Test Plan

Table of Contents

SECTION 1: QA TEAM (and areas of responsibility)

1. Quality Assurance Lead

QA Lead	Office Phone	Mobile Phone	Email
Muhamad Nurie	514-963-1042	514-917-1040	mznurie@msn.com

2. QA Team

MEMBER	RESPONSIBILITIES
Muhamad Nurie	- Direct QA team. - Develop QA processes and procedures - Manage testing schedules - Coordinate bug fixes with development team
Carlin Lee	- Performance QA Engineer: In-charge of testing the product's stability and responsiveness. - Maintain Documentation: Document the test coverage and other metrics for system analysis. Ensure all use-cases are covered by tests.
Mark Said	- Identify bugs and issues: Develop unit and integration tests to check the system against bugs and issues. - Test automation engineer: In-charge of automating the testing and development pipeline with continuous integration tools like TravisCI
Ayman Shehri	- Liaise with developers: In-charge of communicating with the system developers to notify them of any bugs discovered. - Business Analyst: In-charge of ensuring the system meets the business objectives of the client and will satisfy their technical needs.
Adam Yafout	- Performance QA Engineer: Test the product's stability and responsiveness. - Identify bugs and issues: In-charge of developing unit, integration and system testing to test the system for any bugs or issues.

3. External Test Resources

NAME	DETAILS
Servers\Testing Environment	The system should be tested in different test environments. Tests must be conducted on different operating systems, namely, windows, osx and linux.
Testing Tools	For continuous integration, TravisCI is used and is directly integrated with GitHub. For test analysis and to ensure sufficient coverage of test classes, OpenClover and JaCoco are used.
Data	To ensure the system is tested under the correct constraints, sufficient and accurately simulated data must be added to the tests.

SECTION 2 : TESTING PROCEDURES

1. General Approach

a. Basic Responsibilities of Test Team

i. Bugs

1. It is the responsibility of the automation testing engineer and manual testing engineers to detect the bugs as soon as possible with the help of extensive manual testing and high-coverage automated tests.
2. The team must identify the steps causing a bug, and document the bug reproduction procedures.
3. The identified bugs must be communicated with the backend team for resolution as soon as possible. Furthermore, they must be notified of the priority and criticality level of the bugs.
4. The testing team should help the development team in the resolution of the bug.
5. Each bug must be tracked throughout its lifecycle, until it is resolved. Resolved status is achieved when the QA team certifies the bug can not be reproduced anymore.

ii. All production level builds must pass all the automated tests with continuous integration.

iii. The bugs are tracked and communicated to the development team using Atlassian Jira.

2. Daily Activities

a. The Build

i. The QA team must identify and generate the build that encompasses the features implemented across multiple branches.

ii. Run the daily regression tests, as described in “Daily Tests” which follows.

iii. If everything is okay, post the build so everyone can get it.

iv. If there’s a problem, send an email message to the entire dev team that the new build cannot be copied, and contact whichever developers can fix the problem.

v. Decide whether a new build needs to be run that day.

b. Daily Tests

Run tests to verify functionality of different modules, namely inventory, users, accounting, production and sales. The tests should cover all the controller classes under each module.

3. Daily Reports

The tests will automatically generate reports using OpenClover and Jacoco. If the reports fail the 50% coverage parameter, the developers are alerted and the build dropped for the day.

4. Weekly Activities

a. Weekly tests

Run extensive tests for not only the controller classes, but also for the models.

b. Weekly Reports

Generate reports using the OpenClover and Jacoco plugins. Ensure there is adequate coverage and the code remains in a healthy state, judged by parameters like code complexity.

5. Integration of Reports from External Test Groups

The frontend being implemented in React will be tested using different frameworks. Ensure that the backend and the frontend are connected properly: all requests are made to correctly exposed backend APIs.

The backend being implemented in Java will be tested with jUnit and Spring Testing Framework. It must be ensured that all APIs return the expected response.

Any bugs encountered will be posted on the Jira board and will be accessible by both the frontend and the backend team.

6. Focus Testing (if applicable)

The only customer we will communicate with is the Product owner and we will meet with him to extract and clarify their wants and needs for the ERP system. The meetings will be virtual and held through Zoom or Google Meet. If possible, the entire development team will meet with the Product Owner to elicit feedback on their current progress. The feedback will be recorded on a shared Google Doc.

7. Compatibility Testing

The development team shall use Docker to test the application on Windows and Mac OS. Compatibility tests will also be run on Android and iOS. If there are any defects detected during compatibility testing, they must be re-tested to confirm its existence then communicated with the rest of the development team. Once the team confirms the bug's existence, it must then be reported through Jira.

SECTION 3: HOW TESTING REQUIREMENTS ARE GENERATED

- For each sprint, when the user stories are assigned to that sprint from that backlog, the testing team will develop some testing criteria. The testing criteria will ensure that the system is compliant with the client's business objectives. The testing criteria will also ensure that any and all use-cases are completely covered by the tests implemented.
- The testing team will have internal meetings to decide on the non-functional requirements, like performance and security metrics, that the system must pass before being moved into the production phase.
- The testing team will both manually and automatically test the system. Any issues identified will be passed on to the development team with its priority and criticality status.
- All bugs reported to the development team must include information about how they can be replicated.
- If a part of the system has been discovered to be more bug-prone, testers will spend more time testing that part of the software, and possibly identify the root cause of the bugs.
- In case of status change of the bug status in the bug tracking system, some tester or testers depending on the criticality level of the bug, will verify that the bug has indeed been removed and can't be replicated. Other status changes will also require a tester to investigate or work on the bug together.

SECTION 4: TESTING FRAMEWORK

Spring testing framework will be used to test our application. It provides extensive support for both unit testing and integration testing, providing specialized support for testing the web aspect.

Unit Testing

POJO

Plain old java objects can be tested with jUnit without any need of support provided by the Spring Testing Framework.

Mock Objects

Spring Testing Framework allows users to mock Spring specific beans with Spring functionality.

- **Environmental mocking:** MockEnvironment and MockPropertySource allows testing environment specific properties. This will be useful in testing hibernate connection to mysql database.

- **ServletAPI:** org.springframework.mock.web package contains several servlet api mock objects allowing users to test web controllers, filters and contexts.

Unit Testing Support Classes

For unit testing of Spring beans, containers and utilities, Spring Testing framework provides a number of classes. They fall in the following two categories:

- **General Testing Utilities:** This can provide support for testing spring specific objects like Hibernate ORM that we use in our application, or use of Spring specific annotations like @PostConstruct and @PreDestroy.
- **Spring MVC Testing Utilities:** This can be used for Spring *ModelAndView* objects using *ModelAndViewAssert* methods.

Integration Testing

It is important to be able to perform integration testing without deploying the application. This allows for testing of correct wiring of IoC contexts, and Hibernate ORM tool.

Spring Context and Dependency Injection

These can be tested using the TestContext packages. Normally, specific testing of spring contexts and dependency injection is not necessary, as the default test class include ContextLoads() test method

JBDC Testing Support

The *org.springframework.test.jbdc* package has *JDBCTestUtils,* which are utility functions to test the connectivity to the database and their functions. It includes static utility functions like *dropTables()* for relational db testing.

MockMVC

This allows for testing the full Spring MVC request handling, but using mock requests and response objects instead of a running server. MockMVC allows performing all requests that can be performed with the server, for instance including the authentication tokens in headers or async requests, and checking the response in any format. This will be the main package for testing the controller classes in our application.

Static Analysis Tool

We shall use Embold.io as our static analysis tool in order to obtain software metrics for our ERP application.

SECTION 5: BUG TRACKING SOFTWARE

- Jira software shall be used for bug tracking
    - Each team member (9 total) shall use the bug tracking template
    - Everyone on the team should have access to the bug list
        - Each team member must have access to the project
    - “How to report a bug” instructions for using the system
        - Once a bug is identified, create an entry and add relevant details including: description, severity level, steps to reproduce, screenshots and environment if applicable, reported by, assigned to, status, resolution, date reported, date of fix, version and possibly the project files that are affected.
        - Once an issue is created for the bug, then it must be ranked and prioritized based on its importance and urgency.
        - The issue creation can be routed to the correct team member through email.

SECTION 6: BUG CLASSIFICATIONS

Each defect shall be given a severity and a priority. Each severity can be paired with any priority. The following bug classification system is following ScienceSoft’s own version and shall be transferable to our own project.

Severity	**Description**
Critical (red)	Blocks an entire system’s or module’s functionality or causes unrecoverable data loss. Testing cannot proceed further without such a defect being fixed and the product cannot be released.
High (orange)	Affects key functionality of an application wherein the app does not behave as it is expected to, as stated in the requirements. There may exist an unsatisfactory workaround. The software cannot be released.
Medium (yellow)	Affects a minor function in such a way that it does not behave as it is stated in the requirements. A reasonably satisfactory workaround exists. The product can be released if the bug is documented.
Low (green)	Primarily related to an application’s UI. There’s a workaround or the issue can be ignored. It does not impact a product release.

Priority	**Description**
Urgent (red)	Must be fixed within 24 hours after being reported.
High (orange)	Must be fixed in an upcoming release in order to meet the exit criteria.
Medium (yellow)	To be fixed in an upcoming release or subsequent release.

Low (green)	Do not need to be fixed in order to meet the exit criteria, but require fixing before an application becomes generally available.

SECTION 7: BUG TRACKING

- Once a bug is identified and the issue is created, it's status is Active, it must be classified by at least one member of the QA team. During a standup meeting, the team shall discuss the bug and if the severity and priority is high enough, then the scrum master shall assign the bug to a willing developer. The issue should then be marked in progress.
- Once the bug is fixed, it will be marked as Test, then it must be retested by the QA team. The QA team gives their approval if the tests pass, then the issue can be marked as Verified. In the next standup, the issue can then be Closed.

Issue Status	Description
Open	Identified and unassigned
Active	Investigation is underway
Resolved	Fixed and ready for testing
Closed	Can be closed after QA retesting or it is not considered to be a defect

SECTION 8: SCHEDULING AND LOADING

Rotation Plan : There shall be 2 active QA members at a time, chosen by the scrum master. Initially, there will be one two active QA members, then after 15 days one member shall be rotated and another QA member is added. After another 15 days, the longest active QA member rotates and so forth.

Loading Plan : There shall be 1-2 QA testers in sprint 1 and 2 of the project, then 2 during sprint 3-5.

SECTION 9: EQUIPMENT BUDGET AND COSTS

For this project, all team members already have their own computers or laptops that they can use along with internet connection. All other versioning, managing, testing and coverage software used in this project is free.

SECTION 10 : SPRINT 2 UNIT TESTING

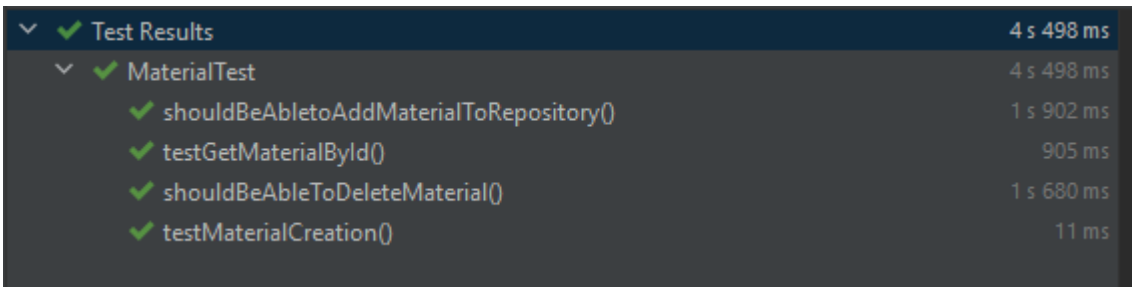
MaterialsController

The MaterialsController is tested by the MaterialsTest.java file.

Methods covered :

- Material instantiation
- Create material
- Retrieve all materials
- Find material by ID

```java
@SpringBootTest(
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT
)
@AutoConfigureMockMvc
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class MaterialTest {
    @Autowired
    protected MockMvc mockMvc;
    @Autowired
    private MaterialRepository materialRepository;

    @Test
    void testMaterialCreation(){
        Material material = new Material();
        assertNotNull(material);
    }

    @Test
    void shouldBeAbletoAddMaterialToRepository(){
        Material material = new Material();
        int count = (int)  materialRepository.count();
        materialRepository.save(material);
        assertEquals( expected: count+1, (int) materialRepository.count());
    }

    @Test
    void shouldBeAbleToDeleteMaterial(){
        Material material = new Material();
        int count = (int)  materialRepository.count();
        materialRepository.save(material);
        assertEquals( expected: count+1, (int) materialRepository.count());
        materialRepository.delete(material);
        assertEquals(count, (int) materialRepository.count());
    }


    @Test
    @WithMockUser(authorities = "ROLE_ADMIN")
    public void testGetMaterialById() throws Exception {
        this.mockMvc
                .perform(MockMvcRequestBuilders.get( urlTemplate: "/materials/{id}",  ...uriVars: 0) .with(csrf()))
                .andDo(MockMvcResultHandlers.print()).andExpect(status().isOk())
                .andReturn();
    }

```



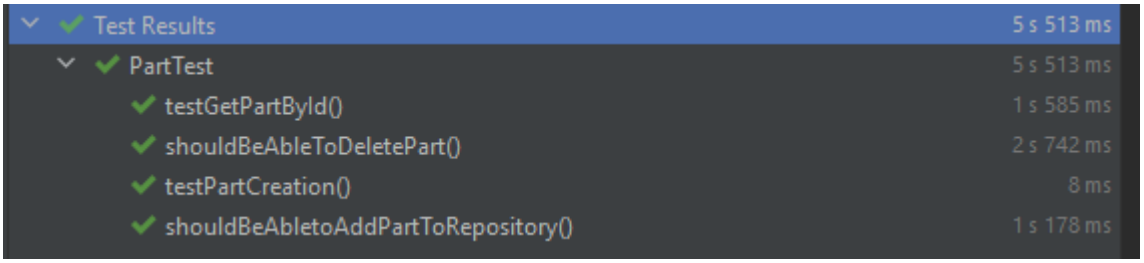
PartsController

The PartsController is tested by the PartsTest.java file.

Methods covered :

- Part instantiation
- Create part
- Retrieve all parts
- Find part by ID

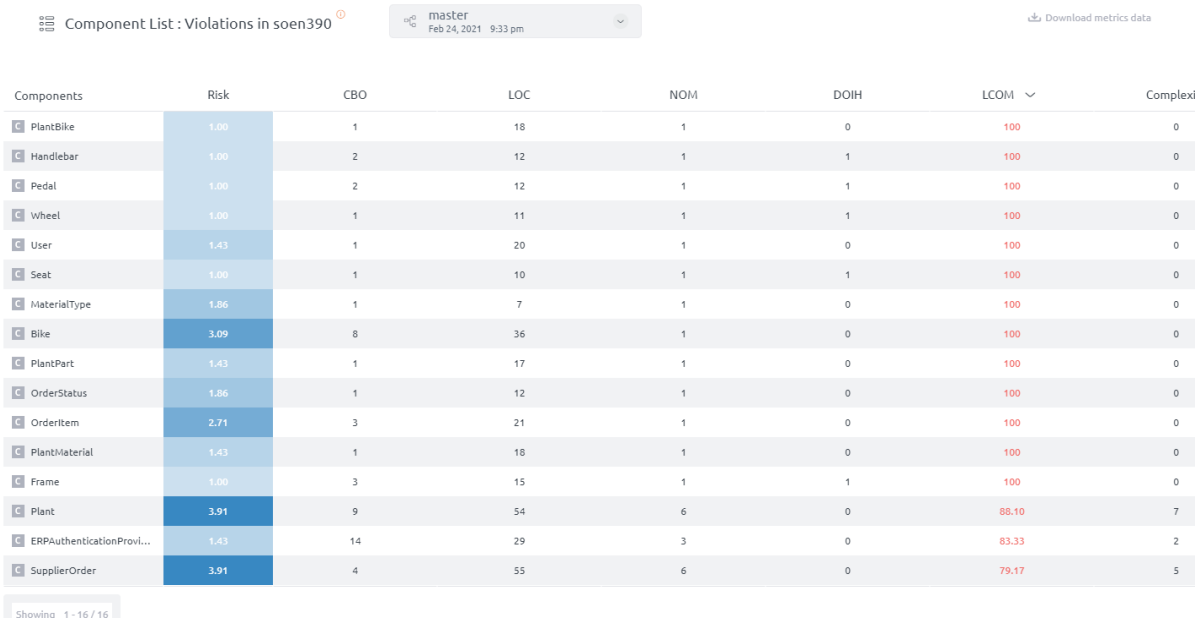
```java
@SpringBootTest(
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT
)
@AutoConfigureMockMvc
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class PartTest {
    @Autowired
    protected MockMvc mockMvc;
    private static final String username = "zubair@gmail.com";
    @Autowired
    private PartRepository partRepository;

    @Test
    void testPartCreation(){
        Part part = new Part();
        assertNotNull(part);
    }

    @Test
    void shouldBeAbletoAddPartToRepository(){
        Part part = new Part();
        int count = (int)  partRepository.count();
        partRepository.save(part);
        assertEquals( expected: count+1, (int) partRepository.count());
    }

    @Test
    void shouldBeAbleToDeletePart(){
        Part part = new Part();
        int count = (int)  partRepository.count();
        partRepository.save(part);
        assertEquals( expected: count+1, (int) partRepository.count());
        partRepository.delete(part);
        assertEquals(count, (int) partRepository.count());
    }


    @Test
    @WithMockUser(authorities = "ROLE_ADMIN")
    public void testGetPartById() throws Exception {
        this.mockMvc
                .perform(MockMvcRequestBuilders.get( urlTemplate: "/parts/{id}",  ...uriVars: 7) .with(csrf()))
                .andDo(MockMvcResultHandlers.print()).andExpect(status().isOk())
                .andReturn();
    }
```



SECTION 11 : SPRINT 2 BACKEND BUG REPORTS

Bug	Severity	Priority	Detected Date	Resolved Date	Fix
UserTest.java not testing requests properly	**Low** ; the actual code is still functioning well	**Low** ; we know it works well from consistent usage	Feb. 9, 2021	Feb. 11, 2021	Used Spring's MockMVC to mock API requests
MaterialsController.java not connected to the database	**High** ; can cause further confusion down the road if not actually connected	**High** ; must be done for Sprint 2, used in defining materials lists	Feb. 18, 2021	Feb. 19, 2021	Used Spring's Repository functionality
Infinite Recursion in Plants Retrieval	**Critical** ; hinders us from further implementing anything Inventory related	**High** ; some Inventory functionality must be done before the end of Sprint 2	Feb. 20, 2021	Feb. 23, 2021	Removed JSONIgnore annotation
MaterialsTest.java and PartsTest.java find by ID tests failing	**Low** ; actual code works	**High** ; need Controller coverage for Sprint 2	Feb. 23, 2021	Feb. 23, 2021	Used Spring's MockMVC to mock API requests

SECTION 12 : SPRINT 2 METRICS



Component List : Violations in soen390

master
Feb 24, 2021 9:33 pm

Download metrics data

Components	Risk	CBO	LOC	NOM	DOIH	LCOM	Complexi
PlantBike	1.00	1	18	1	0	100	0
Handlebar	1.00	2	12	1	1	100	0
Pedal	1.00	2	12	1	1	100	0
Wheel	1.00	1	11	1	1	100	0
User	1.43	1	20	1	0	100	0
Seat	1.00	1	10	1	1	100	0
MaterialType	1.86	1	7	1	0	100	0
Bike	3.09	8	36	1	0	100	0
PlantPart	1.43	1	17	1	0	100	0
OrderStatus	1.86	1	12	1	0	100	0
OrderItem	2.71	3	21	1	0	100	0
PlantMaterial	1.43	1	18	1	0	100	0
Frame	1.00	3	15	1	1	100	0
Plant	3.91	9	54	6	0	88.10	7
ERPAuthenticationProvi...	1.43	14	29	3	0	83.33	2
SupplierOrder	3.91	4	55	6	0	79.17	5

Showing 1 - 16 / 16

LCOM - Lack of Cohesion of Methods :

- Percentage is high for the majority, thus this indicates that the code should be refactored and split into more subclasses with low cohesion. However, upon further investigation of the classes that have 100% cohesion, they are much smaller classes that only have getters and setters, but those are injected by Spring's annotations, so the analysis tool may not have captured this fact so it thinks its variables are not being used at all, hence the 100% lack of cohesion.

CBO - Coupling Between Objects :

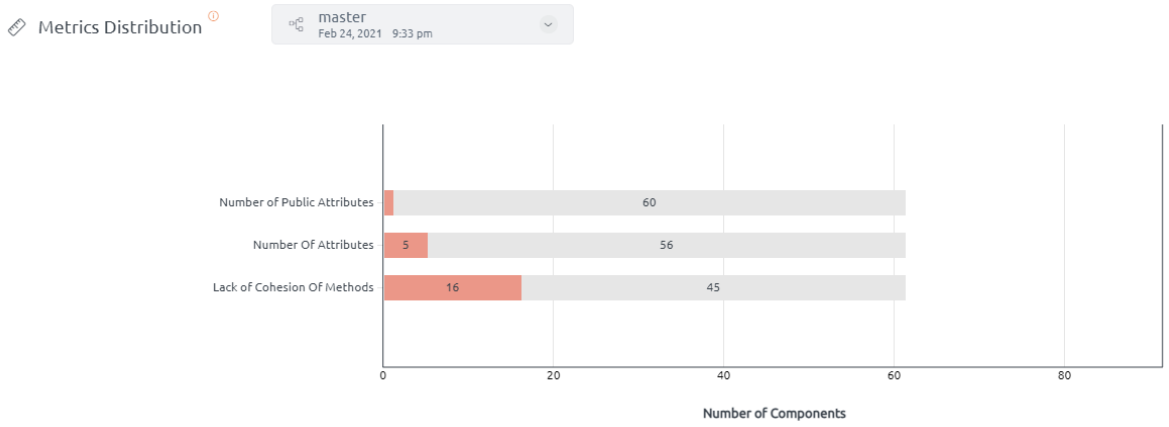
- Strong coupling increases the difficulty in understanding and testing a class. The average CBO is 4 while the highest is 14. The highest is from the main application so it contains many references to other classes required for initialization.

DOIH - Depth of Inheritance Hierarchy :

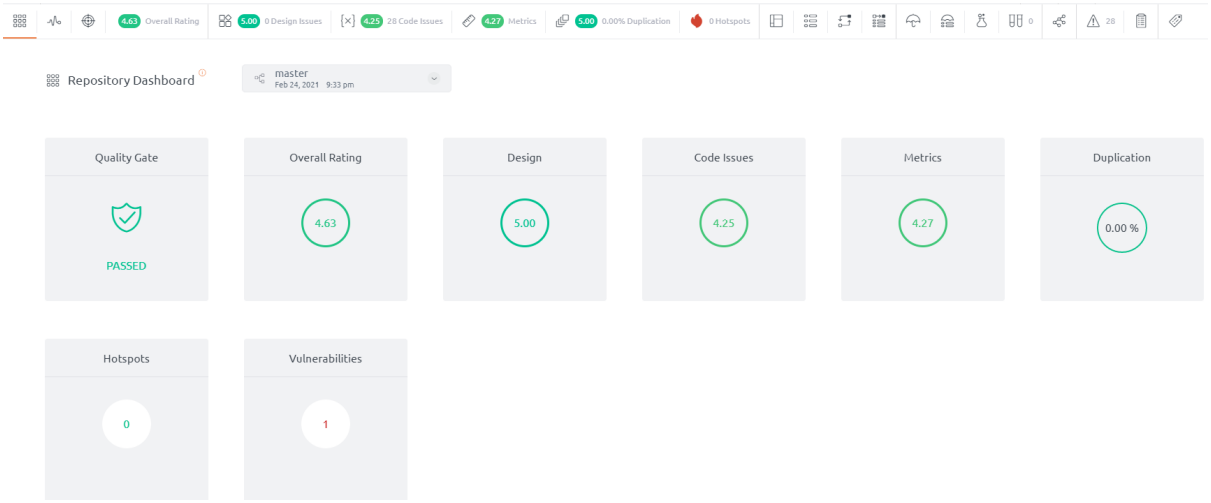
- High value of DOIH indicates high reuse. In our case, the average DOIH is zero because very little inheritance was used. Spring already enforces many design patterns, so it was not necessary for us to make things more complicated. However a DOIH of less than 2 indicates poor exploitation of object oriented principles, so we will try to implement more abstract classes next time.

Complexity - Cyclomatic Complexity :

- In general a Complexity score between 1-10 means the code is structured and well written, has high testability, and having less cost and effort to maintain.
- Our classes are within this range meaning it is easily testable.



Red are violations


4.63 Overall Rating
5.00 0 Design Issues
4.25 28 Code Issues
4.27 Metrics
5.00 0.00% Duplication
0 Hotspots
28
Repository Dashboard
master
Feb 24, 2021 9:33 pm
Quality Gate
PASSED
Overall Rating
4.63
Design
5.00
Code Issues
4.25
Metrics
4.27
Duplication
0.00 %
Hotspots
0
Vulnerabilities
1