Fundamental Project: Inventory Management System (IMS)

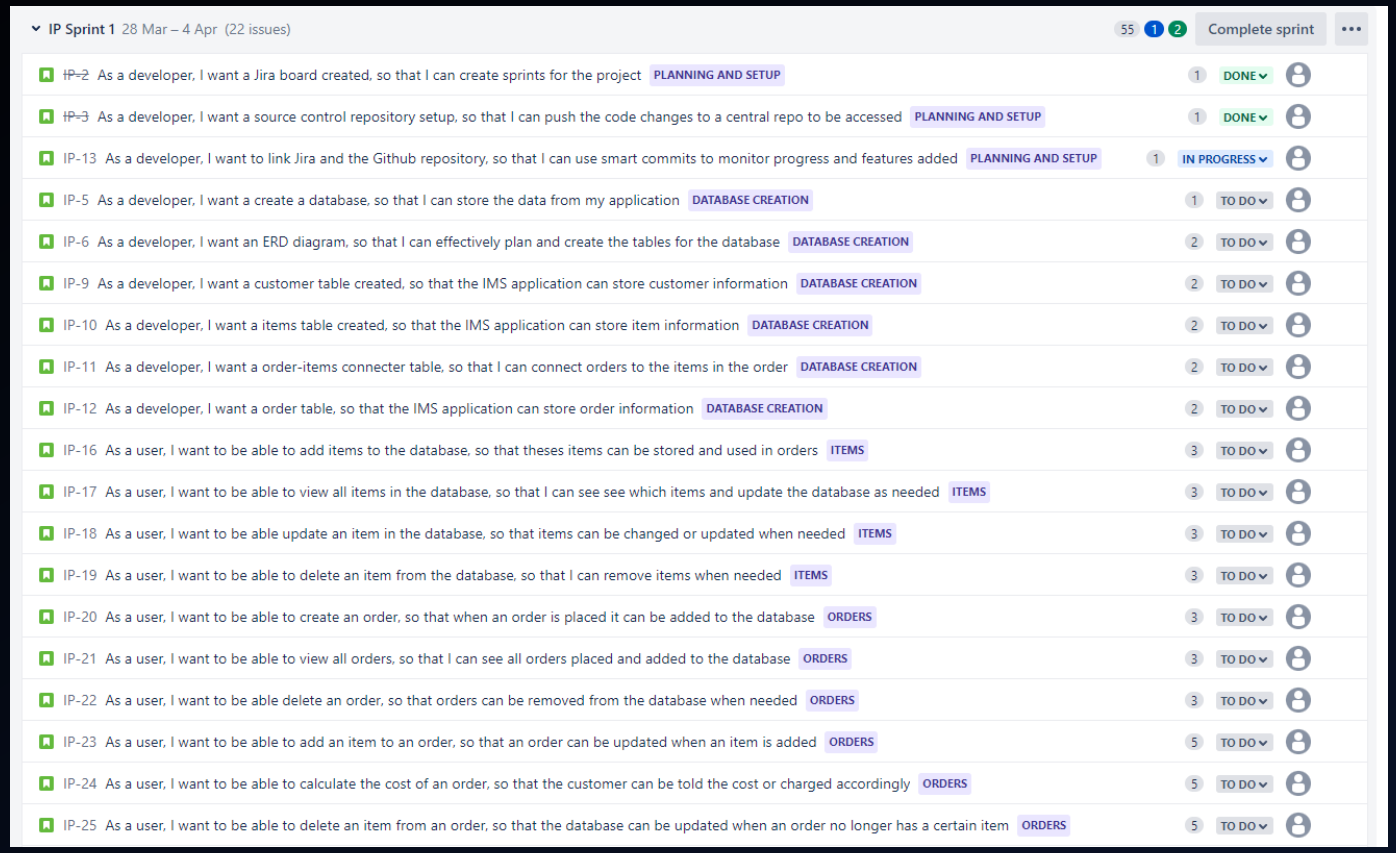
CALLUM ROBINSON
22FEBENABLE3

Introduction

Hi, I am Callum Robinson and this is how I approached this project:

- Firstly, broke down the specification logically from the MVP and scope and created user stories for all functionality needed by this system
- Added these user stories to a Jira project management board to plan out project with acceptance criteria, estimations and prioritisation
- Created the initial ERD diagram for the MySQL database to plan the database before starting to add the connections in java

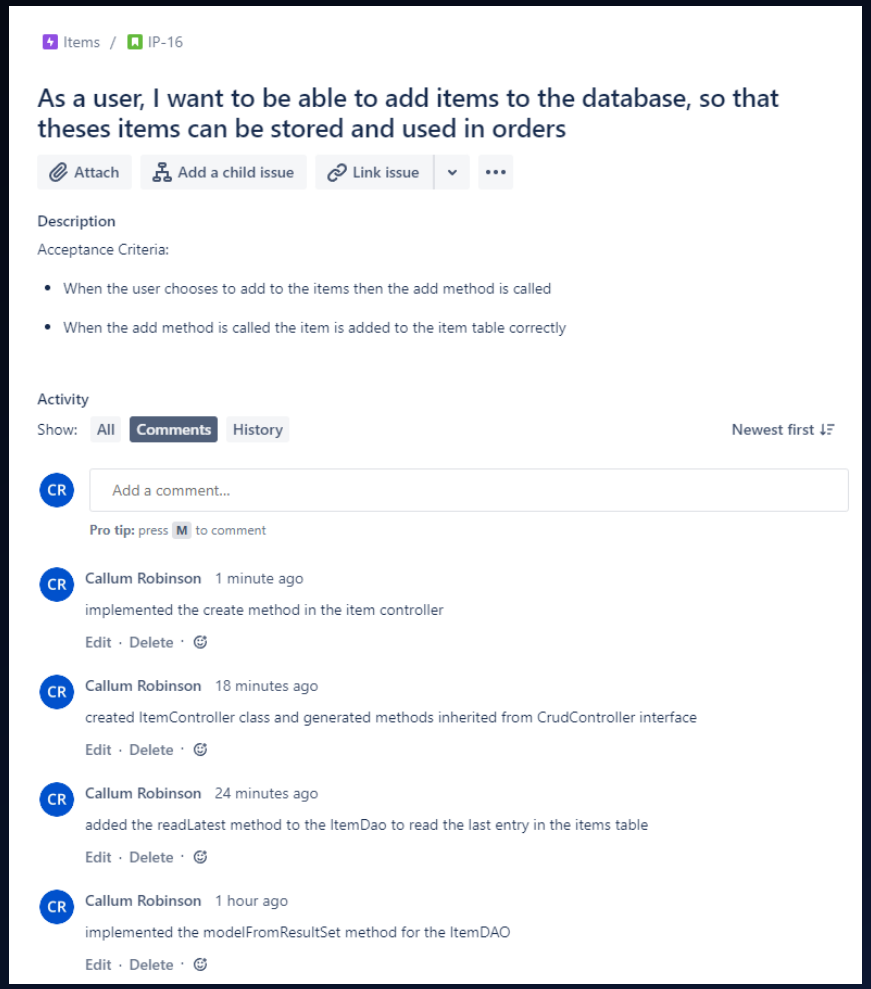
Jira



IP Sprint 1 28 Mar – 4 Apr (22 issues) | 55 | 1 | 2 | Complete sprint

~~IP-2~~	As a developer, I want a Jira board created, so that I can create sprints for the project	PLANNING AND SETUP	1	DONE
~~IP-3~~	As a developer, I want a source control repository setup, so that I can push the code changes to a central repo to be accessed	PLANNING AND SETUP	1	DONE
IP-13	As a developer, I want to link Jira and the Github repository, so that I can use smart commits to monitor progress and features added	PLANNING AND SETUP	1	IN PROGRESS
IP-5	As a developer, I want a create a database, so that I can store the data from my application	DATABASE CREATION	1	TO DO
IP-6	As a developer, I want an ERD diagram, so that I can effectively plan and create the tables for the database	DATABASE CREATION	2	TO DO
IP-9	As a developer, I want a customer table created, so that the IMS application can store customer information	DATABASE CREATION	2	TO DO
IP-10	As a developer, I want a items table created, so that the IMS application can store item information	DATABASE CREATION	2	TO DO
IP-11	As a developer, I want a order-items connecter table, so that I can connect orders to the items in the order	DATABASE CREATION	2	TO DO
IP-12	As a developer, I want a order table, so that the IMS application can store order information	DATABASE CREATION	2	TO DO
IP-16	As a user, I want to be able to add items to the database, so that theses items can be stored and used in orders	ITEMS	3	TO DO
IP-17	As a user, I want to be able to view all items in the database, so that I can see see which items and update the database as needed	ITEMS	3	TO DO
IP-18	As a user, I want to be able update an item in the database, so that items can be changed or updated when needed	ITEMS	3	TO DO
IP-19	As a user, I want to be able to delete an item from the database, so that I can remove items when needed	ITEMS	3	TO DO
IP-20	As a user, I want to be able to create an order, so that when an order is placed it can be added to the database	ORDERS	3	TO DO
IP-21	As a user, I want to be able to view all orders, so that I can see all orders placed and added to the database	ORDERS	3	TO DO
IP-22	As a user, I want to be able delete an order, so that orders can be removed from the database when needed	ORDERS	3	TO DO
IP-23	As a user, I want to be able to add an item to an order, so that an order can be updated when an item is added	ORDERS	5	TO DO
IP-24	As a user, I want to be able to calculate the cost of an order, so that the customer can be told the cost or charged accordingly	ORDERS	5	TO DO
IP-25	As a user, I want to be able to delete an item from an order, so that the database can be updated when an order no longer has a certain item	ORDERS	5	TO DO

The sprint backlog



Items / IP-16

As a user, I want to be able to add items to the database, so that theses items can be stored and used in orders

Attach | Add a child issue | Link issue

Description

Acceptance Criteria:

- When the user chooses to add to the items then the add method is called
- When the add method is called the item is added to the item table correctly

Activity

Show: All | Comments | History | Newest first

Add a comment...

Pro tip: press M to comment

Callum Robinson 1 minute ago
implemented the create method in the item controller
Edit · Delete

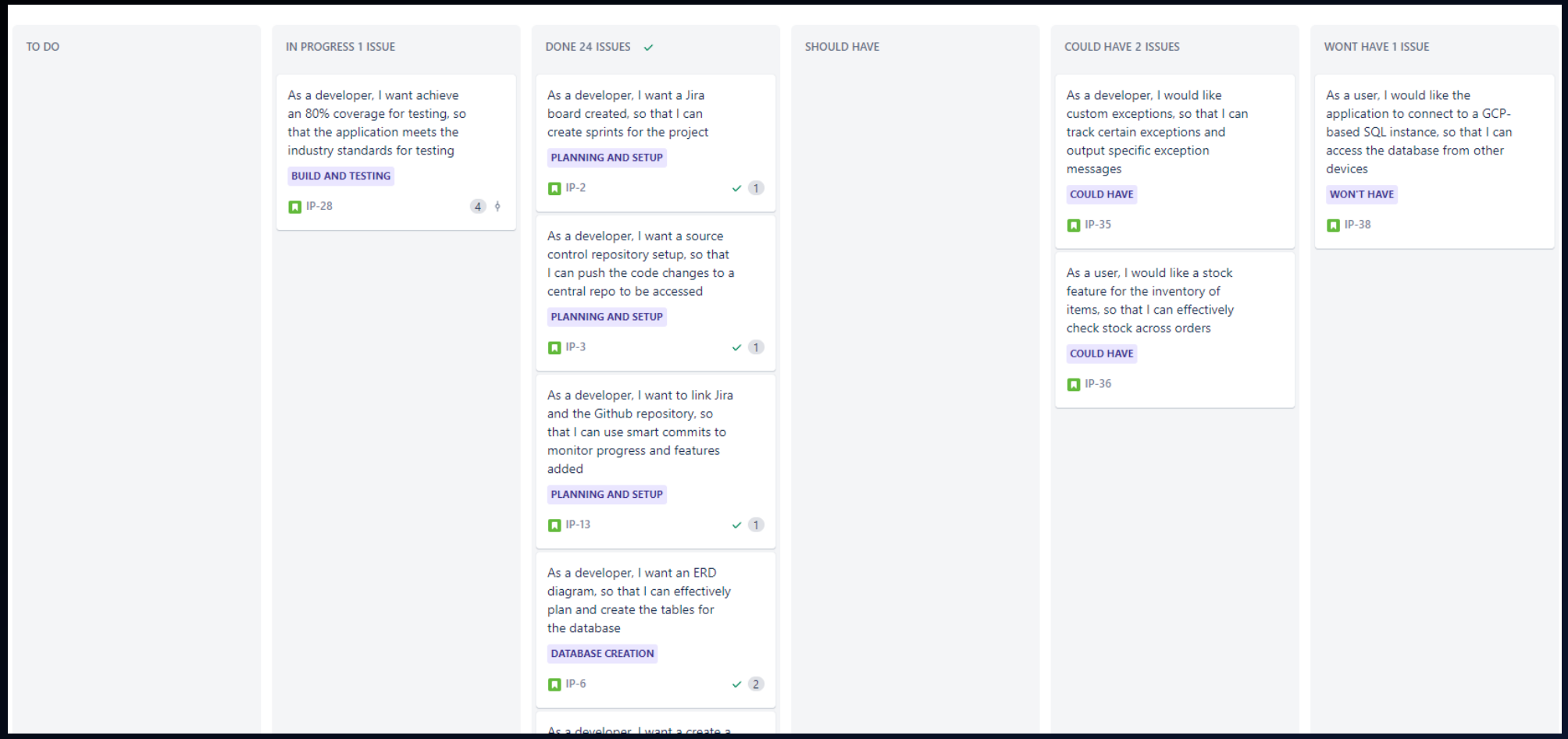
Callum Robinson 18 minutes ago
created ItemController class and generated methods inherited from CrudController interface
Edit · Delete

Callum Robinson 24 minutes ago
added the readLatest method to the ItemDao to read the last entry in the items table
Edit · Delete

Callum Robinson 1 hour ago
implemented the modelFromResultSet method for the ItemDAO
Edit · Delete

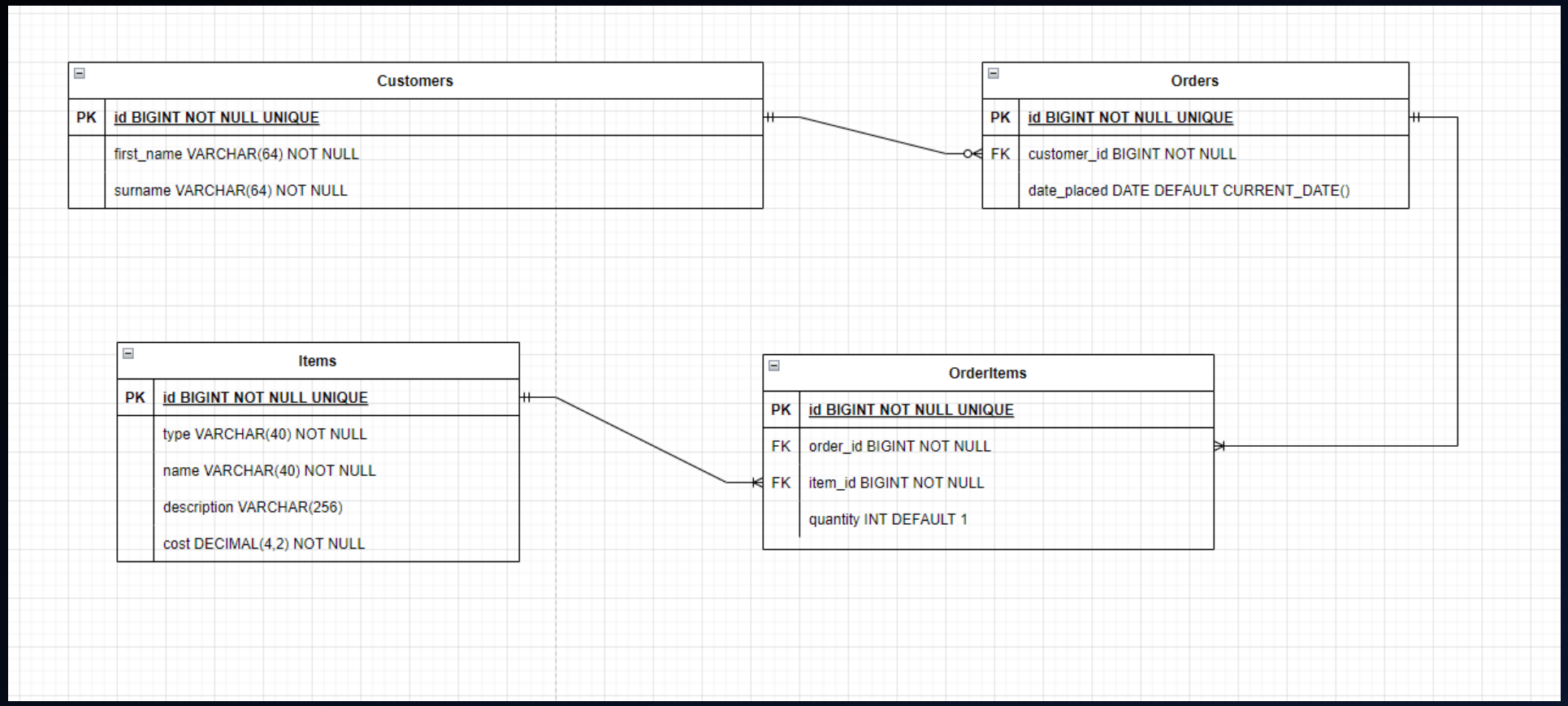
An example of a backlog item with:

- acceptance criteria
- examples of smart commits used



Sprint board with MoSCoW separation (note this is later with most Must Haves and Should Haves completed

Initial ERD



ERD diagram created using: https://app.diagrams.net/

MySQL for the Database

```sql
drop schema ims;

CREATE SCHEMA IF NOT EXISTS `ims`;

USE `ims` ;

CREATE TABLE IF NOT EXISTS `ims`.`customers` (
    `id` BIGINT NOT NULL AUTO_INCREMENT,
    `first_name` VARCHAR(64) NOT NULL,
    `surname` VARCHAR(64) NOT NULL,
    PRIMARY KEY (`id`)
);

CREATE TABLE IF NOT EXISTS `ims`.`items` (
    `id` BIGINT NOT NULL AUTO_INCREMENT,
    `type` VARCHAR(40) NOT NULL,
    `name` VARCHAR(40) NOT NULL,
    `description` VARCHAR(256),
    `cost` DECIMAL(4,2) NOT NULL,
    PRIMARY KEY (`id`)
);

```

```sql
CREATE TABLE IF NOT EXISTS `ims`.`orders` (
    `id` BIGINT NOT NULL AUTO_INCREMENT,
    `customer_id` BIGINT NOT NULL,
    `date_placed` DATE DEFAULT(DATE(CURRENT_TIMESTAMP)),
    PRIMARY KEY (`id`),
    FOREIGN KEY (`customer_id`) REFERENCES `customers`(`id`)
);


CREATE TABLE IF NOT EXISTS `ims`.`orderitems` (
    `order_id` BIGINT NOT NULL,
    `item_id` BIGINT NOT NULL,
    `quantity` INT DEFAULT (1),
    PRIMARY KEY (`order_id`, `item_id`),
    FOREIGN KEY (`order_id`) REFERENCES `orders`(`id`),
    FOREIGN KEY (`item_id`) REFERENCES `items`(`id`)
);
```

Technologies learned for this project

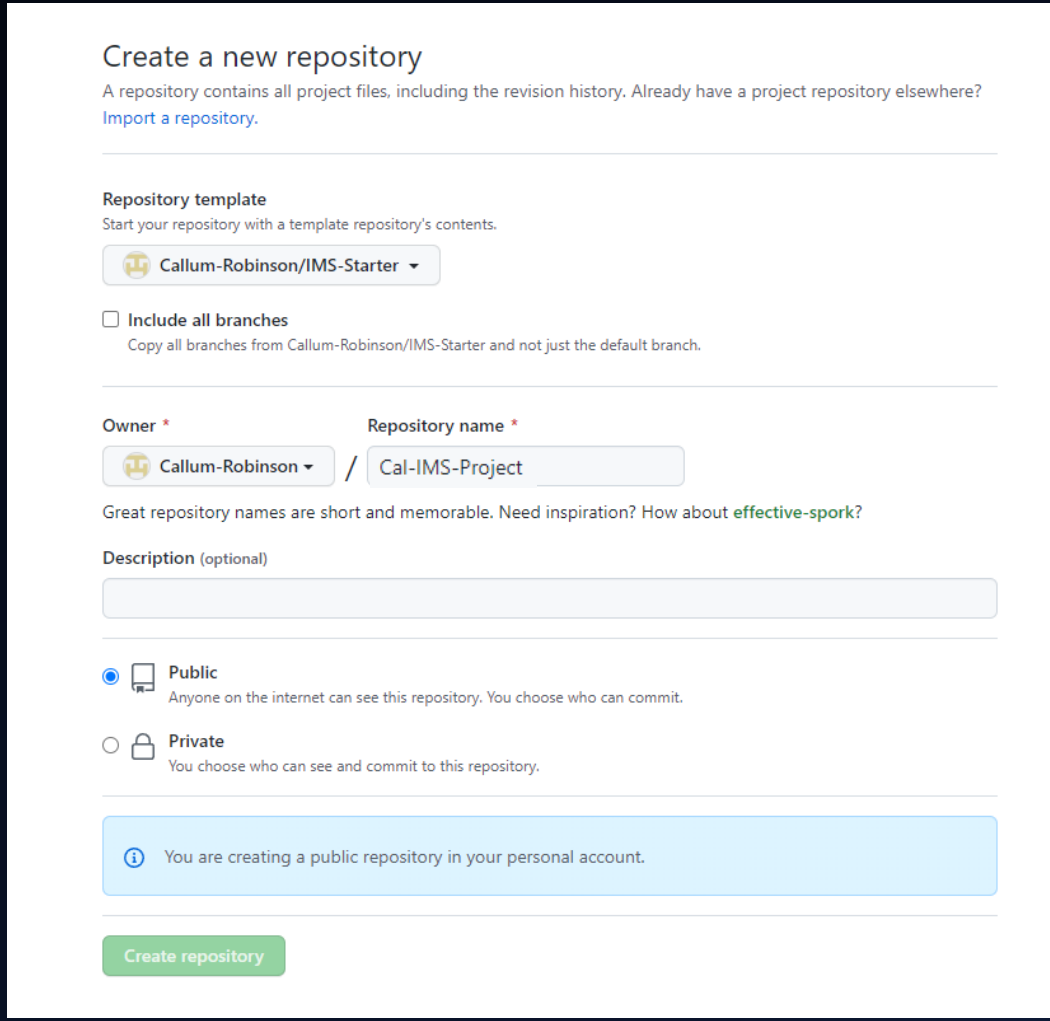
- Jira – project management boards
- Git – version control
- Github – source control management
- MySQL – Databases
- Java – back-end programming language
- Maven – build tool
- JUnit – unit testing
- Mockito – create test objects in unit tests

Version control

- Generated repository using IMS-Starter template forked from https://github.com/JHarry444/IMS-Starter

- Cloned the repository using git bash terminal:

```
calro@DESKTOP-N5LSPFU MINGW64 ~/documents/Java-Learning/IMS-Project
$ git clone https://github.com/Callum-Robinson/Cal-IMS-Project.git
```



- Used a master/dev/feature branch model for each part of functionality added
- This was done as follows:

```
calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (master)
$ git checkout -b dev
```

```
calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (dev)
$ git pull origin master
```

```
calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (dev)
$ git checkout -b feature/items
```

```
calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (feature/items)
$ git pull origin dev
```

- A git bash terminal inside Eclipse was used to stage, commit and push changes to the feature branches:

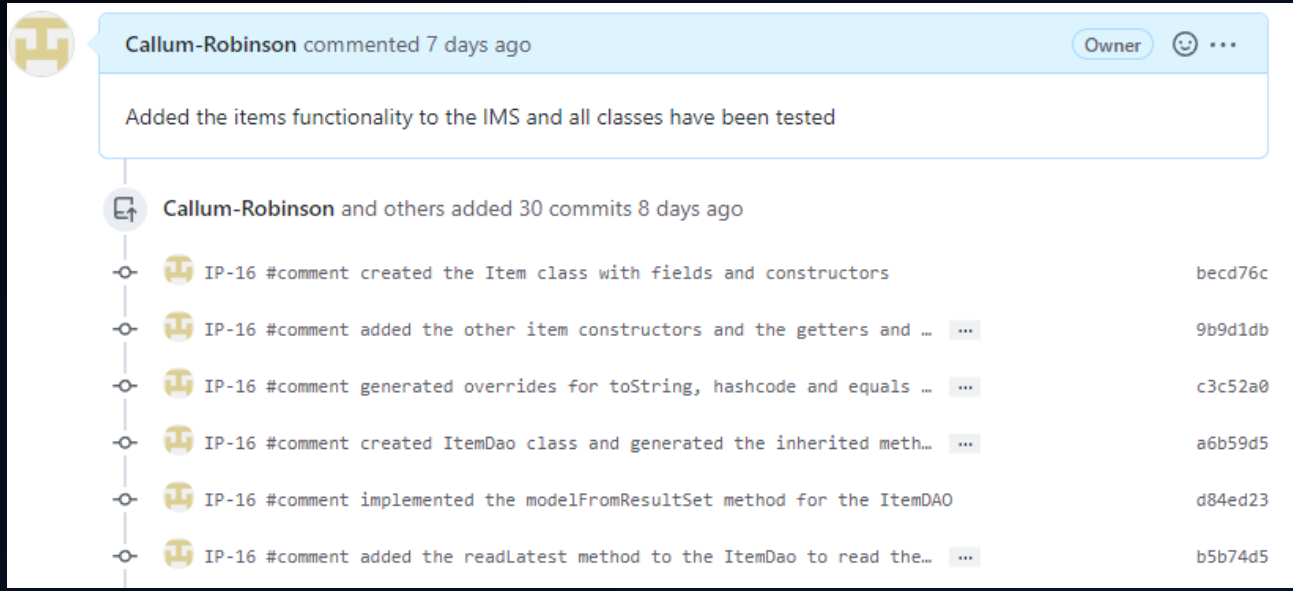
```
calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (feature/items)
$ git add *

calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (feature/items)
$ git commit -m "IP-XX #comment The changes made in commit"

calro@DESKTOP-N5LSPFU MINGW64 ~/Documents/Java-Learning/IMS-Project/Cal-IMS-Project (feature/items)
$ git push -u origin feature/items
```

Note: The IP-XX is the item key on Jira followed by the comment of the changes

- When the feature functionality is completed then create a pull request to the dev branch:



- When the wanted functionality is completed then merge the dev branch into the master branch (using a pull request)

Testing

- Test classes were created for each:
  - Domain
  - DAO
  - Controller
- An example for each type will be shown for Items

Item Test

- The Item test class simply tests the hashcode and equals method using the EqualsVerifier class:

```java
public class ItemTest {

    /*
     * Tests the hashcode and equals method using the EqualsVerifier class
     */
    @Test
    public void testEquals() {
        EqualsVerifier.simple().forClass(Item.class).verify();
    }
}

```

Finished after 0.163 seconds

Runs: 1/1 Errors: 0 Failures: 0

com.qa.ims.persistence.domain.ItemTest [Runne

Item DAO test

- Test each method in the Item DAO after setting up the test database connection:

```
private final ItemDAO DAO = new ItemDAO();

/*
 * Setup the connection with the test database
 */
@Before
public void setup() {
    DBUtils.connect();
    DBUtils.getInstance().init("src/test/resources/sql-schema.sql", "src/test/resources/sql-data.sql");
}
```

- Test the method to create an item in the database:

```
/*
 * Test the create method in the Item DAO
 */
@Test
public void testCreate() {
    final Item created = new Item (2L, "Cake", "Chocolate fudge cake", "Chocolatey fudge cake", 4.89);
    assertEquals(created, DAO.create(created));
}
```

- Test the method to read the latest item in the database:

```
/*
 * Test the read latest method in the Item DAO
 */
@Test
public void testReadLatest() {
    assertEquals(new Item (1L, "Ice cream", "Strawberry ice cream", "Classic strawberry flavoured ice cream", 2.99),
            DAO.readLatest());
}
```

- Test the method to read all items in the database:

```
/*
 * Test the read all method in the Item DAO
 */
@Test
public void testReadAll() {
    List<Item> expected = new ArrayList<>();
    expected.add(new Item(1L, "Ice cream", "Strawberry ice cream", "Classic strawberry flavoured ice cream", 2.99));
    assertEquals(expected, DAO.readAll());
}
```

- Test the method to read an item in the database by id:

```
/*
 * Test the read by id method
 */
@Test
public void testRead() {
    final long ID = 1L;
    assertEquals(new Item(ID, "Ice cream", "Strawberry ice cream", "Classic strawberry flavoured ice cream", 2.99),
            DAO.read(ID));
}
```

- Test the method to update an item in the database:

```
/*
 * Test the update method
 */
@Test
public void testUpdate() {
    final Item updated = new Item(1L, "Cake", "Chocolate fudge cake", "Chocolatey fudge cake", 4.89);
    assertEquals(updated, DAO.update(updated));
}
```

- Test the method to delete an item in the database:

```
/*
 * Test the delete method
 */
@Test
public void testDelete() {
    assertEquals(1, DAO.delete(1));
}
```

Item Controller test

- Mock the Utils and ItemDAO and then inject them into the Item Controller:

```java
@RunWith(MockitoJUnitRunner.class)
public class ItemControllerTest {

    /*
     * Mock the utils and DAO and inject into the controller
     */
    @Mock
    private Utils utils;

    @Mock
    private ItemDAO dao;

    @InjectMocks
    private ItemController controller;
```

- Then test each of the methods in the controller using Mockito to mock the user inputs and ItemDAO methods

- Test the method that takes item fields from the user input and calls the DAO create method:

```java
/*
 * Test the create method in the Item controller by mocking the user input and the DAO create method
 */
@Test
public void testCreate(){
    final String TYPE = "Donut", NAME = "Ring donut", DESC = "Fresh sugared donut";
    final Double COST = 1.20;
    final Item created = new Item(TYPE, NAME, DESC, COST);

    Mockito.when(utils.getString()).thenReturn(TYPE, NAME, DESC);
    Mockito.when(utils.getDouble()).thenReturn(COST);
    Mockito.when(dao.create(created)).thenReturn(created);

    assertEquals(created, controller.create());

    Mockito.verify(utils, Mockito.times(3)).getString();
    Mockito.verify(utils, Mockito.times(1)).getDouble();
    Mockito.verify(dao, Mockito.times(1)).create(created);
}
```

Create constants

Mock the user input and DAO method

Check the return is as expected

Verify the mocked methods are called correctly

- Test the method that takes the data from the DAO readall method and outputs it to the console:

```
/*
 * Test the read all method in the Item controller by mocking the DAO read all method
 */
@Test
public void testReadAll() {
    List <Item> items = new ArrayList<>();
    items.add(new Item (1L, "Ice cream", "Strawberry ice cream", "Classic strawberry flavoured ice cream", 2.99));

    Mockito.when(dao.readAll()).thenReturn(items);

    assertEquals(items, controller.readAll());

    Mockito.verify(dao, Mockito.times(1)).readAll();
}
```

- Test the method that takes the updated data from the user and passes the updated item to the DAO update method:

```java
/*
 * Test the update method in the Item controller by mocking the user input and DAO update method
 */
@Test
public void testUpdate() {
    Item updated = new Item (2L, "Cake", "Chocolate fudge cake", "Chocolatey fudge cake", 4.89);

    Mockito.when(this.utils.getLong()).thenReturn(2L);
    Mockito.when(this.utils.getString()).thenReturn(updated.getType(), updated.getName(), updated.getDescription());
    Mockito.when(this.utils.getDouble()).thenReturn(updated.getCost());
    Mockito.when(this.dao.update(updated)).thenReturn(updated);

    assertEquals(updated, this.controller.update());

    Mockito.verify(this.utils, Mockito.times(1)).getLong();
    Mockito.verify(this.utils, Mockito.times(3)).getString();
    Mockito.verify(this.utils, Mockito.times(1)).getDouble();
    Mockito.verify(this.dao, Mockito.times(1)).update(updated);
}
```

- Test the method that takes the Item id from the user and calls the DAO delete method for that id:

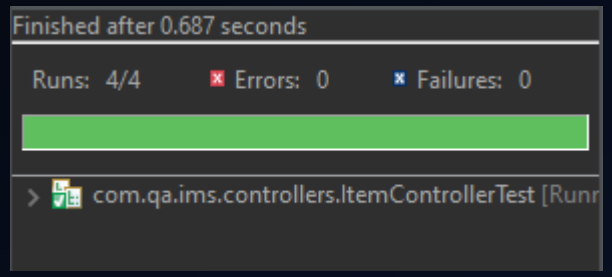
```java
/*
 * Test the delete method in the Item controller by mocking the user input and the DAO delete method
 */
@Test
public void testDelete() {
    final long ID = 1L;

    Mockito.when(utils.getLong()).thenReturn(ID);
    Mockito.when(dao.delete(ID)).thenReturn(1);

    assertEquals(1, this.controller.delete());

    Mockito.verify(utils, Mockito.times(1)).getLong();
    Mockito.verify(dao, Mockito.times(1)).delete(ID);
}
```

- Results of the test file:



Test Coverage

- The test coverage of src/main/java:

src/main/java	62.4 %	2,083	1,257	3,340
com.qa.ims.controller	41.2 %	340	485	825
OrderController.java	13.9 %	44	272	316
Action.java	0.0 %	0	119	119
AddOrRemove.java	0.0 %	0	94	94
CustomerController.java	100.0 %	109	0	109
ItemController.java	100.0 %	141	0	141
OrderItemController.java	100.0 %	46	0	46
com.qa.ims.persistence.dao	77.6 %	907	262	1,169
OrderDAO.java	78.5 %	256	70	326
OrderItemDAO.java	76.9 %	226	68	294
CustomerDAO.java	76.0 %	196	62	258
ItemDAO.java	78.7 %	229	62	291
com.qa.ims.persistence.domain	73.1 %	672	247	919
Domain.java	0.0 %	0	105	105
Order.java	61.5 %	144	90	234
OrderItem.java	77.0 %	174	52	226
Customer.java	100.0 %	175	0	175
Item.java	100.0 %	179	0	179
com.qa.ims	0.0 %	0	164	164
IMS.java	0.0 %	0	148	148
Runner.java	0.0 %	0	16	16
com.qa.ims.utils	63.1 %	164	96	260
Utils.java	14.6 %	12	70	82
DBUtils.java	85.4 %	152	26	178

What wasn’t tested

- Methods in the Order Controller
  - Confusion with Mockito due to the methods using the OrderItem Controller and DAO in addition to the Order DAO and Utils

- The enums Action, Domain and AddOrRemove

Demonstration

- I will demonstrate the following selection of user stories:
  - As a user, I want to be able to add items to the database, so that theses items can be stored and used in orders
  - As a user, I want to be able to view all orders, so that I can see all orders placed and added to the database
  - As a user, I want to be able delete an order, so that orders can be removed from the database when needed

Create Item Demonstration

```
Welcome to the Inventory Management System!
Which entity would you like to use?
CUSTOMER: Information about customers
ITEM: Individual Items
ORDER: Purchases of items
STOP: To close the application
item
What would you like to do with item:
CREATE: To save a new entity into the database
READ: To read an entity from the database
UPDATE: To change an entity already in the database
DELETE: To remove an entity from the database
RETURN: To return to domain selection
create
Please enter the type of item
Crisps
Please enter the name of the item
Quavers
Please enter a description of the item
Cheese and onion quavers
Please enter the cost of the item
1.99
Item created
```

User picks Item

User picks create

User enters the fields for the item

	id	type	name	description	cost
▶	4	Crisps	Quavers	Cheese and onion quavers	1.99

New entry into table (selected only this new item)

Read all Orders Demonstration

```
Welcome to the Inventory Management System!
Which entity would you like to use?
CUSTOMER: Information about customers
ITEM: Individual Items
ORDER: Purchases of items
STOP: To close the application
order
What would you like to do with order:
CREATE: To save a new entity into the database
READ: To read an entity from the database
UPDATE: To change an entity already in the database
DELETE: To remove an entity from the database
RETURN: To return to domain selection
read
Id: 1, Customer id: 1, Customer name: jason lloyd, Date placed: 2020-02-01, Items in order: [[Item Id: 1, Item Type: Cheese, Item Name: Cheddar, Item Cost: 2.39, Quantity: 3]]
Total cost of order = 7.17
Id: 2, Customer id: 2, Customer name: Josh Edwards, Date placed: 2022-03-31, Items in order: [[Item Id: 1, Item Type: Cheese, Item Name: Cheddar, Item Cost: 2.39, Quantity: 8]]
Total cost of order = 19.12
```

User picks Order

User picks Read

As shown this lists all orders with:

- Order id
- Customer id
- Customer name (uses a JOIN with the customer table and concatenates the Strings)
- Date placed
- A list of items in the order (with item id, item type, item name, item cost and quantity)
- The total calculated cost of the order

Add Item to Order Demonstration

```
Welcome to the Inventory Management System!
Which entity would you like to use?
CUSTOMER: Information about customers
ITEM: Individual Items
ORDER: Purchases of items
STOP: To close the application
order
What would you like to do with order:
CREATE: To save a new entity into the database
READ: To read an entity from the database
UPDATE: To change an entity already in the database
DELETE: To remove an entity from the database
RETURN: To return to domain selection
update
What would you like to do?
ADD: Add an item to an order
REMOVE: Remove an item from an order
RETURN: To return
add
Enter the id of the order
1
Please enter the item id
4
Please enter the quantity of the item
8
Item added to order
Do you wish to add another?
no
```

User picks Order

User picks Update

User picks Add

User enters id of order to add to

User enters id of item to add

User enters the quantity of the item

User is asked if they would like to add another (this would loop for another item addition if yes)

order_id	item_id	type	name	cost	quantity
1	1	Cheese	Cheddar	2.39	3
1	4	Crisps	Quavers	1.99	8

Item is added to Order

Sprint Review

- What was completed
  - The MVP for the Application
  - Made the outputs more readable for the different read functionalities
  - Implemented the functionality that when an customer is deleted then their orders are also removed and similarly when an item is removed then it is removed from any order it is a part of

- What was left behind
  - Custom exceptions to help track errors better
  - A stock feature that compared the items ordered to their stock available
  - Using a GCP-based SQL instance so that it can be accessed from other devices

Sprint Retrospective

- What went well
  - All CRUD functionality implemented
  - Jira board and ERD planned well
  - Classes readable with effective comment placement throughout
  - Regular use of smart commits to ease progress tracking both on the repository and on the Jira board
- What could be improved
  - More practice with estimation on user stories
  - Planning the UML layout of the classes beforehand
  - Increased test coverage
  - Possibly more feature branches merged into dev more often

Conclusion

- An effective Inventory Management System created with good practice of everything learned so far in the course
- Definite increased motivation and inspiration for future projects

- Revision of Mockito is needed
- More experience with planning with UML and with estimations

Any Questions?