IMS Project

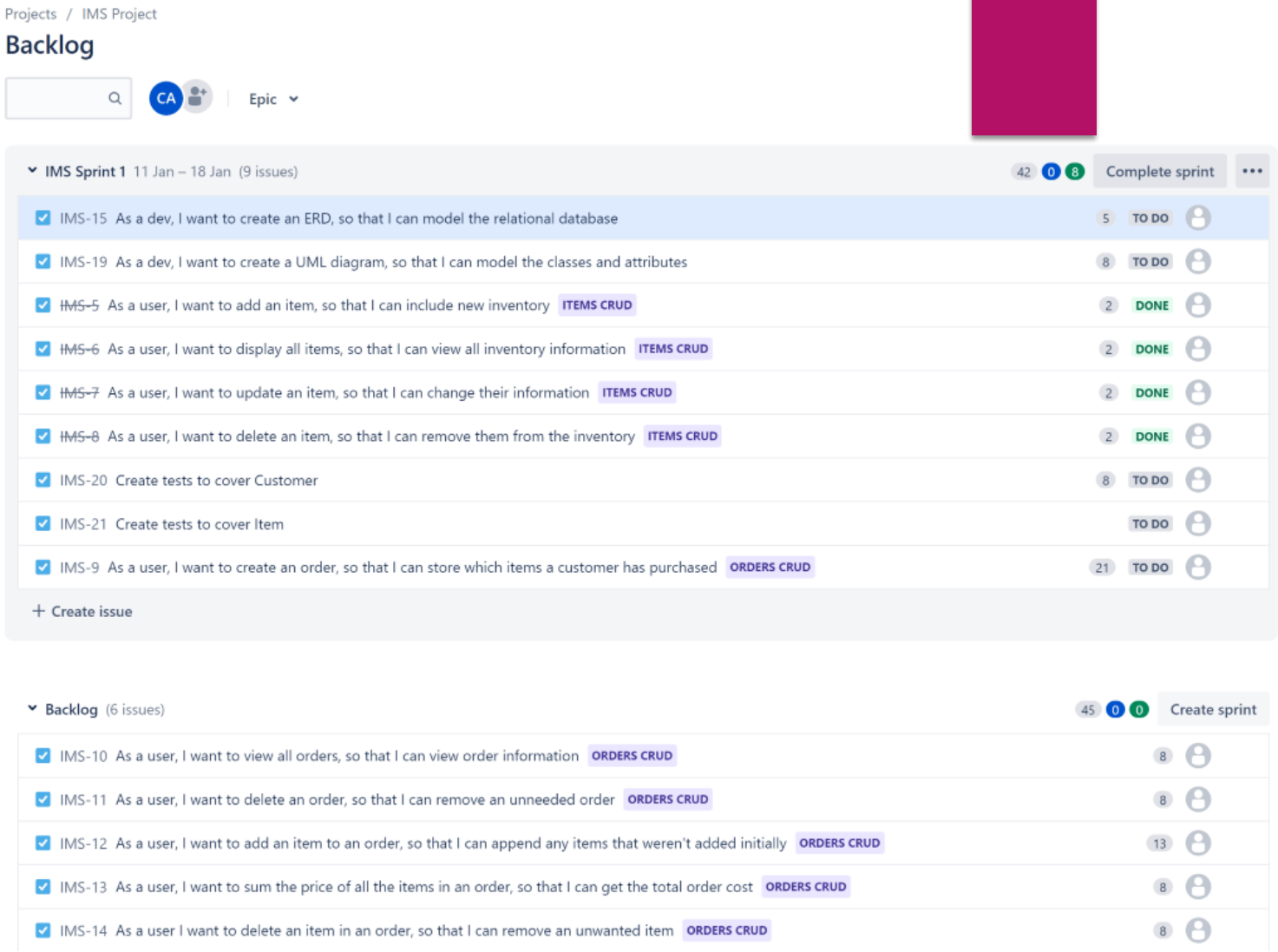
CLAES ALFONSO

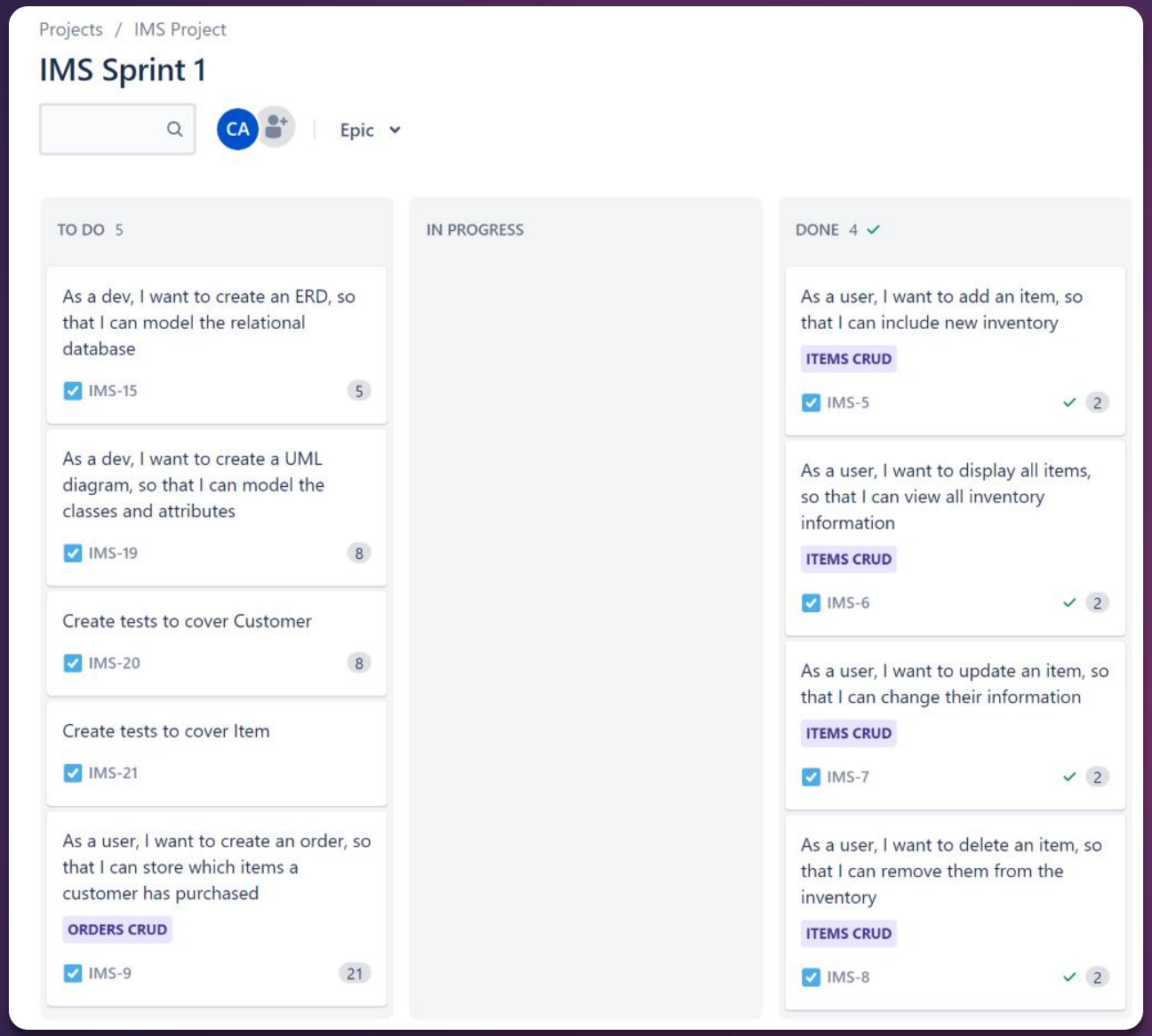
Introduction

- **Project Objective:** Develop an application using Java to interact with an SQL database.
- The following technologies were used for this project:
  - **Version Control System**: Git
  - **Source Code Management**: GitHub
  - **Project Management :** Jira
  - **Database Management System**: MySQL (local MySQL Workbench or GCP instance)
  - **Back-End Programming Language**: Java
  - **Build Tool**: Maven
  - **Unit Testing**: JUnit

Project Management Overview

- Sprints were split roughly evenly to spread the workload, using the Story Point Estimates.
- User stories split into epics for each database table.
- Items CRUD stories completed.

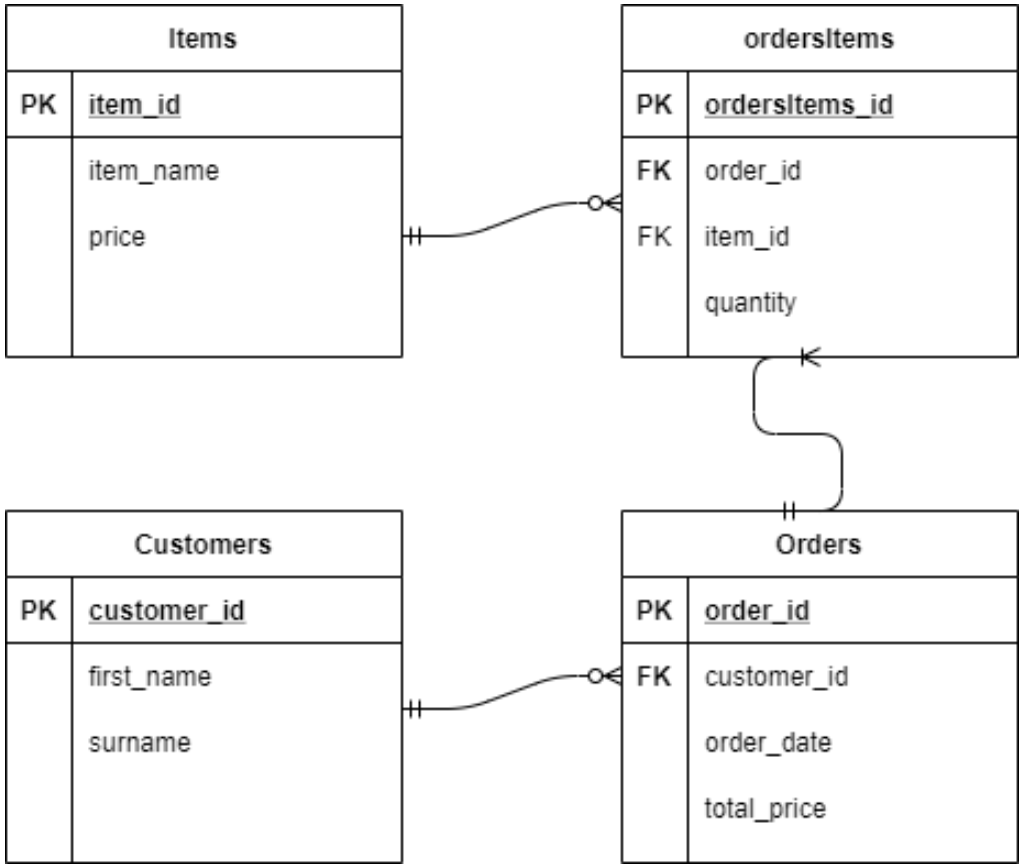




Kanban Board: Sprint 1

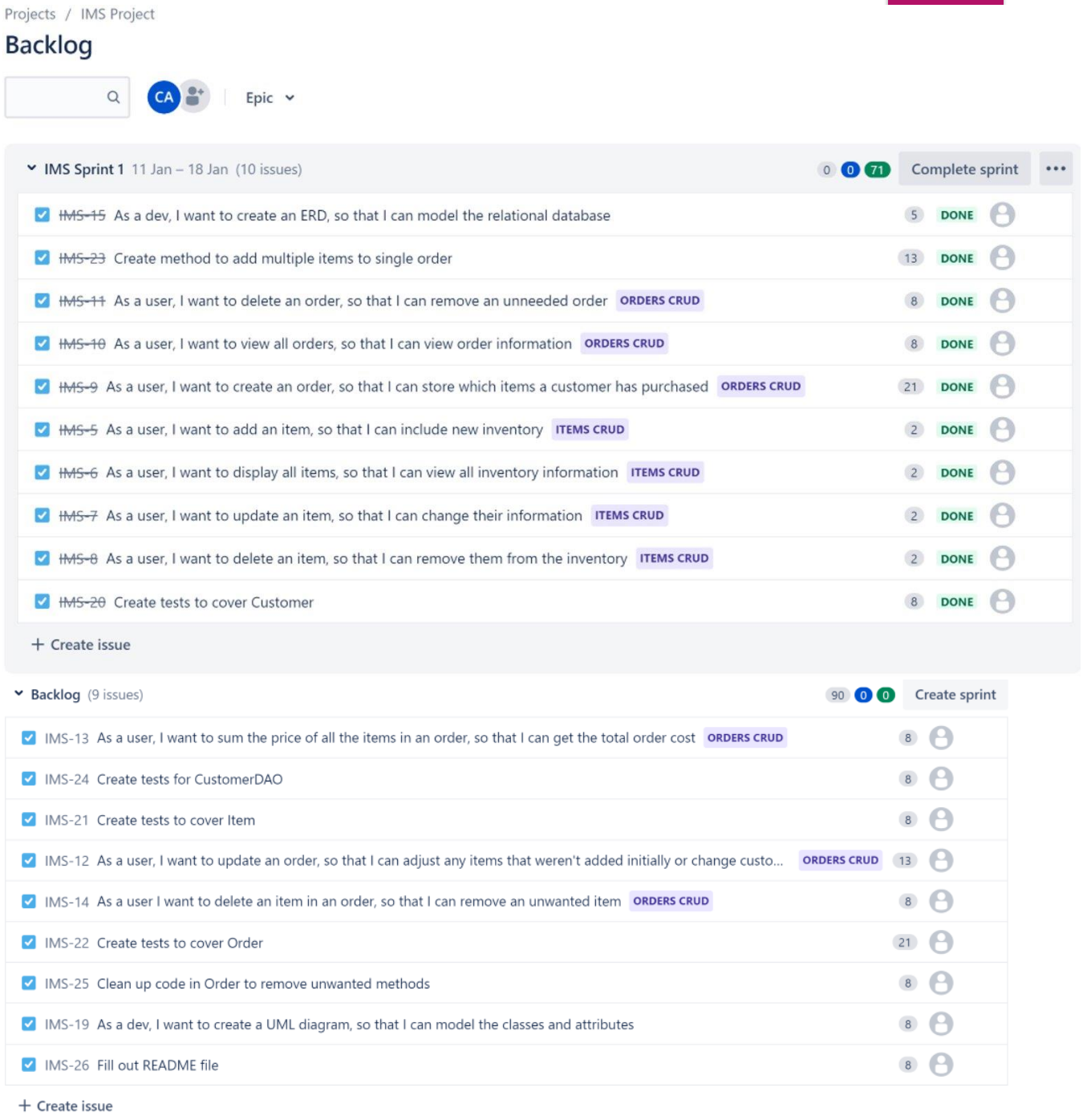
Modelling the SQL Database

- An Entity Relationship Diagram was used to model the relationships between database tables and entries.
- As SQL doesn't support a many-to-many relationship, an intermediary table "ordersItems" is used.
- This was especially useful when writing a method to add multiple items to one order.



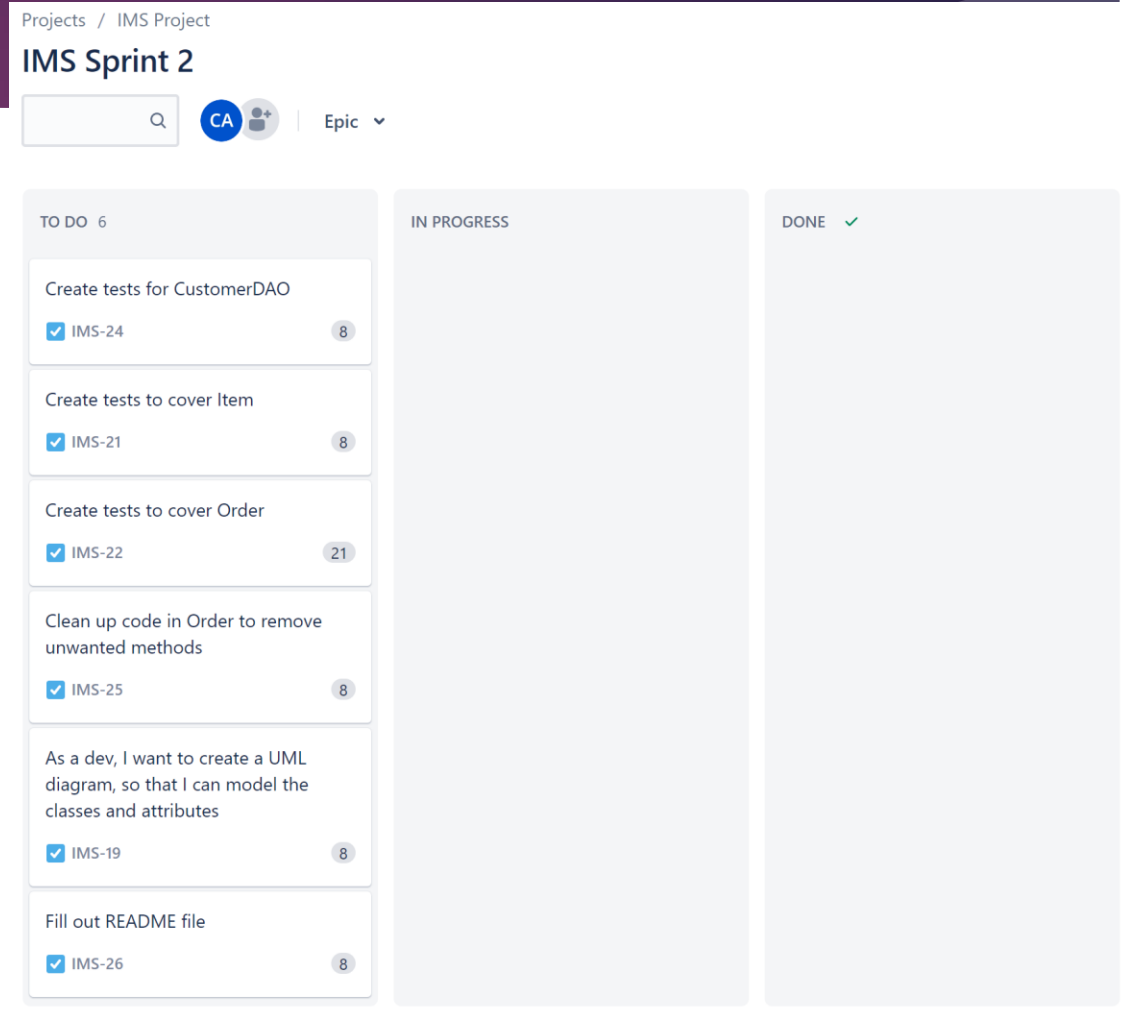
End of Sprint 1

- By the end of Sprint 1, I had completed almost all MVP functionality
- Added additional tasks to break up larger methods into smaller ones.



Sprint 2

- A problem occurred in the middle of Sprint 2 as I had signed up using my QA email.
- This led to another user within the academytrainee.atlassian domain accidentally deleting my sprint.
- Sprint 2 was then remade.



User Story Example 1

- As a user, I want to create an Order, so that I can store which Items a Customer has purchased.

```java
public Order create() {
    boolean done = false;
    List<Long> items = new ArrayList<>();
    List<Integer> quantities = new ArrayList<>();
    LOGGER.info("Please enter the customer id");
    Long customer_id = Long.valueOf(getInput());
    LOGGER.info("Please enter the item id");
    items.add(Long.valueOf(getInput()));
    LOGGER.info("Please enter the quantity");
    quantities.add(Integer.valueOf(getInput()));
    while (!done) {
        LOGGER.info("Add more items? y/n");
        String yn = String.valueOf(Utils.getInstance().getInput().toUpperCase());
        if (yn.equals("Y")) {
            LOGGER.info("Please enter the item id");
            items.add(Long.valueOf(getInput()));
            LOGGER.info("Please enter the quantity");
            quantities.add(Integer.valueOf(getInput()));
        } else if (yn.equals("N")) {
            done = true;
        } else {
            LOGGER.info("Invalid selection please try again");
        }
    }
    LOGGER.info("Please enter the date");
    String date = getInput();
    Order order = orderService.create(new Order(customer_id, items, date).quantities(quantities));

    LOGGER.info("-Order created-");
    return order;
}
```

(Order Controller)

User Story Example 1

- As a user, I want to create an Order, so that I can store which Items a Customer has purchased.

```
public Order create(Order order) {
    try (Connection connection = DBUtils.getInstance().getConnection();
            Statement statement = connection.createStatement();) {
        statement.executeUpdate("INSERT INTO orders(customer_id, order_date) values('" + order.getCustomer_id()
                + "','" + order.getOrder_date() + "')");
        List<Long> items_id = new ArrayList<Long>();
        items_id = order.getItems_id();
        List<Integer> quantity = new ArrayList<Integer>();
        quantity = order.getQuantities();
        int j = 0;
        for (Long i : items_id) {
            statement.executeUpdate("INSERT INTO ordersItems(order_id, item_id, quantity) values('"
                    + readLatestOrderID() + "','" + i + "','" + quantity.get(j) + "')");
            j++;
        }
        statement.executeUpdate(
                "update orders set total_price ='" + calcTotalPrice() + "' where order_id =" + readLatestOrderID());
        return readLatest();
    } catch (Exception e) {
        LOGGER.debug(e.getStackTrace());
        LOGGER.error(e.getMessage());
    }
    return null;
}
```

(Order DAO)

User Story Example 2

- As a user, I want to sum the price of all Items in an Order, so that I can calculate the total order cost

```
public Double calcTotalPrice() {
    try (Connection connection = DBUtils.getInstance().getConnection();
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(
                    "SELECT SUM(price*quantity) AS total FROM items i JOIN ordersItems oi ON i.item_id=oi.item_id WHERE order_id="
                            + readLatestOrderID());) {
        resultSet.next();
        return resultSet.getDouble("total");
    } catch (Exception e) {
        LOGGER.debug(e);
        LOGGER.error(e.getMessage());
    }
    return null;
}
```

Continuous Integration

- Git used for version control.
- Regular commits and pushes were used to mitigate losing work.
- The feature-branch model was used to separate certain functions into different branches.
- Project pushed to a GitHub repository


```
C:\Users\admin\Desktop\ims-demo>git branch -a
* dev
  feature-CustomerDaoTest
  feature-customertest
  feature-items
  feature-itemtest
  feature-orderCalcPrice
  feature-orderDelete
  feature-orderMultItems
  feature-orderUpdate
  feature-orders
  itemtest2
  master
  orderTest
  updated_base
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev
  remotes/origin/feature-CustomerDaoTest
  remotes/origin/feature-customertest
  remotes/origin/feature-items
  remotes/origin/feature-itemtest
  remotes/origin/feature-orderCalcPrice
  remotes/origin/feature-orderDelete
  remotes/origin/feature-orderMultItems
  remotes/origin/feature-orderUpdate
  remotes/origin/feature-orders
  remotes/origin/itemtest2
  remotes/origin/master
  remotes/origin/orderTest
```

Live Demonstration

Testing

- 72.6% coverage
- Main testing problems occurred within the Order tests

calfonso-ims (21 Jan 2021 23:44:43)				
Element	Coverage	ed Instructions	ed Instructions	tal Instructions
calfonso-ims	84.5 %	4,395	806	5,201
src/main/java	72.6 %	2,057	778	2,835
com.qa.ims.persistence.dao	66.2 %	711	363	1,074
com.qa.ims	0.0 %	0	175	175
com.qa.ims.controller	72.7 %	463	174	637
com.qa.ims.persistence.domain	94.6 %	706	40	746
com.qa.ims.utils	78.2 %	93	26	119
com.qa.ims.services	100.0 %	84	0	84
src/test/java	98.8 %	2,338	28	2,366
com.qa.ims.persistence.dao	96.2 %	559	22	581
com.qa.ims.controller	99.0 %	609	6	615
com.qa.ims.persistence.domain	100.0 %	930	0	930
com.qa.ims.services	100.0 %	240	0	240

Project Sprint Review

MoSCoW Approach

Must Have	Should have	Could have	Won't have this time
Basic CRUD functionality ✔	Methods to add/remove multiple items to an order ✔	Method to capitalise customer names ✘	Customer address ✘
Item name ✔	Minimum 60% test coverage ✔	90%+ test coverage ✘	Customer email ✘
Customer name ✔			Automatic Item stock count ✘
Total order price ✔			Total revenue from each customer ✘

Conclusion - Sprint Retrospective

What Went Well:	What could be improved:
Completed a working application with all MVP requirements and additional functionality.	Focus an equal amount of time on testing as on development
Completing some difficult methods independently.	Don't create a Jira board using the QA email
Developed understanding of and used design patterns.	Cleaner code, especially for Order

Thank you for listening

Questions?