Project Presentation

INVENTORY MANAGEMENT SYSTEM (IMS)

Simon White

Introducing myself:

- Simon White – Belfast, 22 years old
- Background in coding through Business IT degree
- Year of professional experience as Software Engineer

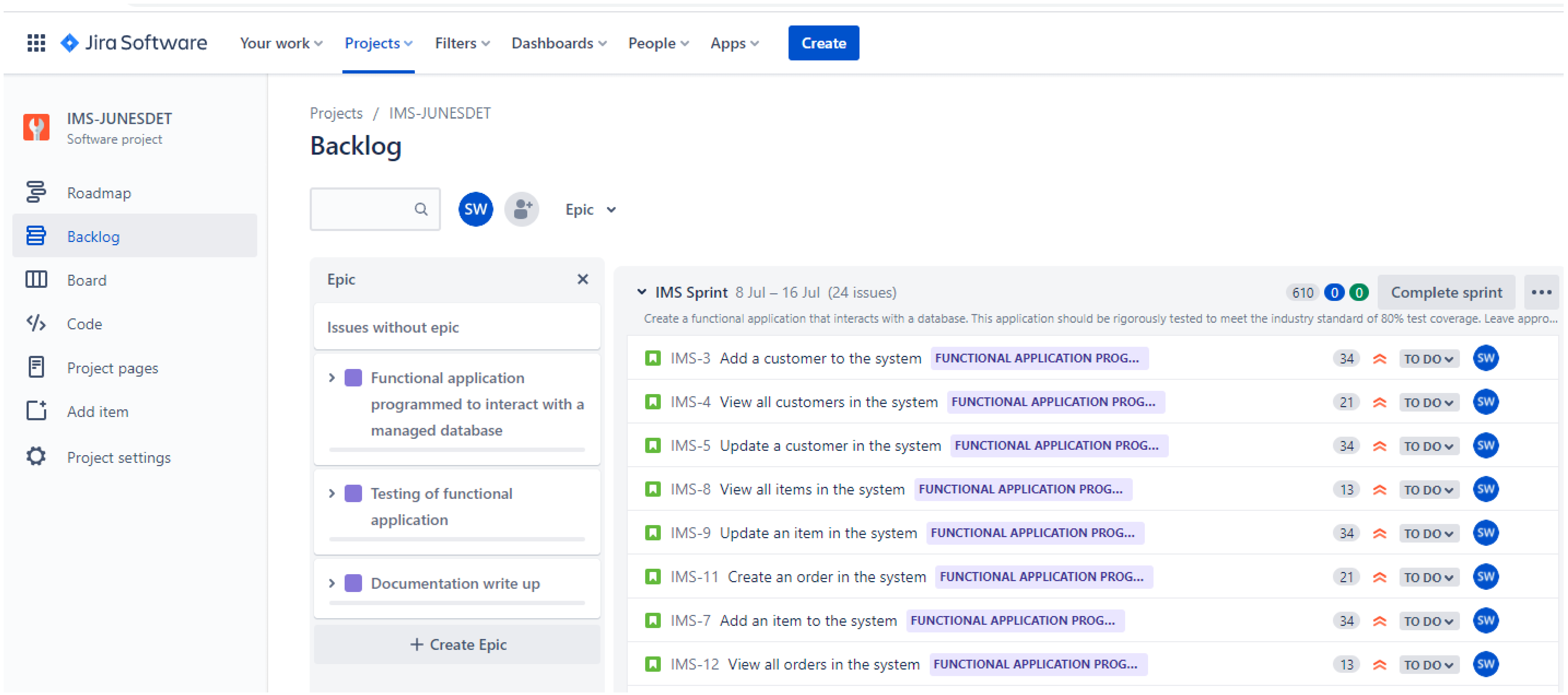
Inventory Management System (IMS)

- Developed an IMS for American Golf
- The purpose of the IMS is to identify every inventory entity and be able to interact with their information - CRUD

Approach: AGILE

- Requirements: Project specification
- Analysis: JIRA
- Design: ERD, UML
- Coding: Java
- Testing: Maven
- Operations: Provide jar file to run the finalised application

Analysis: Jira



Jira Software · Your work · Projects · Filters · Dashboards · People · Apps · Create

IMS-JUNESDET
Software project

Roadmap · Backlog · Board · Code · Project pages · Add item · Project settings

Projects / IMS-JUNESDET

Backlog

SW · Epic

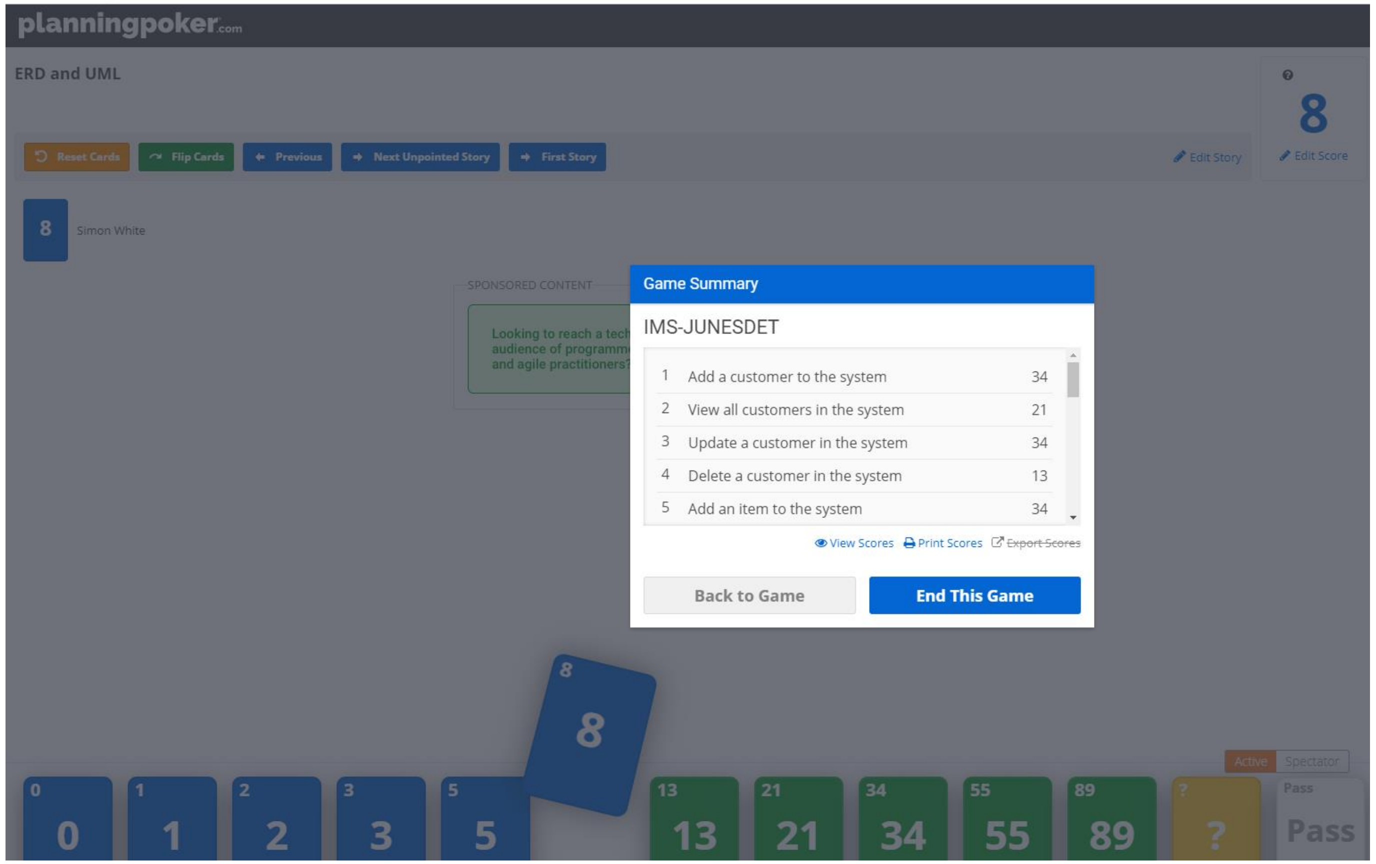
Epic
- Issues without epic
- Functional application programmed to interact with a managed database
- Testing of functional application
- Documentation write up
- + Create Epic

IMS Sprint 8 Jul – 16 Jul (24 issues) 610 0 0 Complete sprint

Create a functional application that interacts with a database. This application should be rigorously tested to meet the industry standard of 80% test coverage. Leave appro...

IMS-3	Add a customer to the system	FUNCTIONAL APPLICATION PROG...	34		TO DO	SW
IMS-4	View all customers in the system	FUNCTIONAL APPLICATION PROG...	21		TO DO	SW
IMS-5	Update a customer in the system	FUNCTIONAL APPLICATION PROG...	34		TO DO	SW
IMS-8	View all items in the system	FUNCTIONAL APPLICATION PROG...	13		TO DO	SW
IMS-9	Update an item in the system	FUNCTIONAL APPLICATION PROG...	34		TO DO	SW
IMS-11	Create an order in the system	FUNCTIONAL APPLICATION PROG...	21		TO DO	SW
IMS-7	Add an item to the system	FUNCTIONAL APPLICATION PROG...	34		TO DO	SW
IMS-12	View all orders in the system	FUNCTIONAL APPLICATION PROG...	13		TO DO	SW

- Planning Poker used to get story points (estimations)
- Priority was given to all issues that are part of MVP (Minimal viable product). These were set at highest priority in order to achieve MVP.
- 3 epics for functionality, testing and documentation

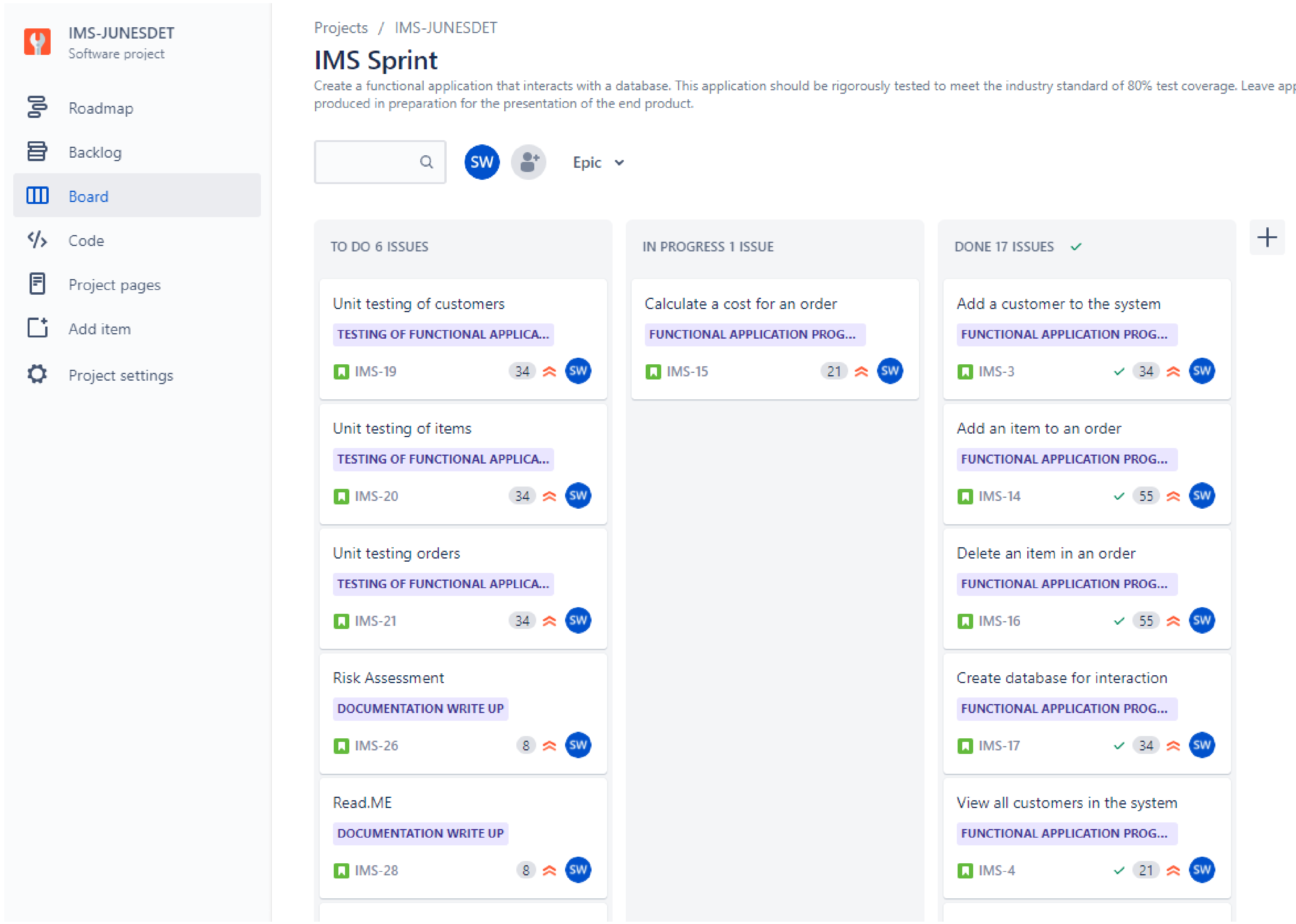


Game Summary

IMS-JUNESDET

1	Add a customer to the system	34
2	View all customers in the system	21
3	Update a customer in the system	34
4	Delete a customer in the system	13
5	Add an item to the system	34

IMS Sprint: Kanban board

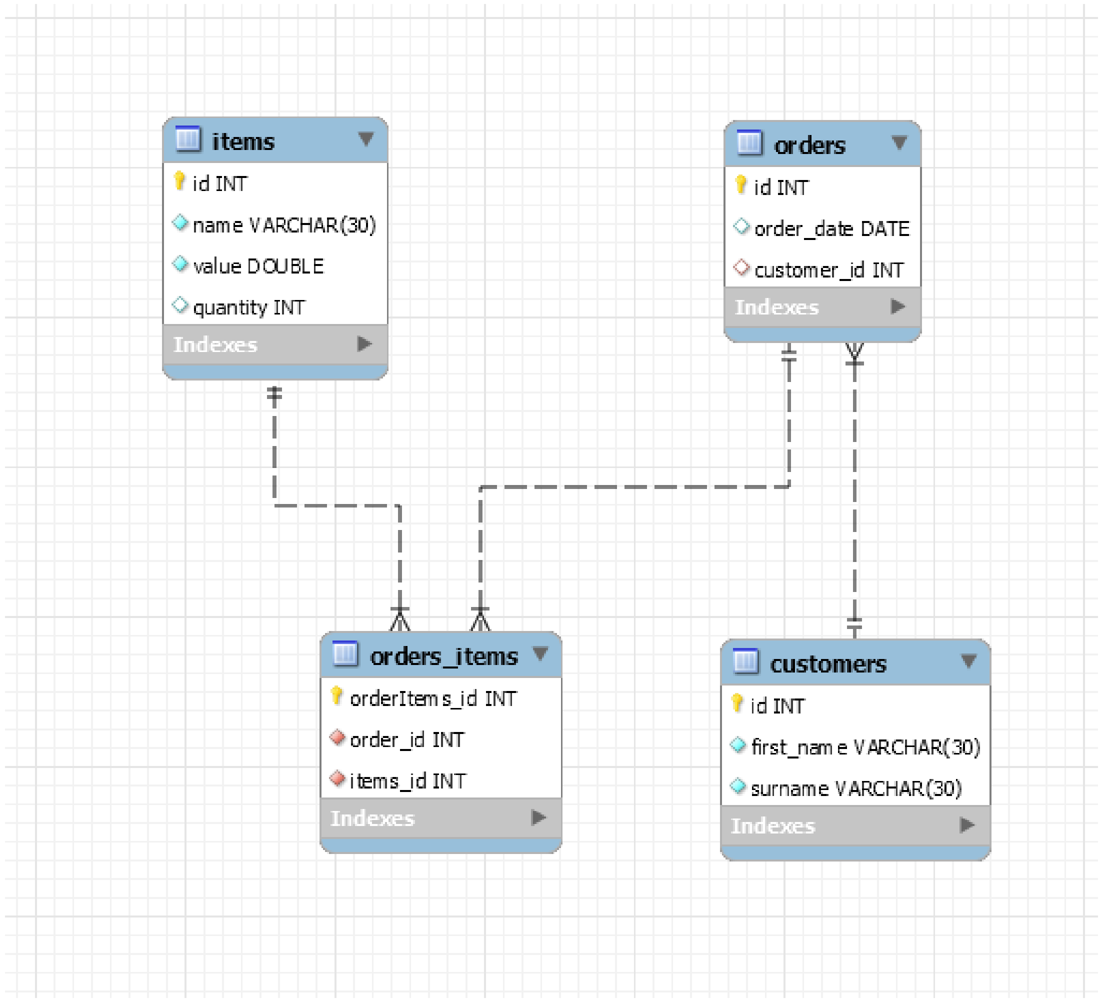


IMS Sprint

Create a functional application that interacts with a database. This application should be rigorously tested to meet the industry standard of 80% test coverage. Leave appropriate time for documentation to be efficiently produced in preparation for the presentation of the end product.

TO DO 6 ISSUES	IN PROGRESS 1 ISSUE	DONE 17 ISSUES ✓
Unit testing of customers — TESTING OF FUNCTIONAL APPLICA... — IMS-19 — 34 — SW	Calculate a cost for an order — FUNCTIONAL APPLICATION PROG... — IMS-15 — 21 — SW	Add a customer to the system — FUNCTIONAL APPLICATION PROG... — IMS-3 — ✓ 34 — SW
Unit testing of items — TESTING OF FUNCTIONAL APPLICA... — IMS-20 — 34 — SW		Add an item to an order — FUNCTIONAL APPLICATION PROG... — IMS-14 — ✓ 55 — SW
Unit testing orders — TESTING OF FUNCTIONAL APPLICA... — IMS-21 — 34 — SW		Delete an item in an order — FUNCTIONAL APPLICATION PROG... — IMS-16 — ✓ 55 — SW
Risk Assessment — DOCUMENTATION WRITE UP — IMS-26 — 8 — SW		Create database for interaction — FUNCTIONAL APPLICATION PROG... — IMS-17 — ✓ 34 — SW
Read.ME — DOCUMENTATION WRITE UP — IMS-28 — 8 — SW		View all customers in the system — FUNCTIONAL APPLICATION PROG... — IMS-4 — ✓ 21 — SW

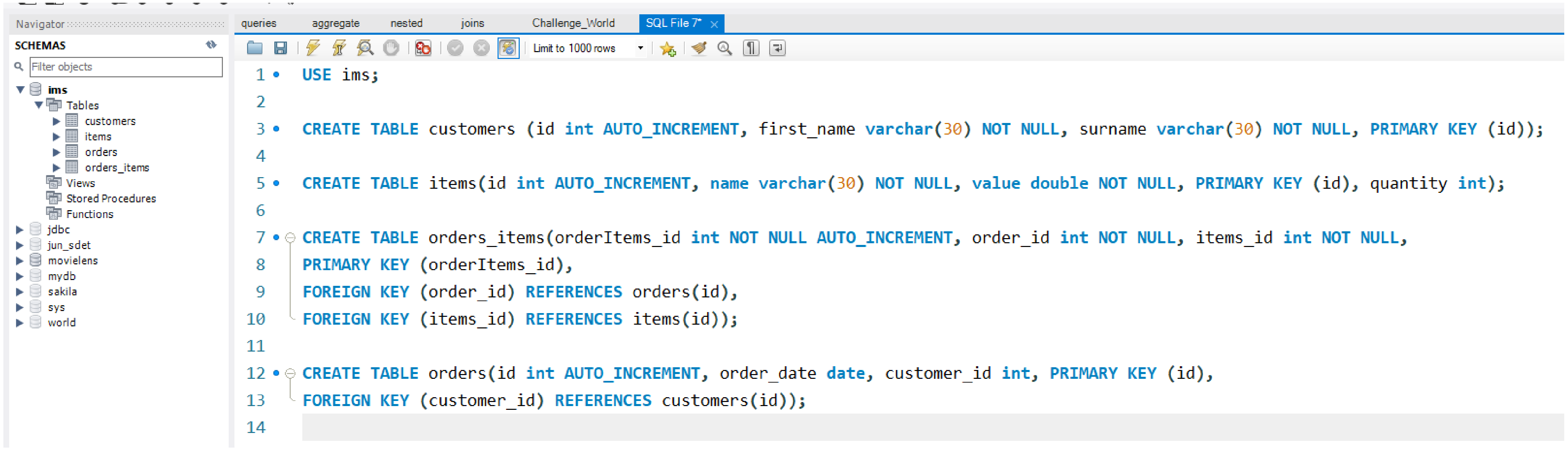
Design: ERD and Database Implementation



Entities: items, orders, order items, customers

Foreign Keys: customer, order, item

Database Implementation

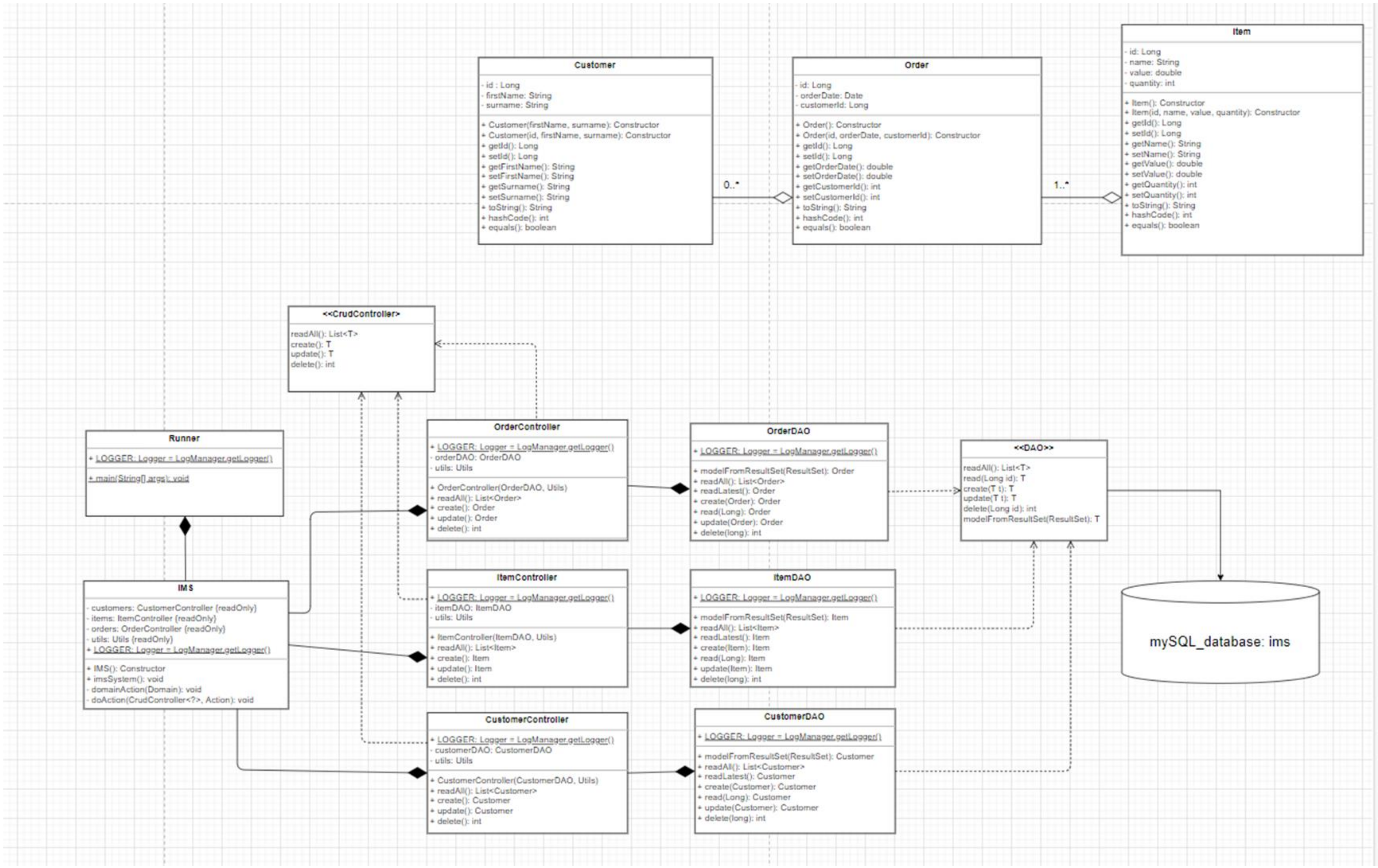


```sql
USE ims;

CREATE TABLE customers (id int AUTO_INCREMENT, first_name varchar(30) NOT NULL, surname varchar(30) NOT NULL, PRIMARY KEY (id));

CREATE TABLE items(id int AUTO_INCREMENT, name varchar(30) NOT NULL, value double NOT NULL, PRIMARY KEY (id), quantity int);

CREATE TABLE orders_items(orderItems_id int NOT NULL AUTO_INCREMENT, order_id int NOT NULL, items_id int NOT NULL,
PRIMARY KEY (orderItems_id),
FOREIGN KEY (order_id) REFERENCES orders(id),
FOREIGN KEY (items_id) REFERENCES items(id));

CREATE TABLE orders(id int AUTO_INCREMENT, order_date date, customer_id int, PRIMARY KEY (id),
FOREIGN KEY (customer_id) REFERENCES customers(id));

```

Design: UML

- Aggregation
Classes that can live without each other but still have a relationship

- Composition
Two classes highly dependent on each other



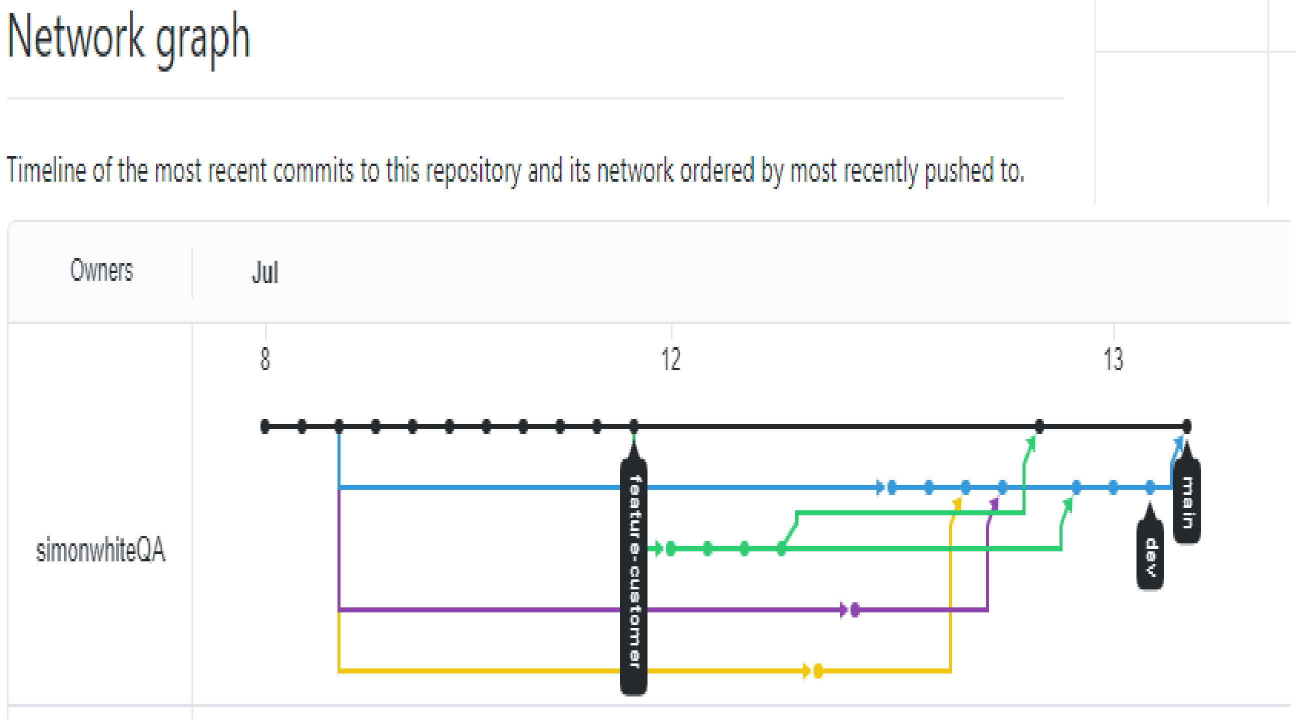
Consultant Journey: Technologies

- **Version Control System**: Git
- **Source Code Management:** GitHub
- **Kanban Board**: Jira
- **Database Management System**: MySQL Server 5.7+
- **Back-End Programming Language**: Java
- **Build Tool:** Maven
- **Unit Testing:** JUnit

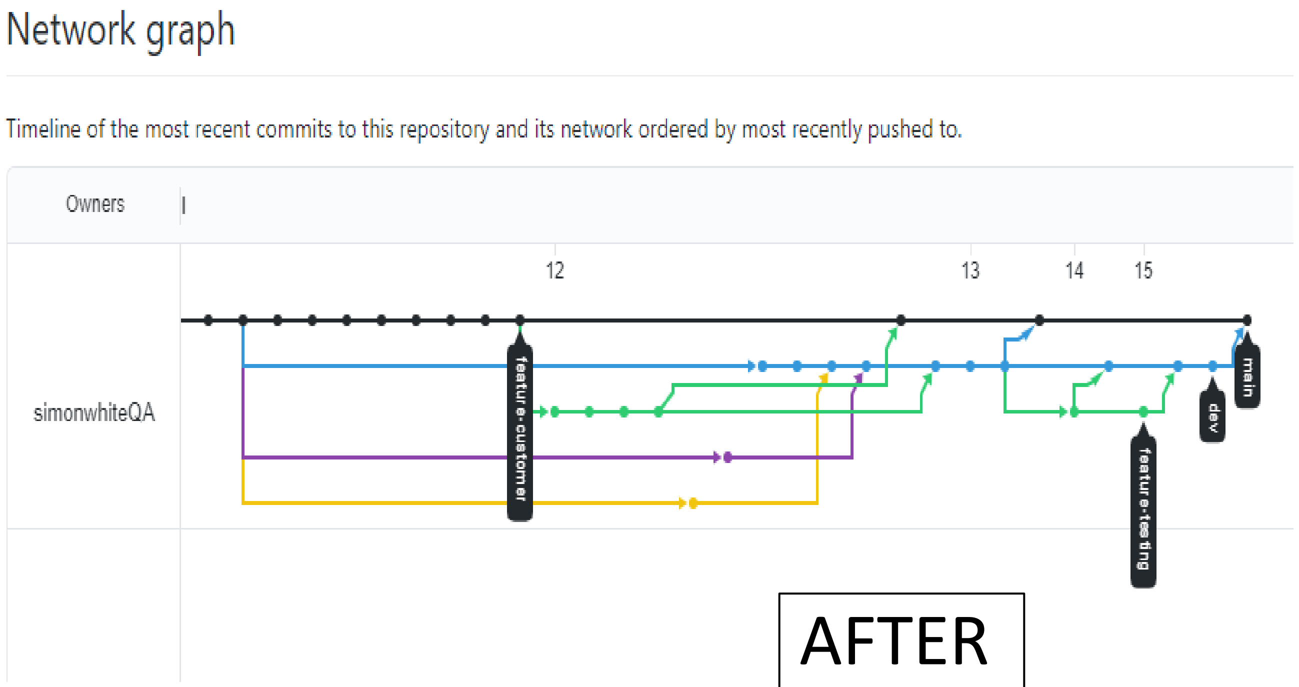
Continuous Integration

- Feature branch model (GIT)

BEFORE


Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

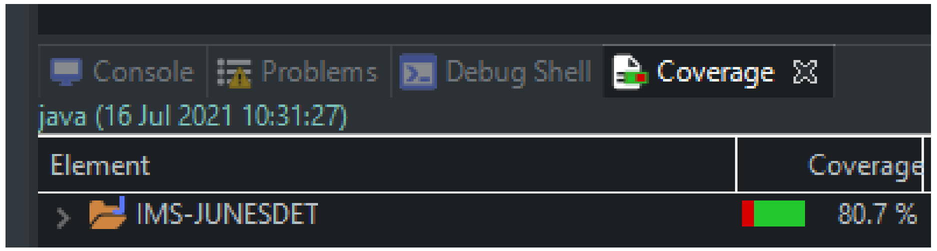

Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

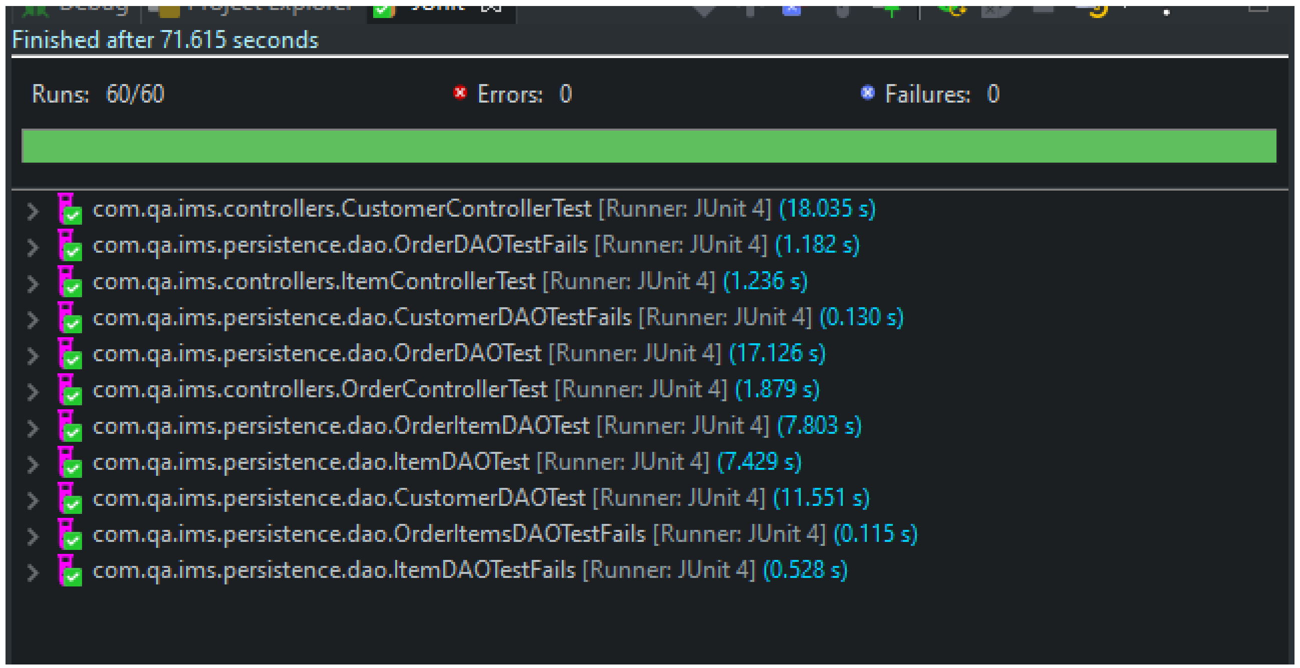
AFTER

Testing

- Unit testing: Unit testing is a testing approach that targets the very fundamental building blocks of an application, the idea is to prove that each 'unit' of the application is functioning as expected.
- Industry standard of 80% for testing coverage.
- JUnit: The unit testing dependency for maven to test java application
- Using Maven: Maven is a build automation tool



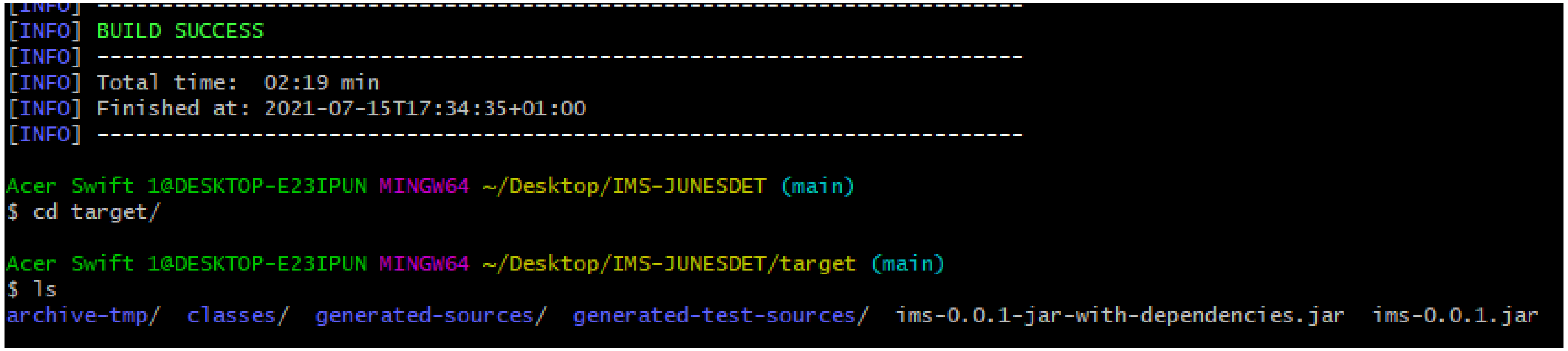
```
java (16 Jul 2021 10:31:27)
Element                         Coverage
> IMS-JUNESDET                  80.7 %
```



```
Finished after 71.615 seconds

Runs: 60/60          Errors: 0          Failures: 0

> com.qa.ims.controllers.CustomerControllerTest [Runner: JUnit 4] (18.035 s)
> com.qa.ims.persistence.dao.OrderDAOTestFails [Runner: JUnit 4] (1.182 s)
> com.qa.ims.controllers.ItemControllerTest [Runner: JUnit 4] (1.236 s)
> com.qa.ims.persistence.dao.CustomerDAOTestFails [Runner: JUnit 4] (0.130 s)
> com.qa.ims.persistence.dao.OrderDAOTest [Runner: JUnit 4] (17.126 s)
> com.qa.ims.controllers.OrderControllerTest [Runner: JUnit 4] (1.879 s)
> com.qa.ims.persistence.dao.OrderItemDAOTest [Runner: JUnit 4] (7.803 s)
> com.qa.ims.persistence.dao.ItemDAOTest [Runner: JUnit 4] (7.429 s)
> com.qa.ims.persistence.dao.CustomerDAOTest [Runner: JUnit 4] (11.551 s)
> com.qa.ims.persistence.dao.OrderItemsDAOTestFails [Runner: JUnit 4] (0.115 s)
> com.qa.ims.persistence.dao.ItemDAOTestFails [Runner: JUnit 4] (0.528 s)
```

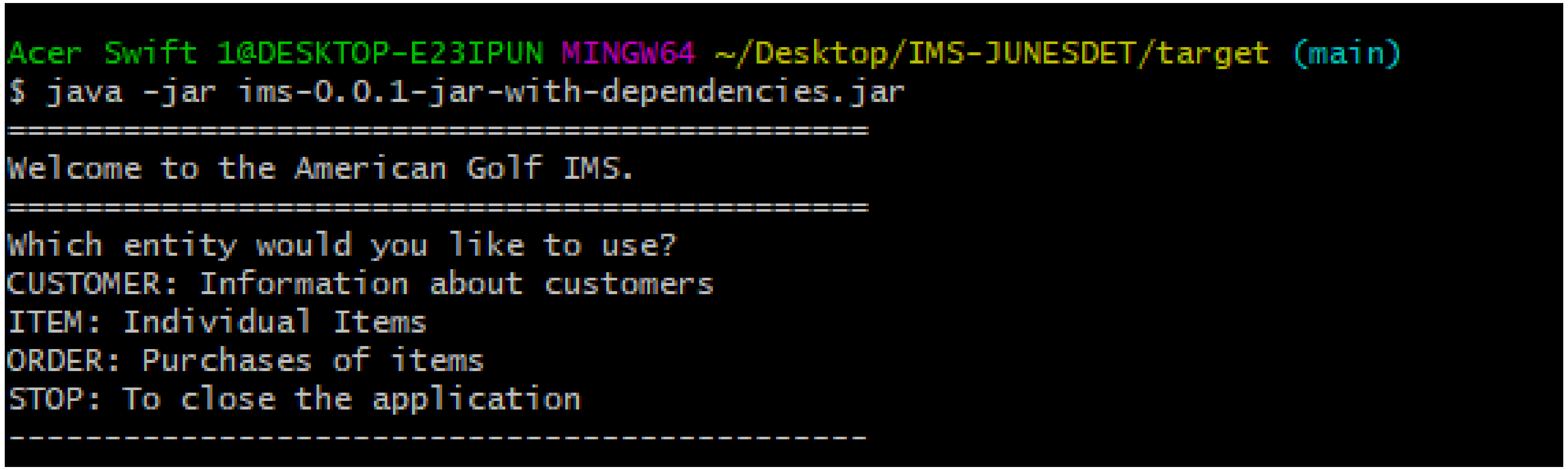
Demonstration

Build jar file



```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:19 min
[INFO] Finished at: 2021-07-15T17:34:35+01:00
[INFO] ------------------------------------------------------------------------

Acer Swift 1@DESKTOP-E23IPUN MINGW64 ~/Desktop/IMS-JUNESDET (main)
$ cd target/

Acer Swift 1@DESKTOP-E23IPUN MINGW64 ~/Desktop/IMS-JUNESDET/target (main)
$ ls
archive-tmp/  classes/  generated-sources/  generated-test-sources/  ims-0.0.1-jar-with-dependencies.jar  ims-0.0.1.jar
```

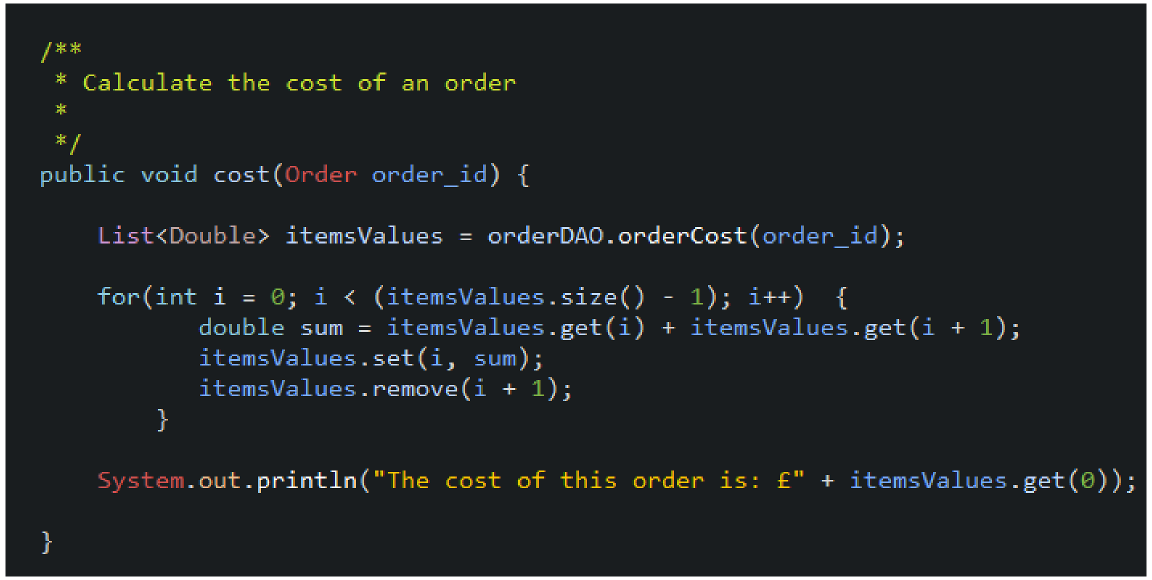
IMS system live for use on command line



```
Acer Swift 1@DESKTOP-E23IPUN MINGW64 ~/Desktop/IMS-JUNESDET/target (main)
$ java -jar ims-0.0.1-jar-with-dependencies.jar
=============================================
Welcome to the American Golf IMS.
=============================================
Which entity would you like to use?
CUSTOMER: Information about customers
ITEM: Individual Items
ORDER: Purchases of items
STOP: To close the application
---------------------------------------------
```

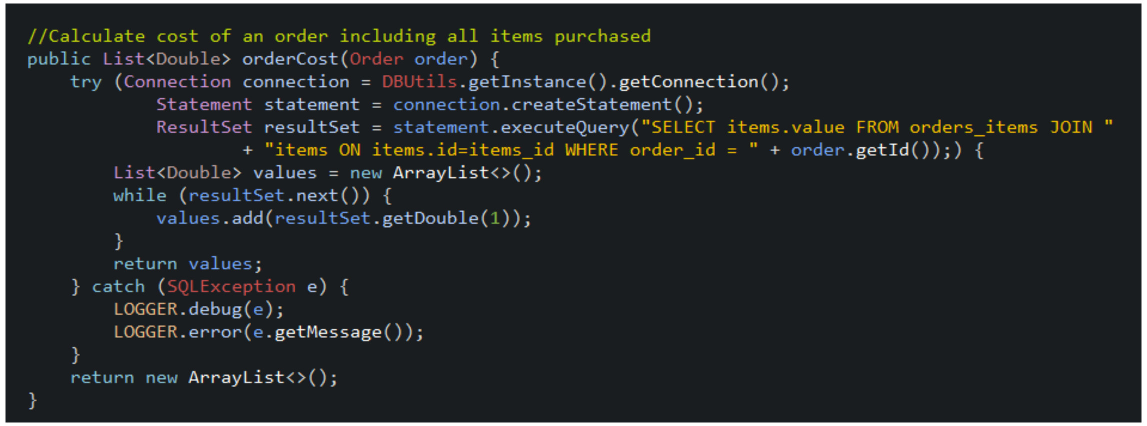
Calculate cost/Add items to order

Order Controller



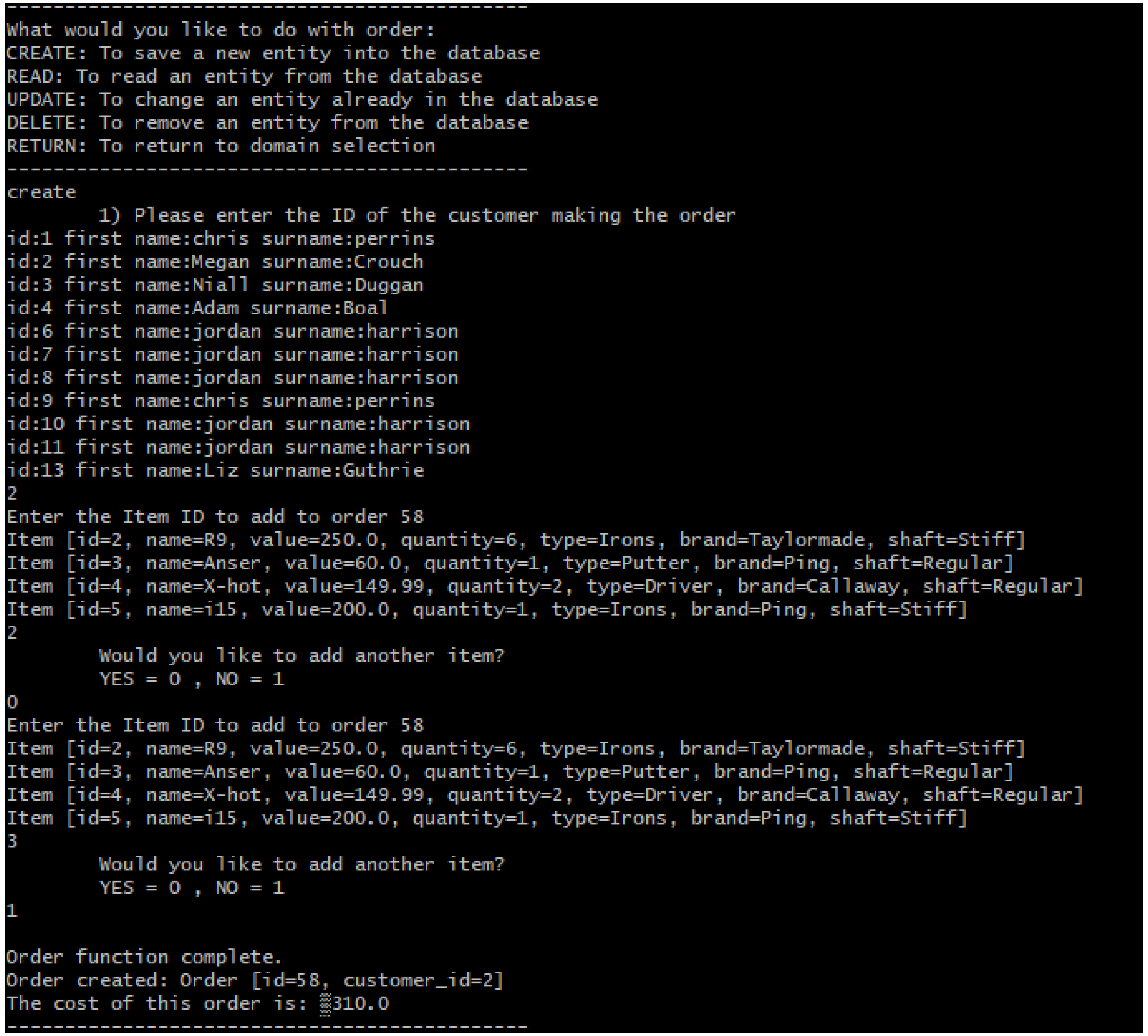
```
/**
 * Calculate the cost of an order
 *
 */
public void cost(Order order_id) {

    List<Double> itemsValues = orderDAO.orderCost(order_id);

    for(int i = 0; i < (itemsValues.size() - 1); i++)  {
           double sum = itemsValues.get(i) + itemsValues.get(i + 1);
           itemsValues.set(i, sum);
           itemsValues.remove(i + 1);
        }

    System.out.println("The cost of this order is: £" + itemsValues.get(0));

}
```

Order DAO



```
//Calculate cost of an order including all items purchased
public List<Double> orderCost(Order order) {
    try (Connection connection = DBUtils.getInstance().getConnection();
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT items.value FROM orders_items JOIN "
                    + "items ON items.id=items_id WHERE order_id = " + order.getId());) {
        List<Double> values = new ArrayList<>();
        while (resultSet.next()) {
            values.add(resultSet.getDouble(1));
        }
        return values;
    } catch (SQLException e) {
        LOGGER.debug(e);
        LOGGER.error(e.getMessage());
    }
    return new ArrayList<>();
}
```

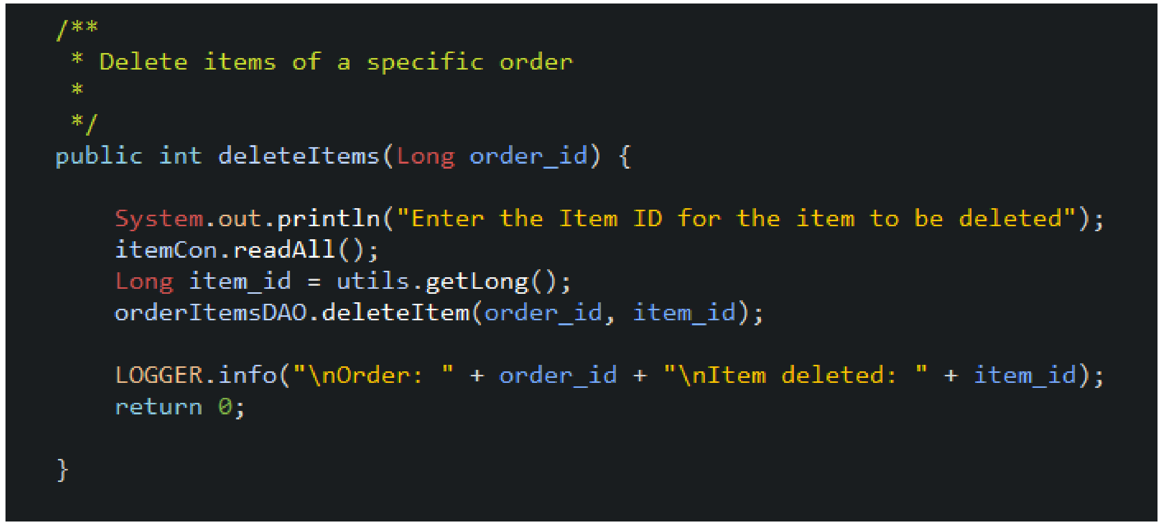
Method run on jar



```
---------------------------------------------
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
---------------------------------------------
create
        1) Please enter the ID of the customer making the order
id:1 first name:chris surname:perrins
id:2 first name:Megan surname:Crouch
id:3 first name:Niall surname:Duggan
id:4 first name:Adam surname:Boal
id:6 first name:jordan surname:harrison
id:7 first name:jordan surname:harrison
id:8 first name:jordan surname:harrison
id:9 first name:chris surname:perrins
id:10 first name:jordan surname:harrison
id:11 first name:jordan surname:harrison
id:13 first name:Liz surname:Guthrie
2
Enter the Item ID to add to order 58
Item [id=2, name=R9, value=250.0, quantity=6, type=Irons, brand=Taylormade, shaft=Stiff]
Item [id=3, name=Anser, value=60.0, quantity=1, type=Putter, brand=Ping, shaft=Regular]
Item [id=4, name=X-hot, value=149.99, quantity=2, type=Driver, brand=Callaway, shaft=Regular]
Item [id=5, name=i15, value=200.0, quantity=1, type=Irons, brand=Ping, shaft=Stiff]
2
        Would you like to add another item?
        YES = 0 , NO = 1
0
Enter the Item ID to add to order 58
Item [id=2, name=R9, value=250.0, quantity=6, type=Irons, brand=Taylormade, shaft=Stiff]
Item [id=3, name=Anser, value=60.0, quantity=1, type=Putter, brand=Ping, shaft=Regular]
Item [id=4, name=X-hot, value=149.99, quantity=2, type=Driver, brand=Callaway, shaft=Regular]
Item [id=5, name=i15, value=200.0, quantity=1, type=Irons, brand=Ping, shaft=Stiff]
3
        Would you like to add another item?
        YES = 0 , NO = 1
1

Order function complete.
Order created: Order [id=58, customer_id=2]
The cost of this order is: ▒310.0
---------------------------------------------
```

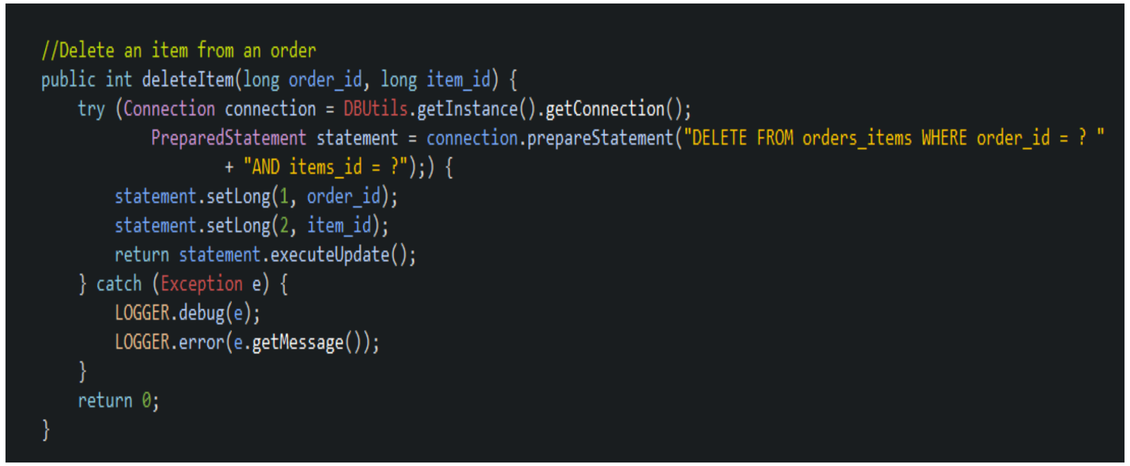
Delete an item from an order

Order Controller



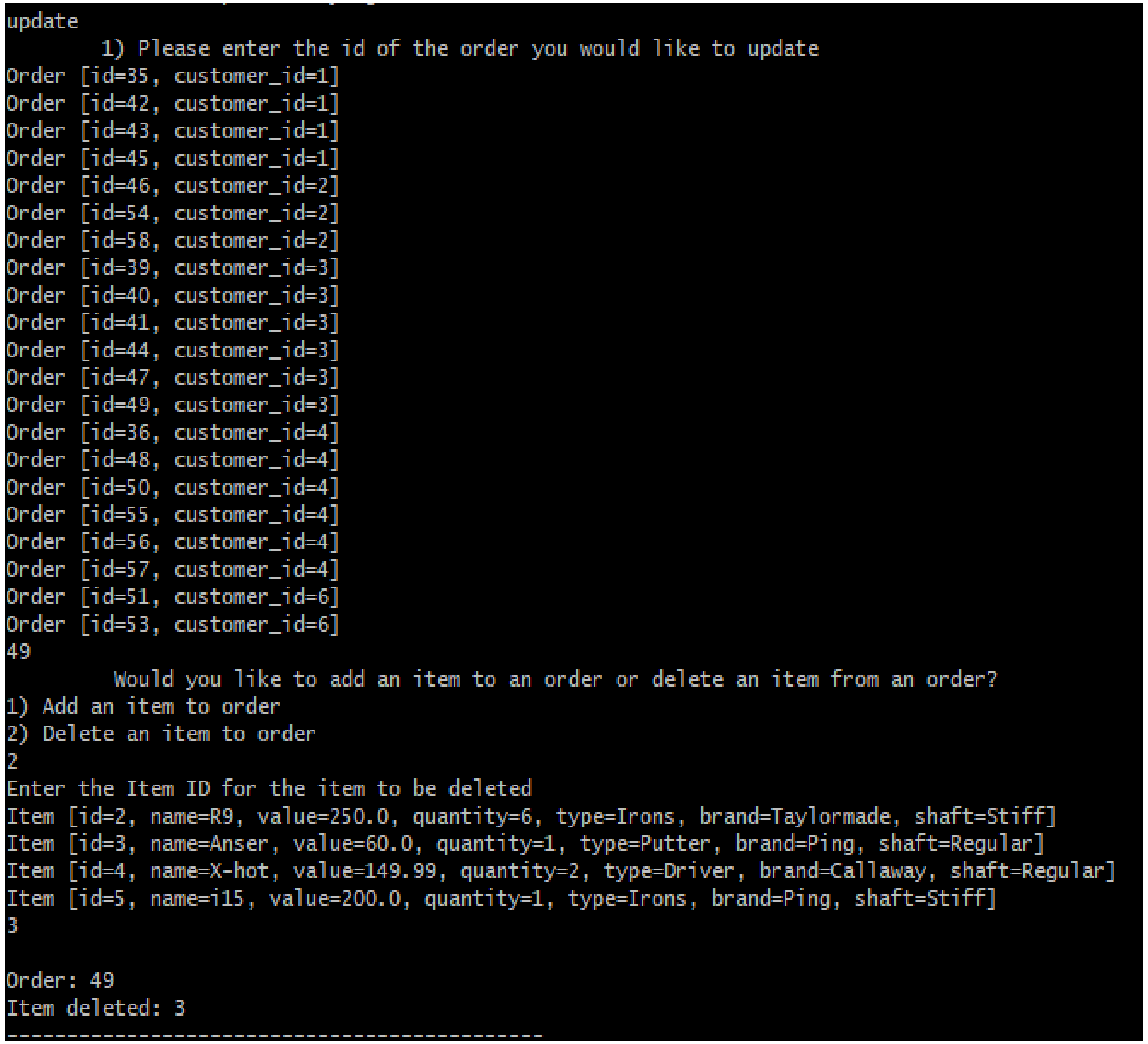
```
/**
 * Delete items of a specific order
 *
 */
public int deleteItems(Long order_id) {

    System.out.println("Enter the Item ID for the item to be deleted");
    itemCon.readAll();
    Long item_id = utils.getLong();
    orderItemsDAO.deleteItem(order_id, item_id);

    LOGGER.info("\nOrder: " + order_id + "\nItem deleted: " + item_id);
    return 0;

}
```

Order items DAO



```
//Delete an item from an order
public int deleteItem(long order_id, long item_id) {
    try (Connection connection = DBUtils.getInstance().getConnection();
            PreparedStatement statement = connection.prepareStatement("DELETE FROM orders_items WHERE order_id = ? "
                    + "AND items_id = ?");) {
        statement.setLong(1, order_id);
        statement.setLong(2, item_id);
        return statement.executeUpdate();
    } catch (Exception e) {
        LOGGER.debug(e);
        LOGGER.error(e.getMessage());
    }
    return 0;
}
```

Method run on jar

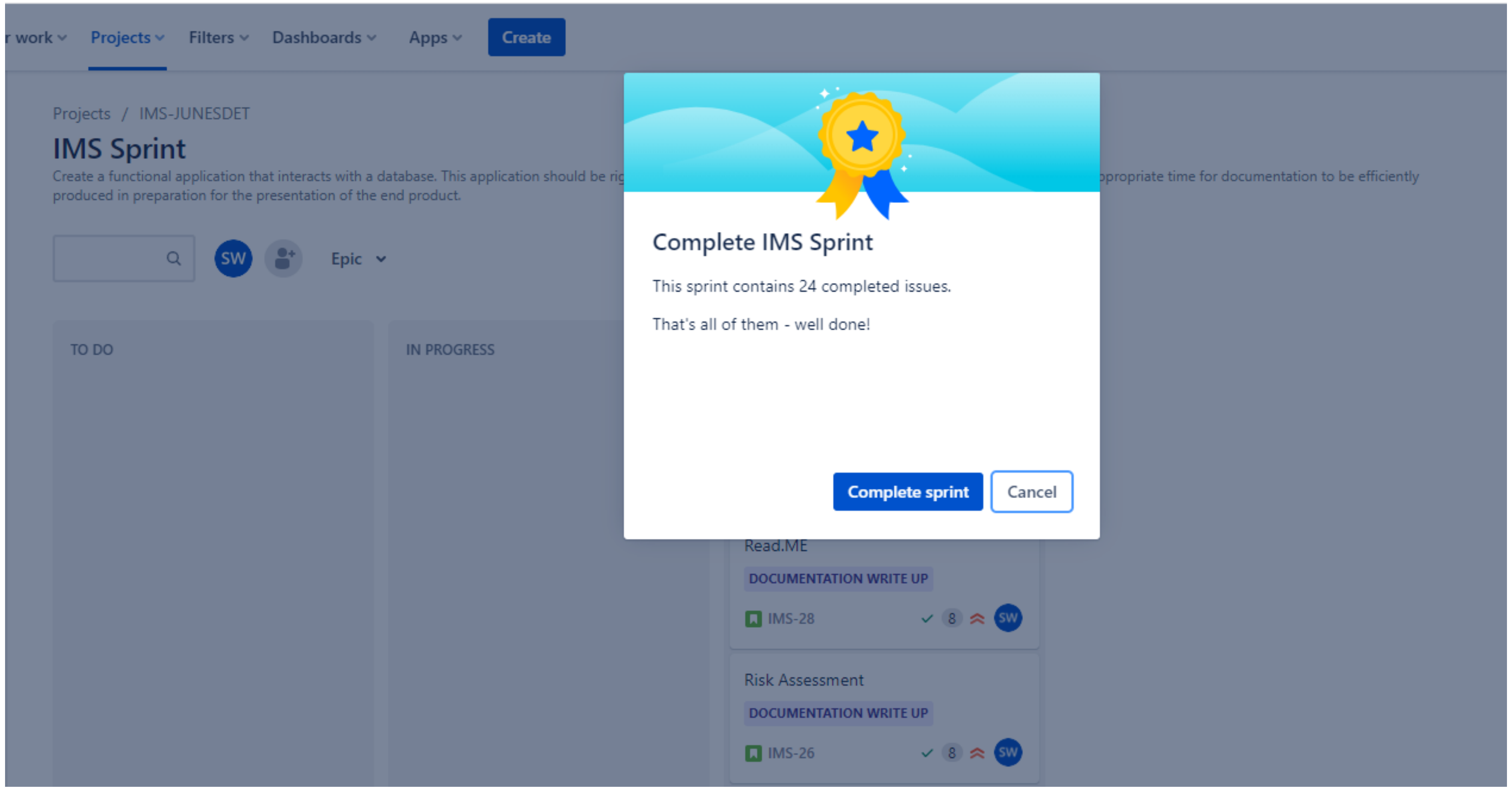


```
update
        1) Please enter the id of the order you would like to update
Order [id=35, customer_id=1]
Order [id=42, customer_id=1]
Order [id=43, customer_id=1]
Order [id=45, customer_id=1]
Order [id=46, customer_id=2]
Order [id=54, customer_id=2]
Order [id=58, customer_id=2]
Order [id=39, customer_id=3]
Order [id=40, customer_id=3]
Order [id=41, customer_id=3]
Order [id=44, customer_id=3]
Order [id=47, customer_id=3]
Order [id=49, customer_id=3]
Order [id=36, customer_id=4]
Order [id=48, customer_id=4]
Order [id=50, customer_id=4]
Order [id=55, customer_id=4]
Order [id=56, customer_id=4]
Order [id=57, customer_id=4]
Order [id=51, customer_id=6]
Order [id=53, customer_id=6]
49
         Would you like to add an item to an order or delete an item from an order?
1) Add an item to order
2) Delete an item to order
2
Enter the Item ID for the item to be deleted
Item [id=2, name=R9, value=250.0, quantity=6, type=Irons, brand=Taylormade, shaft=Stiff]
Item [id=3, name=Anser, value=60.0, quantity=1, type=Putter, brand=Ping, shaft=Regular]
Item [id=4, name=X-hot, value=149.99, quantity=2, type=Driver, brand=Callaway, shaft=Regular]
Item [id=5, name=i15, value=200.0, quantity=1, type=Irons, brand=Ping, shaft=Stiff]
3

Order: 49
Item deleted: 3
---------------------------------------------
```

Sprint review

Completed (MVP)

- Add a **customer** to the system
- View all **customers** in the system
- Update a **customer** in the system
- Delete a **customer** in the system
- Add an **item** to the system
- View all **items** in the system
- Update an **item** in the system
- Delete an **item** in the system
- Create an **order** in the system
- View all **orders** in the system
- Delete an **order** in the system
- Add an **item** to an **order**
- Calculate a cost for an **order**
- Delete an **item** in an **order**



IMS Sprint

Create a functional application that interacts with a database. This application should be rig ppropriate time for documentation to be efficiently produced in preparation for the presentation of the end product.

Complete IMS Sprint

This sprint contains 24 completed issues.

That's all of them - well done!

Sprint retrospective

Positives

- Improved understanding of the fundamentals of competing an project using an agile approach
- Improved understanding of Java and its corresponding principles for good practise
- Interaction with a database ran smoothly and my understanding of the usage of an intermediary table for many to many relationships has improved.
- Testing training was utilised to ensure industry standard coverage was met.
- Use of Jira for planning and monitoring project progress was utilised effectively

Positives

- Carelessness with continuous integration meant that a push to main branch did not follow the feature branch model.
- Project timing was continuously a concern as story point estimations may not have been accurate into the complexity of some requirements such as add an item to a order.

Conclusion

- All requirements were met
- Java application runs as expected
- Testing coverage meets industry standard and all tests pass
- Understanding of the project life cycle from requirements to implementation to finalising the product has vastly improved
- Mistakes that I have encountered during this project will be influential in how I approach projects in the future
- In particular, continuous integration has been identified as a weak topic and in the future I was follow the feature branch model more accurately