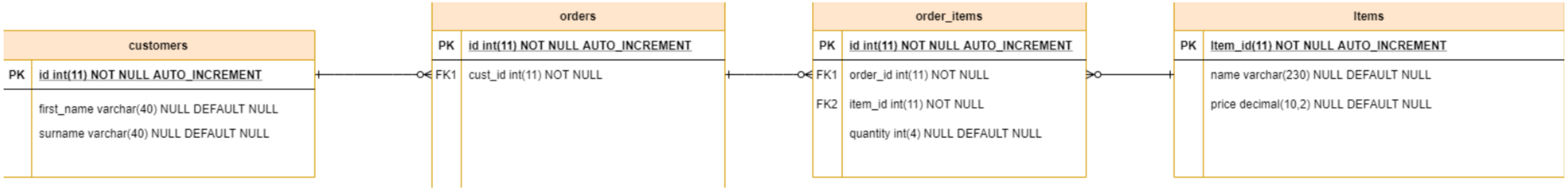
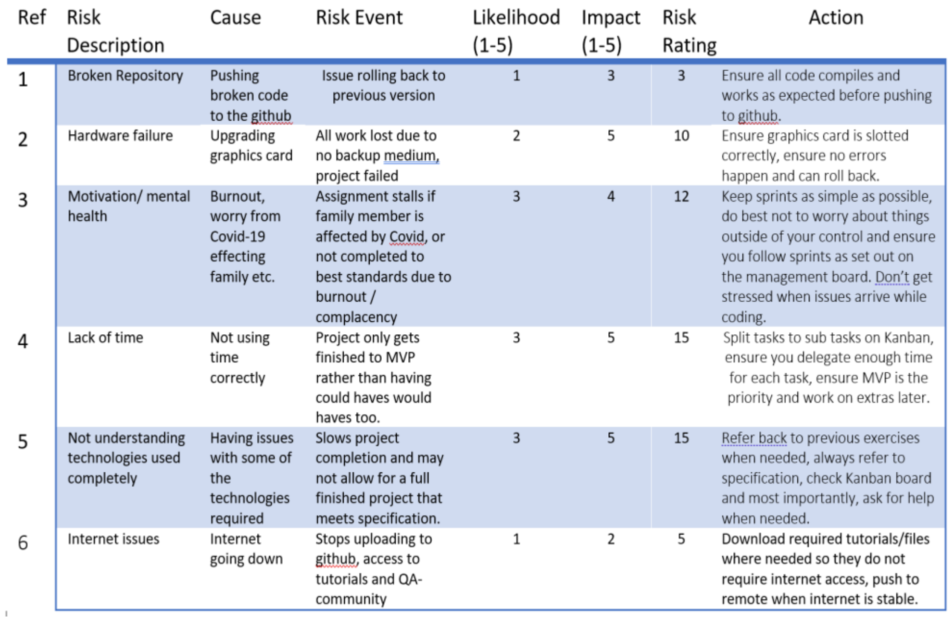
IMS Project

RYAN GLOSSOP

Design


customers
PK id int(11) NOT NULL AUTO_INCREMENT
first_name varchar(40) NULL DEFAULT NULL
surname varchar(40) NULL DEFAULT NULL
orders
PK id int(11) NOT NULL AUTO_INCREMENT
FK1 cust_id int(11) NOT NULL
order_items
PK id int(11) NOT NULL AUTO_INCREMENT
FK1 order_id int(11) NOT NULL
FK2 item_id int(11) NOT NULL
quantity int(4) NULL DEFAULT NULL
Items
PK Item_id(11) NOT NULL AUTO_INCREMENT
name varchar(230) NULL DEFAULT NULL
price decimal(10,2) NULL DEFAULT NULL



Ref	Risk Description	Cause	Risk Event	Likelihood (1-5)	Impact (1-5)	Risk Rating	Action
1	Broken Repository	Pushing broken code to the github	Issue rolling back to previous version	1	3	3	Ensure all code compiles and works as expected before pushing to github.
2	Hardware failure	Upgrading graphics card	All work lost due to no backup medium, project failed	2	5	10	Ensure graphics card is slotted correctly, ensure no errors happen and can roll back.
3	Motivation/ mental health	Burnout, worry from Covid-19 effecting family etc.	Assignment stalls if family member is affected by Covid, or not completed to best standards due to burnout / complacency	3	4	12	Keep sprints as simple as possible, do best not to worry about things outside of your control and ensure you follow sprints as set out on the management board. Don't get stressed when issues arrive while coding.
4	Lack of time	Not using time correctly	Project only gets finished to MVP rather than having could haves would haves too.	3	5	15	Split tasks to sub tasks on Kanban, ensure you delegate enough time for each task, ensure MVP is the priority and work on extras later.
5	Not understanding technologies used completely	Having issues with some of the technologies required	Slows project completion and may not allow for a full finished project that meets specification.	3	5	15	Refer back to previous exercises when needed, always refer to specification, check Kanban board and most importantly, ask for help when needed.
6	Internet issues	Internet going down	Stops uploading to github, access to tutorials and QA-community	1	2	5	Download required tutorials/files where needed so they do not require internet access, push to remote when internet is stable.

Consultant Journey

Version Control: Git

Source Code Management: GitHub

Kanban Board: Jira

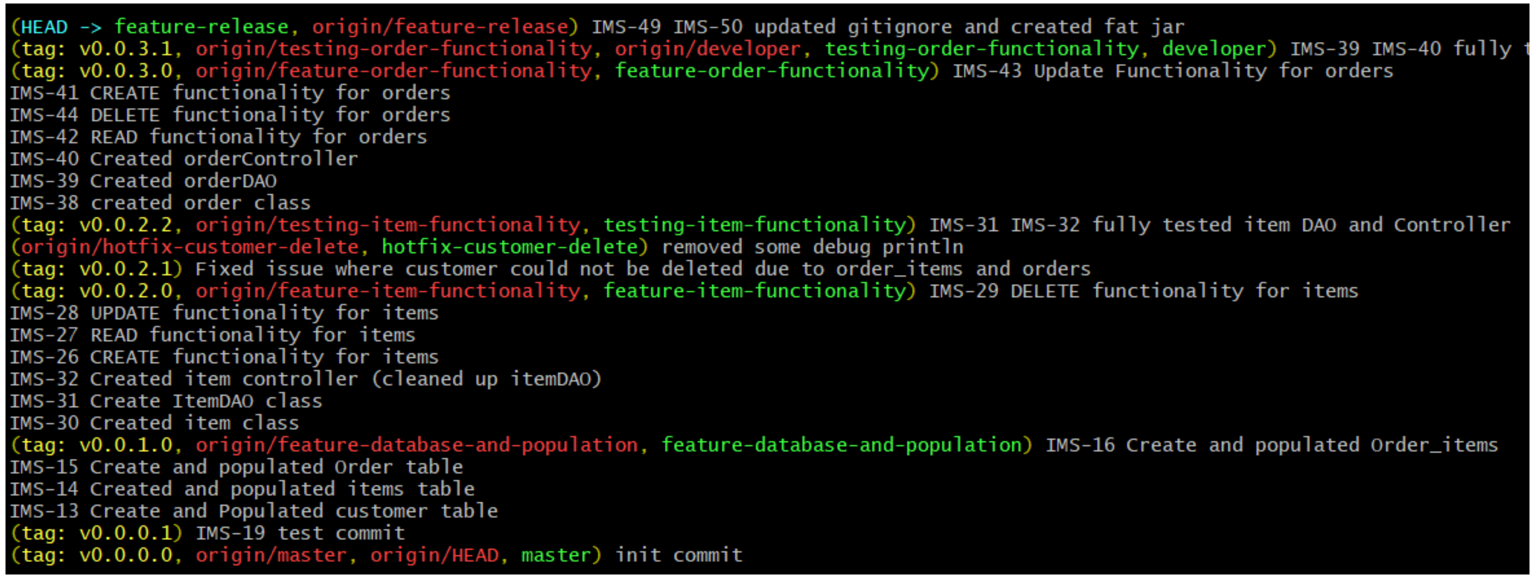
Database: MySQL

Programming Language: Java

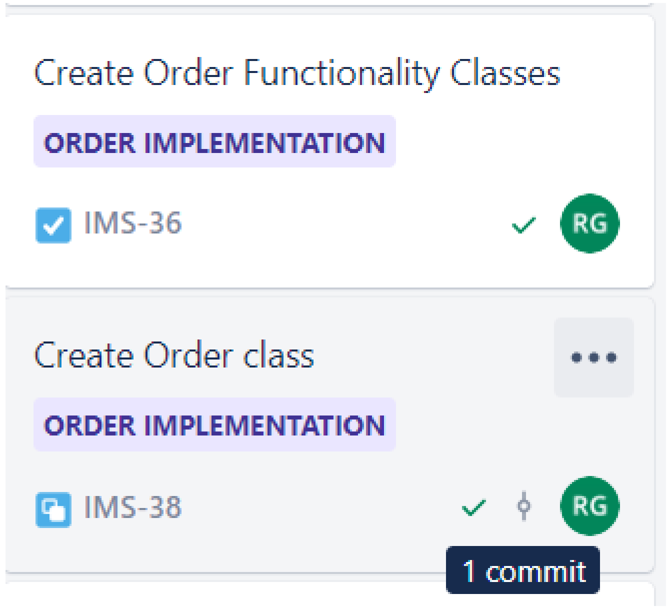
Build Tool: Maven

Unit Testing: Junit, Mockito

Continuous Integration



```
(HEAD -> feature-release, origin/feature-release) IMS-49 IMS-50 updated gitignore and created fat jar
(tag: v0.0.3.1, origin/testing-order-functionality, origin/developer, testing-order-functionality, developer) IMS-39 IMS-40 fully t
(tag: v0.0.3.0, origin/feature-order-functionality, feature-order-functionality) IMS-43 Update Functionality for orders
IMS-41 CREATE functionality for orders
IMS-44 DELETE functionality for orders
IMS-42 READ functionality for orders
IMS-40 Created orderController
IMS-39 Created orderDAO
IMS-38 created order class
(tag: v0.0.2.2, origin/testing-item-functionality, testing-item-functionality) IMS-31 IMS-32 fully tested item DAO and Controller
(origin/hotfix-customer-delete, hotfix-customer-delete) removed some debug println
(tag: v0.0.2.1) Fixed issue where customer could not be deleted due to order_items and orders
(tag: v0.0.2.0, origin/feature-item-functionality, feature-item-functionality) IMS-29 DELETE functionality for items
IMS-28 UPDATE functionality for items
IMS-27 READ functionality for items
IMS-26 CREATE functionality for items
IMS-32 Created item controller (cleaned up itemDAO)
IMS-31 Create ItemDAO class
IMS-30 Created item class
(tag: v0.0.1.0, origin/feature-database-and-population, feature-database-and-population) IMS-16 Create and populated Order_items
IMS-15 Create and populated Order table
IMS-14 Created and populated items table
IMS-13 Create and Populated customer table
(tag: v0.0.0.1) IMS-19 test commit
(tag: v0.0.0.0, origin/master, origin/HEAD, master) init commit
```



Create Order Functionality Classes

ORDER IMPLEMENTATION

✓ IMS-36 ✓ RG

Create Order class

ORDER IMPLEMENTATION

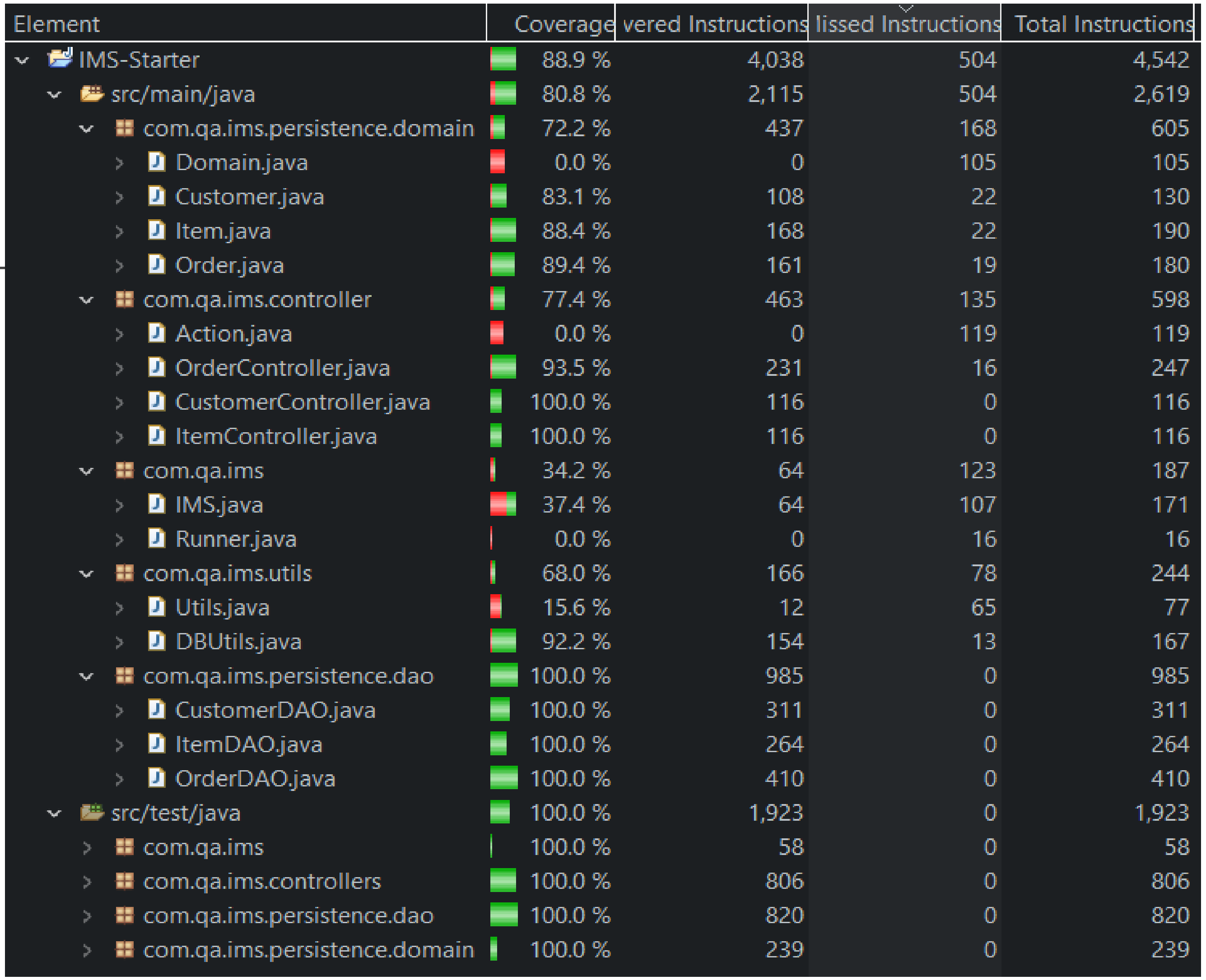
IMS-38 ✓ RG

1 commit

Testing

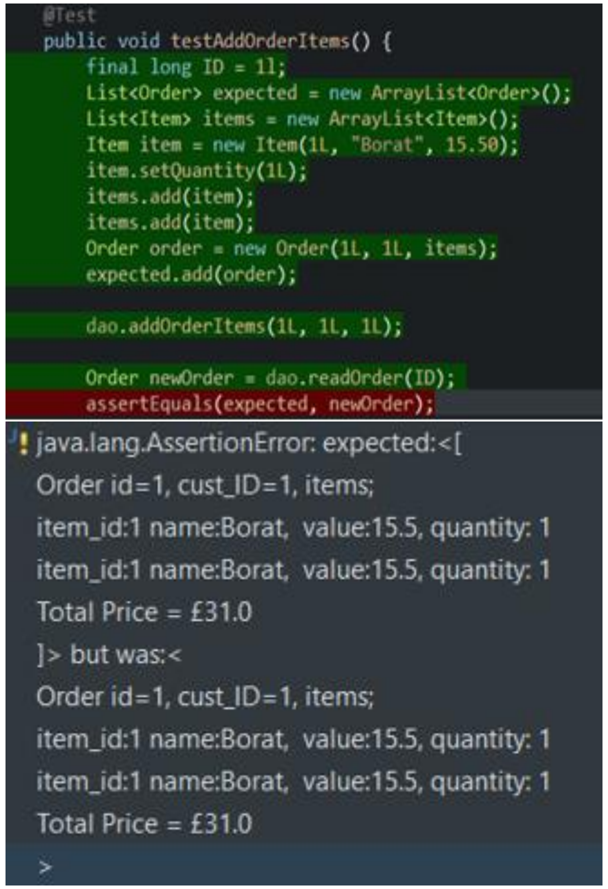
80.8% Line coverage in tests

62/62 Tests run successfully



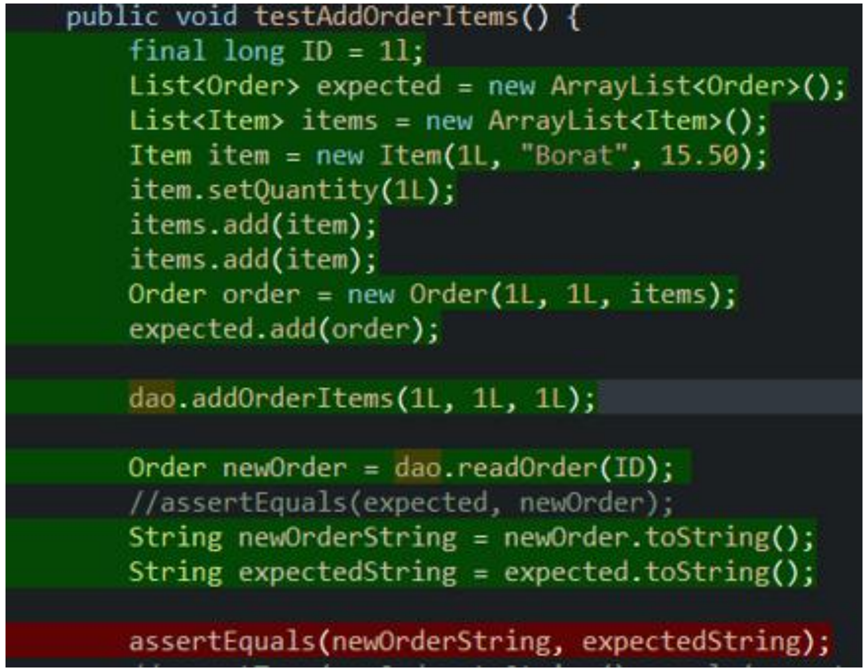
Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
IMS-Starter	88.9 %	4,038	504	4,542
src/main/java	80.8 %	2,115	504	2,619
com.qa.ims.persistence.domain	72.2 %	437	168	605
Domain.java	0.0 %	0	105	105
Customer.java	83.1 %	108	22	130
Item.java	88.4 %	168	22	190
Order.java	89.4 %	161	19	180
com.qa.ims.controller	77.4 %	463	135	598
Action.java	0.0 %	0	119	119
OrderController.java	93.5 %	231	16	247
CustomerController.java	100.0 %	116	0	116
ItemController.java	100.0 %	116	0	116
com.qa.ims	34.2 %	64	123	187
IMS.java	37.4 %	64	107	171
Runner.java	0.0 %	0	16	16
com.qa.ims.utils	68.0 %	166	78	244
Utils.java	15.6 %	12	65	77
DBUtils.java	92.2 %	154	13	167
com.qa.ims.persistence.dao	100.0 %	985	0	985
CustomerDAO.java	100.0 %	311	0	311
ItemDAO.java	100.0 %	264	0	264
OrderDAO.java	100.0 %	410	0	410
src/test/java	100.0 %	1,923	0	1,923
com.qa.ims	100.0 %	58	0	58
com.qa.ims.controllers	100.0 %	806	0	806
com.qa.ims.persistence.dao	100.0 %	820	0	820
com.qa.ims.persistence.domain	100.0 %	239	0	239

Issue

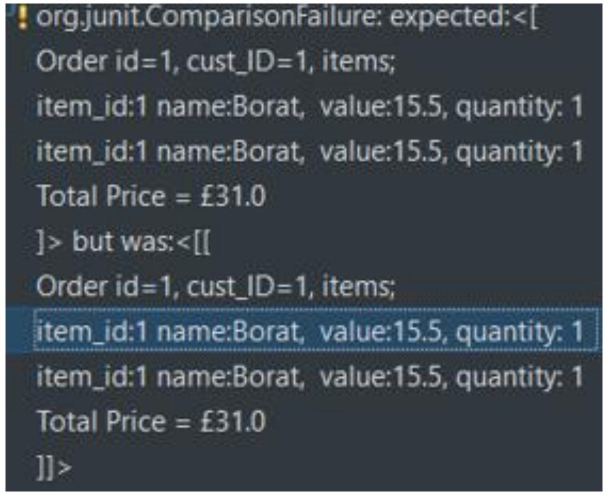


```
@Test
public void testAddOrderItems() {
    final long ID = 1l;
    List<Order> expected = new ArrayList<Order>();
    List<Item> items = new ArrayList<Item>();
    Item item = new Item(1L, "Borat", 15.50);
    item.setQuantity(1L);
    items.add(item);
    items.add(item);
    Order order = new Order(1L, 1L, items);
    expected.add(order);

    dao.addOrderItems(1L, 1L, 1L);

    Order newOrder = dao.readOrder(ID);
    assertEquals(expected, newOrder);
```

```
java.lang.AssertionError: expected:<[
Order id=1, cust_ID=1, items;
item_id:1 name:Borat,  value:15.5, quantity: 1
item_id:1 name:Borat,  value:15.5, quantity: 1
Total Price = £31.0
]> but was:<
Order id=1, cust_ID=1, items;
item_id:1 name:Borat,  value:15.5, quantity: 1
item_id:1 name:Borat,  value:15.5, quantity: 1
Total Price = £31.0
>
```

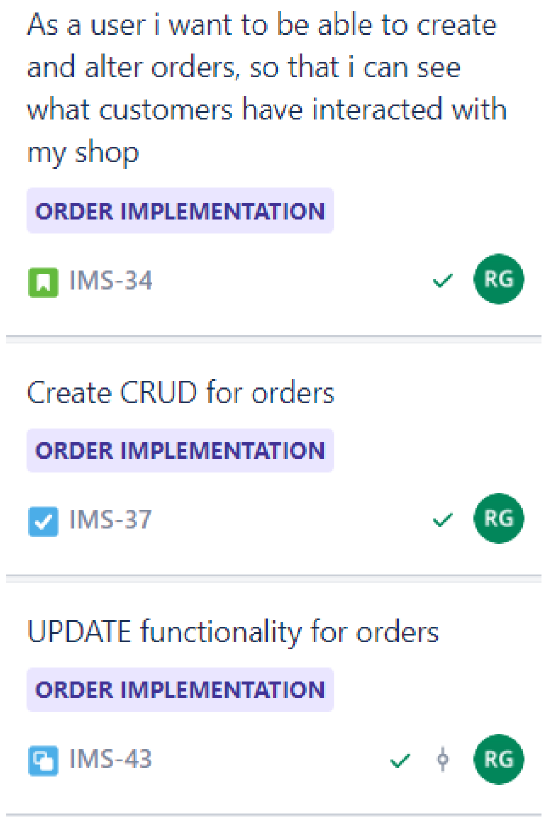


```
public void testAddOrderItems() {
    final long ID = 1l;
    List<Order> expected = new ArrayList<Order>();
    List<Item> items = new ArrayList<Item>();
    Item item = new Item(1L, "Borat", 15.50);
    item.setQuantity(1L);
    items.add(item);
    items.add(item);
    Order order = new Order(1L, 1L, items);
    expected.add(order);

    dao.addOrderItems(1L, 1L, 1L);

    Order newOrder = dao.readOrder(ID);
    //assertEquals(expected, newOrder);
    String newOrderString = newOrder.toString();
    String expectedString = expected.toString();

    assertEquals(newOrderString, expectedString);
```



```
org.junit.ComparisonFailure: expected:<[
Order id=1, cust_ID=1, items;
item_id:1 name:Borat,  value:15.5, quantity: 1
item_id:1 name:Borat,  value:15.5, quantity: 1
Total Price = £31.0
]> but was:<[[
Order id=1, cust_ID=1, items;
item_id:1 name:Borat,  value:15.5, quantity: 1
item_id:1 name:Borat,  value:15.5, quantity: 1
Total Price = £31.0
]]>
```

User Stories


As a user i want to be able to create and alter orders, so that i can see what customers have interacted with my shop

ORDER IMPLEMENTATION

IMS-34 ✓ RG

Create CRUD for orders

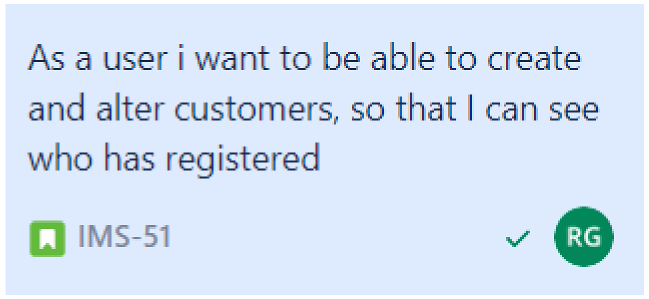
ORDER IMPLEMENTATION

IMS-37 ✓ RG

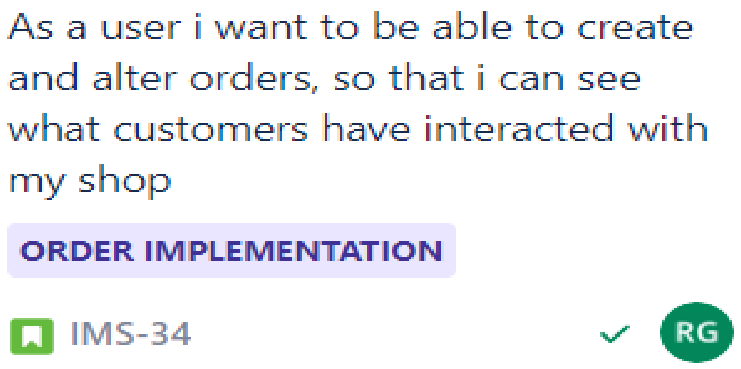
UPDATE functionality for orders

ORDER IMPLEMENTATION

IMS-43 ✓ RG

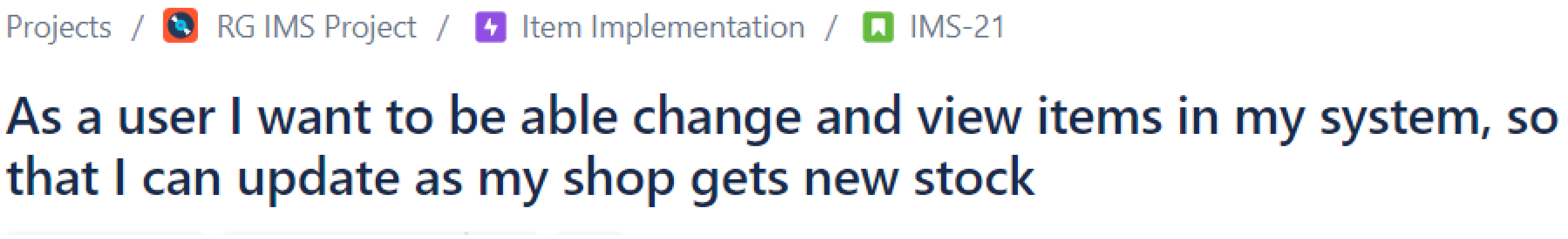

As a user i want to be able to create and alter customers, so that I can see who has registered

IMS-51 ✓ RG


As a user i want to be able to create and alter orders, so that i can see what customers have interacted with my shop

ORDER IMPLEMENTATION

IMS-34 ✓ RG


Projects / RG IMS Project / Item Implementation / IMS-21

As a user I want to be able change and view items in my system, so that I can update as my shop gets new stock

Sprint Review

- Managed to fully complete the project, including MVP, source control, jira, risk assessment, UML and ERD.
- Also including unit testing and mostly easy to read logger info.

- I really wanted to add a fully functioning stock system, this would work by checking to see if the Item table in the database had stock, if It did then orders were allowed otherwise not.
- I also wanted to add a more graphical/pretty user interface.

Sprint Retrospective

PROS

- Versioning / Source control.
- Managed to get MVP.
- Organisation.

CONS

- Misunderstanding user stories vs tasks.
- Didn't get any extra features added.
- Only minimum testing (80%).
- Problems understanding Mockito.

Conclusion

Reflection.

- Enjoyed the challenge.
- Enjoyed learning new technologies and how they are used in industry.
- Appreciate the help given when needed.

Future Steps.

- Fully Working stock check. – better stock management.
- Ability for customers or staff to log in (Different options per user)
- More graphical user interface.

Thanks for listening

Any Questions?