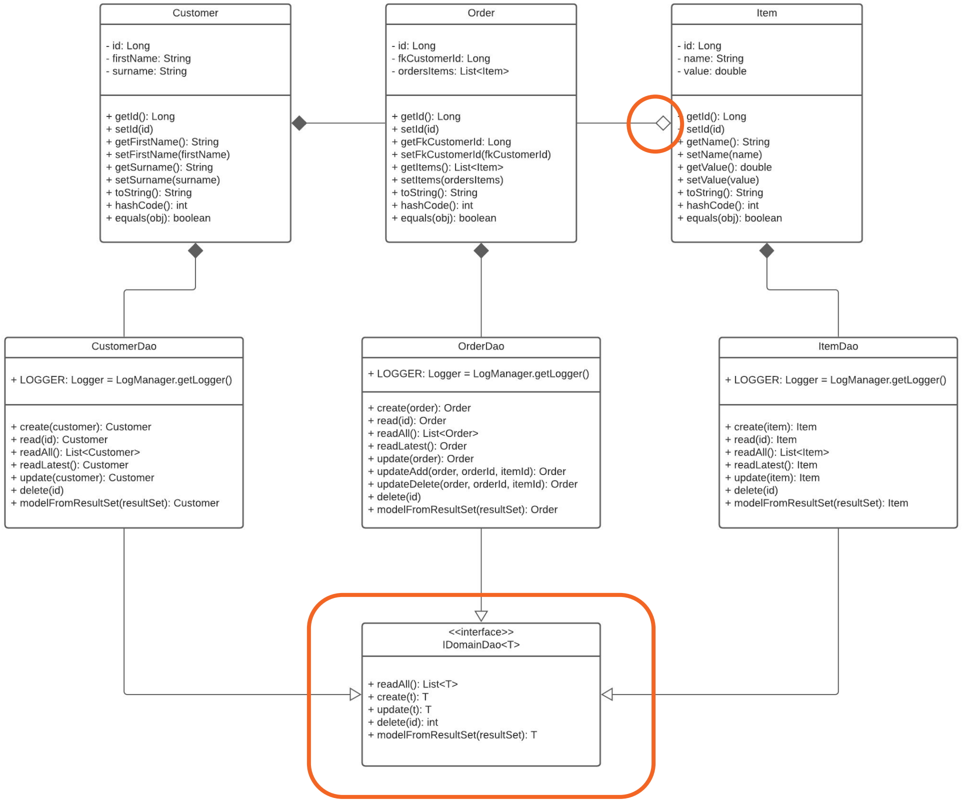
+ create(orderline): Orderline
+ delete(fkOrderId, fkItemId)

ItemDao

+ create(item): Item
+ read(id): Item
+ update(item): Item
+ delete(id)

Unified Modeling Language Class Diagram

v0.1



Customer

- id: Long
- firstName: String
- surname: String

+ getId(): Long
+ setId(id)
+ getFirstName(): String
+ setFirstName(firstName)
+ getSurname(): String
+ setSurname(surname)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean

Order

- id: Long
- fkCustomerId: Long
- ordersItems: List<Item>

+ getId(): Long
+ setId(id)
+ getFkCustomerId: Long
+ setFkCustomerId(fkCustomerId)
+ getItems(): List<Item>
+ setItems(ordersItems)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean

Item

- id: Long
- name: String
- value: double

+ getId(): Long
+ setId(id)
+ getName(): String
+ setName(name)
+ getValue(): double
+ setValue(value)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean

CustomerDao

+ LOGGER: Logger = LogManager.getLogger()

+ create(customer): Customer
+ read(id): Customer
+ readAll(): List<Customer>
+ readLatest(): Customer
+ update(customer): Customer
+ delete(id)
+ modelFromResultSet(resultSet): Customer

OrderDao

+ LOGGER: Logger = LogManager.getLogger()

+ create(order): Order
+ read(id): Order
+ readAll(): List<Order>
+ readLatest(): Order
+ update(order): Order
+ updateAdd(order, orderId, itemId): Order
+ updateDelete(order, orderId, itemId): Order
+ delete(id)
+ modelFromResultSet(resultSet): Order

ItemDao

+ LOGGER: Logger = LogManager.getLogger()

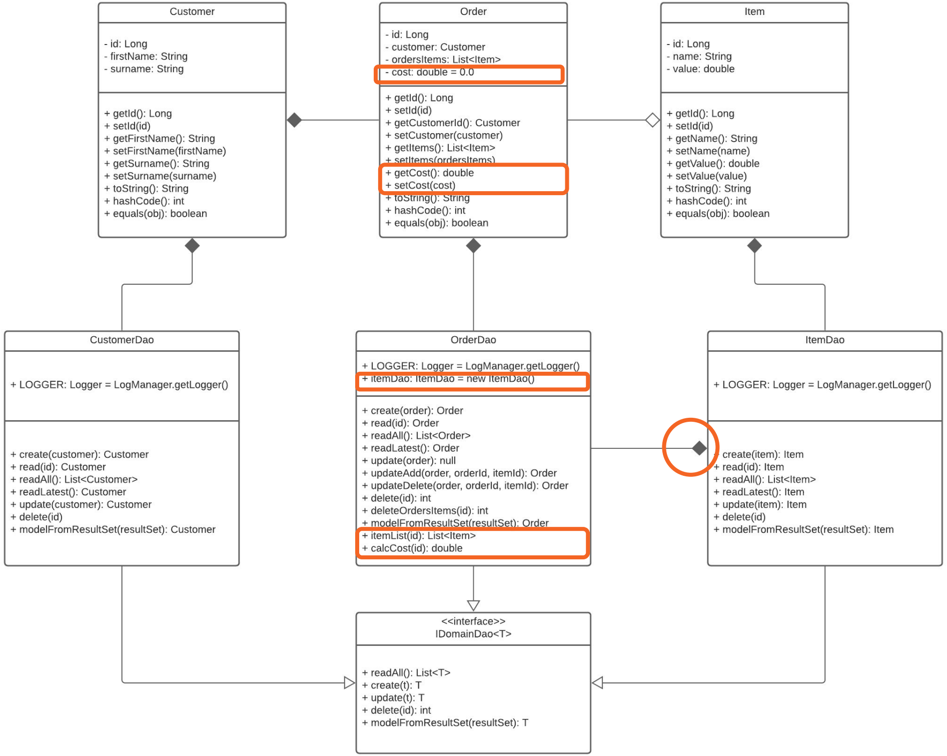
+ create(item): Item
+ read(id): Item
+ readAll(): List<Item>
+ readLatest(): Item
+ update(item): Item
+ delete(id)
+ modelFromResultSet(resultSet): Item

<<interface>>
IDomainDao<T>

+ readAll(): List<T>
+ create(t): T
+ update(t): T
+ delete(id): int
+ modelFromResultSet(resultSet): T

Unified Modeling Language Class Diagram

v0.3 / Final



```
Customer

- id: Long
- firstName: String
- surname: String

+ getId(): Long
+ setId(id)
+ getFirstName(): String
+ setFirstName(firstName)
+ getSurname(): String
+ setSurname(surname)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean
```

```
Order

- id: Long
- customer: Customer
- ordersItems: List<Item>
- cost: double = 0.0

+ getId(): Long
+ setId(id)
+ getCustomerId(): Customer
+ setCustomer(customer)
+ getItems(): List<Item>
+ setItems(ordersItems)
+ getCost(): double
+ setCost(cost)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean
```

```
Item

- id: Long
- name: String
- value: double

+ getId(): Long
+ setId(id)
+ getName(): String
+ setName(name)
+ getValue(): double
+ setValue(value)
+ toString(): String
+ hashCode(): int
+ equals(obj): boolean
```

```
CustomerDao

+ LOGGER: Logger = LogManager.getLogger()

+ create(customer): Customer
+ read(id): Customer
+ readAll(): List<Customer>
+ readLatest(): Customer
+ update(customer): Customer
+ delete(id)
+ modelFromResultSet(resultSet): Customer
```

```
OrderDao

+ LOGGER: Logger = LogManager.getLogger()
+ itemDao: ItemDao = new ItemDao()

+ create(order): Order
+ read(id): Order
+ readAll(): List<Order>
+ readLatest(): Order
+ update(order): null
+ updateAdd(order, orderId, itemId): Order
+ updateDelete(order, orderId, itemId): Order
+ delete(id): int
+ deleteOrdersItems(id): int
+ modelFromResultSet(resultSet): Order
+ itemList(id): List<Item>
+ calcCost(id): double
```

```
ItemDao

+ LOGGER: Logger = LogManager.getLogger()

+ create(item): Item
+ read(id): Item
+ readAll(): List<Item>
+ readLatest(): Item
+ update(item): Item
+ delete(id)
+ modelFromResultSet(resultSet): Item
```

```
<<interface>>
IDomainDao<T>

+ readAll(): List<T>
+ create(t): T
+ update(t): T
+ delete(id): int
+ modelFromResultSet(resultSet): T
```

Consultant Journey

QA Academy has been a joy to learn at and I feel it imparted, to me, all of the skills that I needed to complete this project.

➔ **Agile**

A new way to work that's both fast and fun!

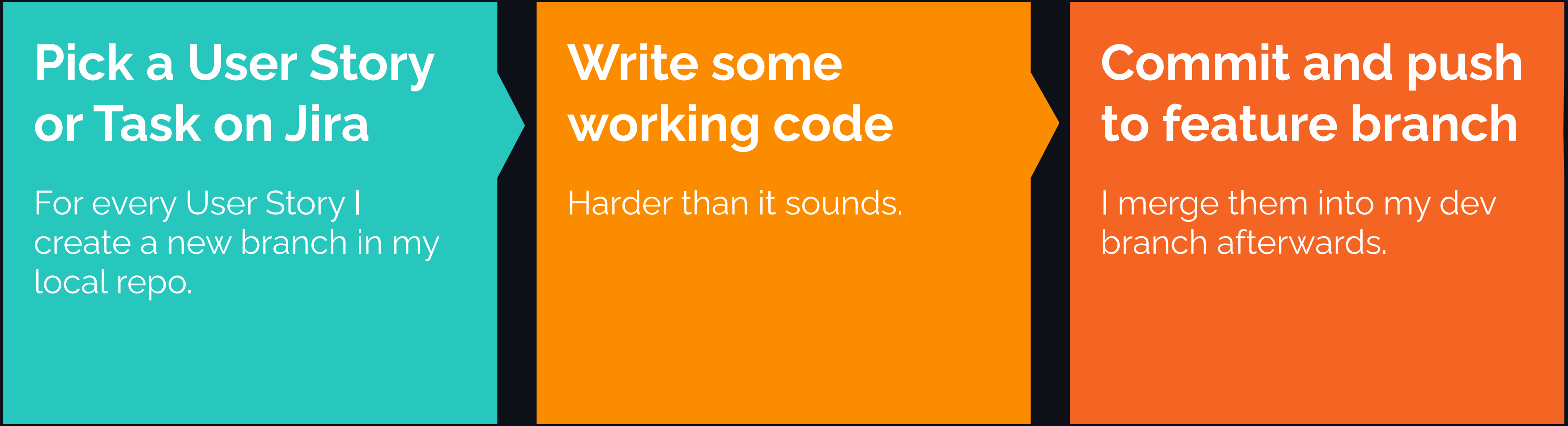
➔ **Java and OOP**

A new language and a new way to program!

➔ **Databases and SQL**

Data has to be stored somewhere!

Version Control System



Pick a User Story or Task on Jira

For every User Story I create a new branch in my local repo.

Write some working code

Harder than it sounds.

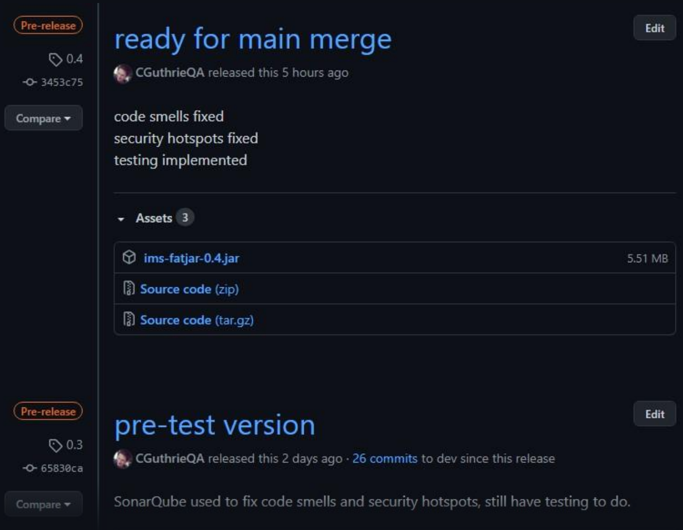
Commit and push to feature branch

I merge them into my dev branch afterwards.

More Version Control!

Releases

On GitHub



ready for main merge

CGuthrieQA released this 5 hours ago

code smells fixed
security hotspots fixed
testing implemented

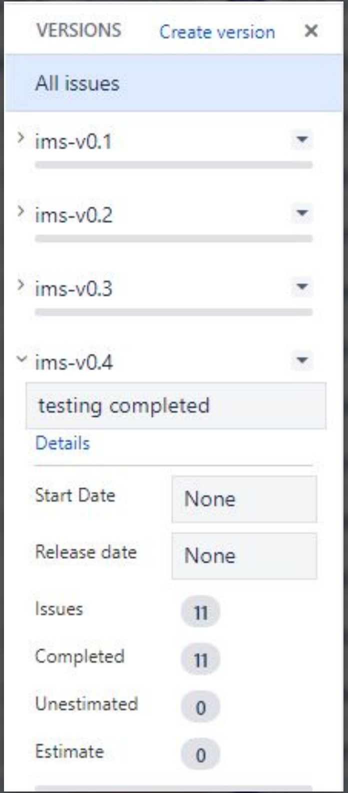
ims-fatjar-0.4.jar	5.51 MB
Source code (zip)	
Source code (tar.gz)	

pre-test version

CGuthrieQA released this 2 days ago · 26 commits to dev since this release

SonarQube used to fix code smells and security hotspots, still have testing to do.

Jira was also used to track versions



VERSIONS Create version ×

All issues

ims-v0.1

ims-v0.2

ims-v0.3

ims-v0.4

testing completed

Details

Start Date	None
Release date	None
Issues	11
Completed	11
Unestimated	0
Estimate	0

How I handled tests using JUnit to reach over 80% coverage for domain and DAO objects.

➔ **Write**

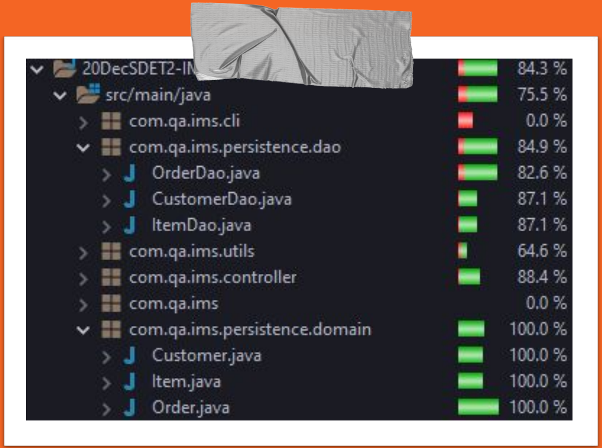
Create code fo a test.

➔ **Run**

To check if it passes and works as intended.

➔ **Refactor**

Remove duplicate code, simplify it.

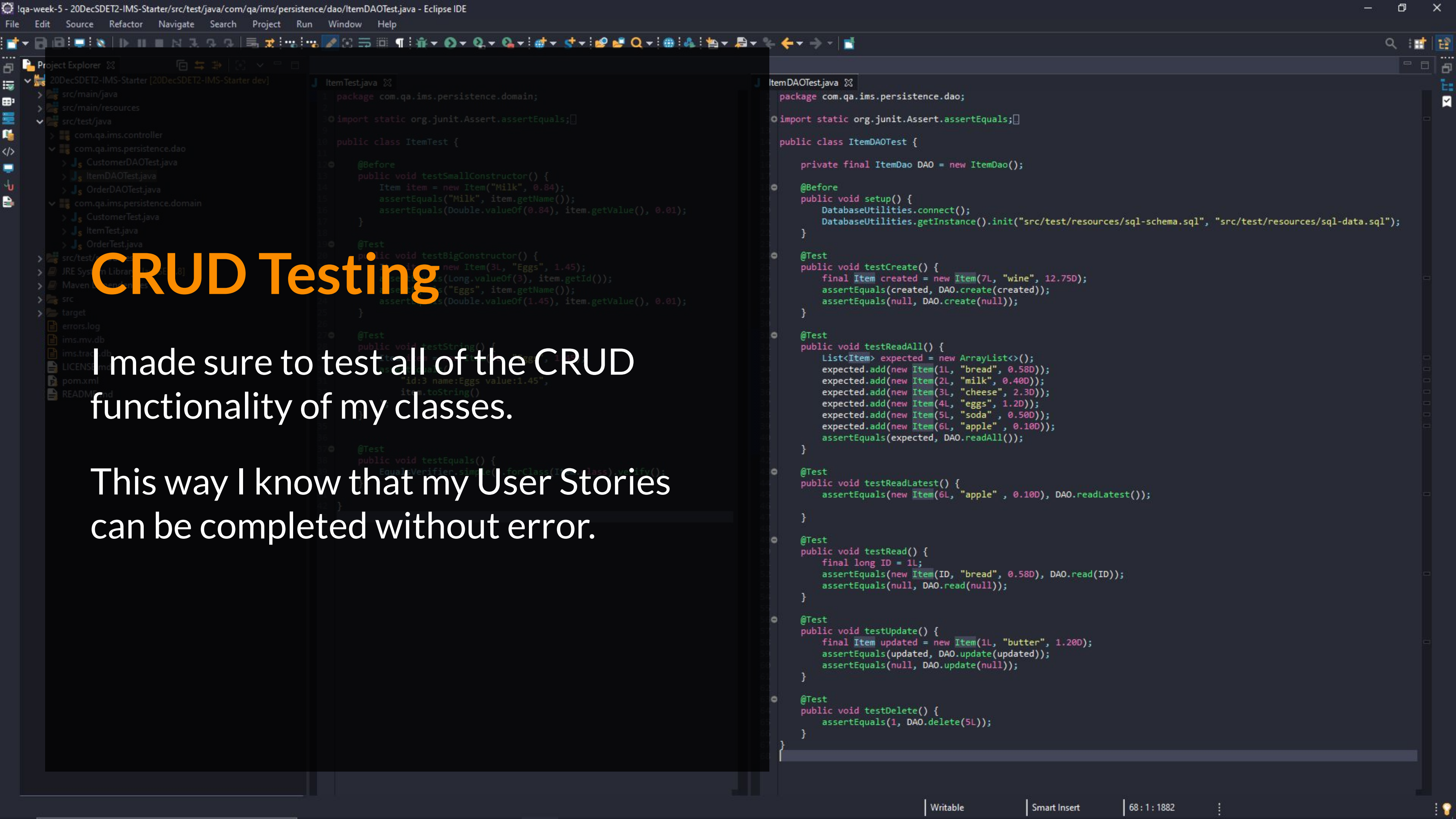


```
v 20DecSDET2-IM                                  84.3 %
  v src/main/java                                75.5 %
    > com.qa.ims.cli                              0.0 %
    v com.qa.ims.persistence.dao                 84.9 %
      > OrderDao.java                            82.6 %
      > CustomerDao.java                         87.1 %
      > ItemDao.java                             87.1 %
    > com.qa.ims.utils                           64.6 %
    > com.qa.ims.controller                      88.4 %
    > com.qa.ims                                  0.0 %
    v com.qa.ims.persistence.domain             100.0 %
      > Customer.java                           100.0 %
      > Item.java                               100.0 %
      > Order.java                              100.0 %
```

CRUD Testing

I made sure to test all of the CRUD functionality of my classes.

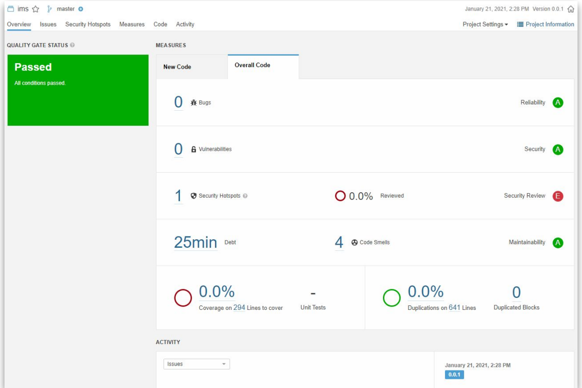
This way I know that my User Stories can be completed without error.


!qa-week-5 - 20DecSDET2-IMS-Starter/src/test/java/com/qa/ims/persistence/dao/ItemDAOTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ItemTest.java
ItemDAOTest.java
Writable
Smart Insert
68 : 1 : 1882

```java
package com.qa.ims.persistence.dao;

import static org.junit.Assert.assertEquals;

public class ItemDAOTest {

    private final ItemDao DAO = new ItemDao();

    @Before
    public void setup() {
        DatabaseUtilities.connect();
        DatabaseUtilities.getInstance().init("src/test/resources/sql-schema.sql", "src/test/resources/sql-data.sql");
    }

    @Test
    public void testCreate() {
        final Item created = new Item(7L, "wine", 12.75D);
        assertEquals(created, DAO.create(created));
        assertEquals(null, DAO.create(null));
    }

    @Test
    public void testReadAll() {
        List<Item> expected = new ArrayList<>();
        expected.add(new Item(1L, "bread", 0.58D));
        expected.add(new Item(2L, "milk", 0.40D));
        expected.add(new Item(3L, "cheese", 2.3D));
        expected.add(new Item(4L, "eggs", 1.2D));
        expected.add(new Item(5L, "soda" , 0.50D));
        expected.add(new Item(6L, "apple" , 0.10D));
        assertEquals(expected, DAO.readAll());
    }

    @Test
    public void testReadLatest() {
        assertEquals(new Item(6L, "apple" , 0.10D), DAO.readLatest());

    }

    @Test
    public void testRead() {
        final long ID = 1L;
        assertEquals(new Item(ID, "bread", 0.58D), DAO.read(ID));
        assertEquals(null, DAO.read(null));
    }

    @Test
    public void testUpdate() {
        final Item updated = new Item(1L, "butter", 1.20D);
        assertEquals(updated, DAO.update(updated));
        assertEquals(null, DAO.update(null));
    }

    @Test
    public void testDelete() {
        assertEquals(1, DAO.delete(5L));
    }
}
```

SonarQube

v0.1



January 21, 2021, 2:28 PM Version 0.0.1

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code | Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

1 Security Hotspots — 0.0% Reviewed — Security Review E

25min Debt — 4 Code Smells — Maintainability A

0.0% Coverage on 294 Lines to cover — - Unit Tests

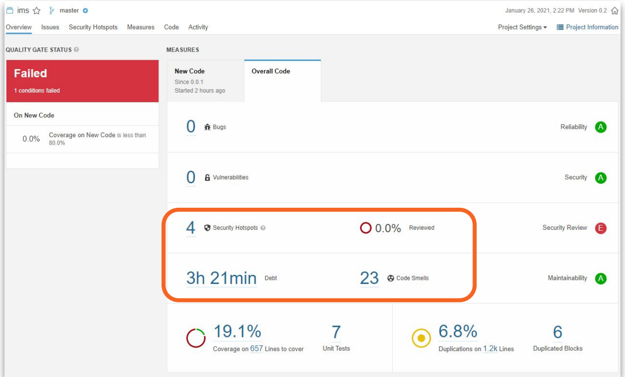
0.0% Duplications on 641 Lines — 0 Duplicated Blocks

ACTIVITY

Issues

January 21, 2021, 2:28 PM
0.0.1

v0.2



January 26, 2021, 2:22 PM Version 0.2

QUALITY GATE STATUS

Failed

1 conditions failed

On New Code

0.0% Coverage on New Code is less than 80.0%

MEASURES

New Code
Since 0.0.1
Started 2 hours ago

Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

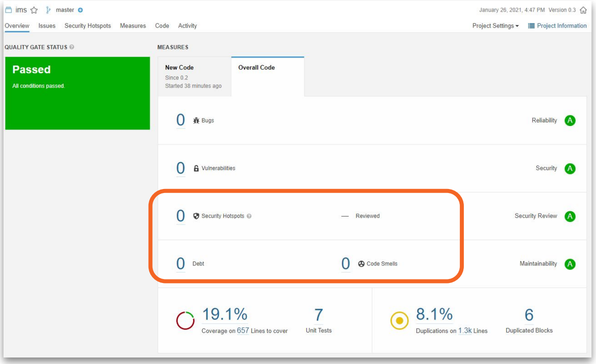
4 Security Hotspots — 0.0% Reviewed — Security Review E

3h 21min Debt — 23 Code Smells — Maintainability A

19.1% Coverage on 657 Lines to cover — 7 Unit Tests

6.8% Duplications on 1.2k Lines — 6 Duplicated Blocks

v0.3



January 26, 2021, 4:47 PM Version 0.3

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since 0.2
Started 38 minutes ago

Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

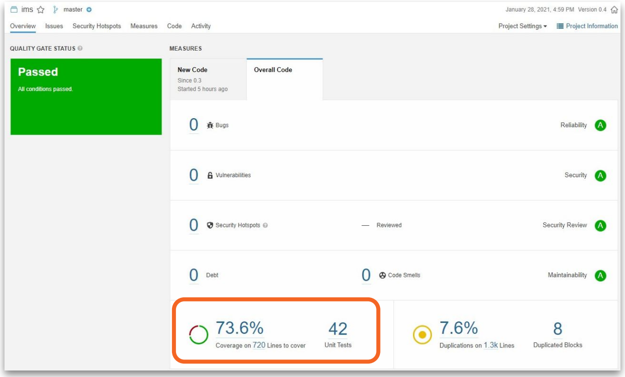
0 Security Hotspots — — Reviewed — Security Review A

0 Debt — 0 Code Smells — Maintainability A

19.1% Coverage on 657 Lines to cover — 7 Unit Tests

8.1% Duplications on 1.3k Lines — 6 Duplicated Blocks

v0.4



January 28, 2021, 4:59 PM Version 0.4

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since 0.3
Started 5 hours ago

Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

0 Security Hotspots — — Reviewed — Security Review A

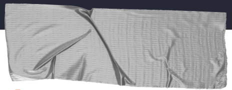
0 Debt — 0 Code Smells — Maintainability A

73.6% Coverage on 720 Lines to cover — 42 Unit Tests

7.6% Duplications on 1.3k Lines — 8 Duplicated Blocks

Select Administrator: Command Prompt

```
→ C:\Users\camer\Desktop\!qa-week-5\20DecSDET2-IMS-Starter\target› java -jar ims-fatjar-0.4.jar
```



An Idea

Using the Jira board to run through **user stories** could be fun?

Demonstration

Time for a live demonstration of the project.

Using a Jira board to plan a sprint proved it is a powerful tool for larger projects and group work.

For this project all sprint goals were achieved in time!

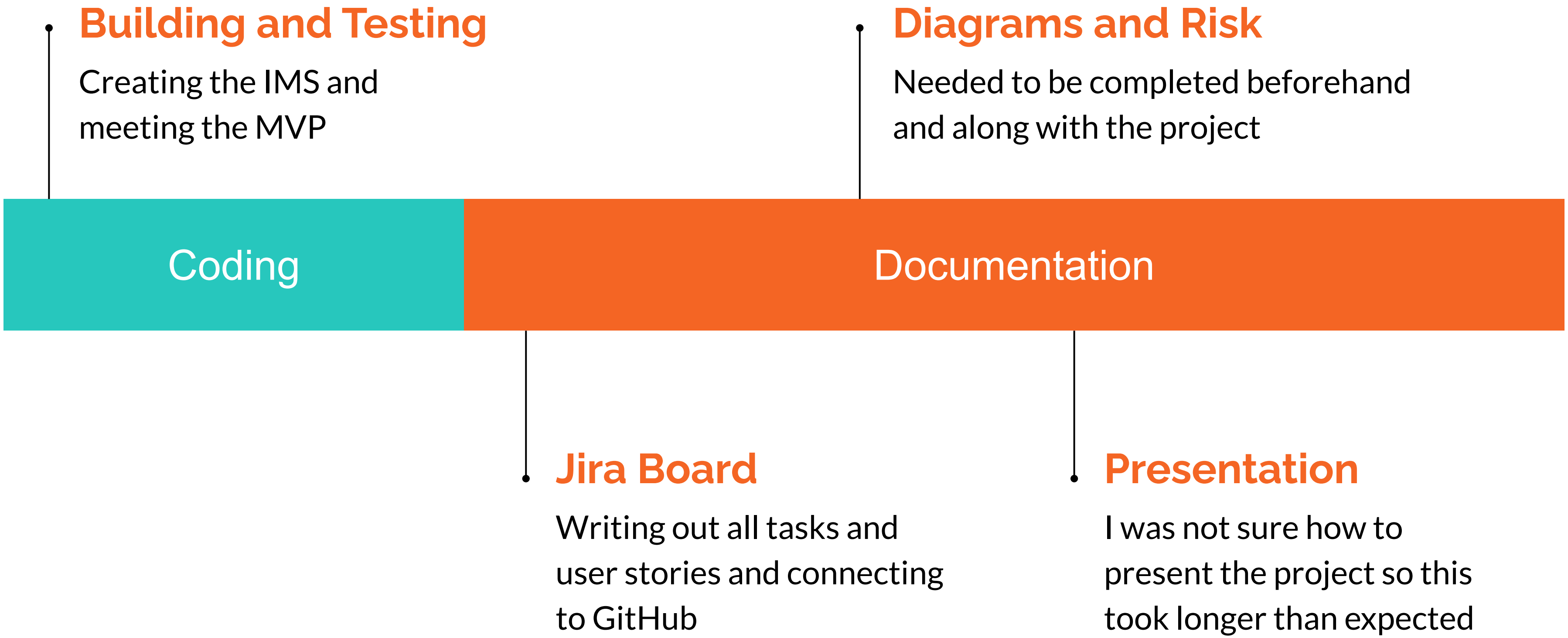
IMS Sprint

By the end of the sprint all of the coding needed to be complete:

- ➔ **Tasks**
  Anything that isn't a user story.
- ➔ **User Stories**
  Specific things that should be doable by someone using the product.

What went well?
What could be improved?

Time Management



Building and Testing

Creating the IMS and meeting the MVP

Diagrams and Risk

Needed to be completed beforehand and along with the project

Coding

Documentation

Jira Board

Writing out all tasks and user stories and connecting to GitHub

Presentation

I was not sure how to present the project so this took longer than expected

Conclusion

An exciting and slightly stressful journey to meeting the minimum viable product.

➔ **Thank you for listening**
I am aware the presentation was quite verbose

➔ **Do you have any questions?**
Feel free to ask!