Test Report

Virtual Letter of Life (VLOL) Application

Version 1.0

Sefanit Urgessa

Mohammed Allibalogun

Robert Garcia

Meron Getachew Debela

Charles Baisie

Augustin Mwamba

Jamal Bourne

SWEN 670 Software Engineering Project

University of Maryland Global Campus

Date: 08/04/2020

**Table of Contents**

[1. Introduction 4](#_Toc47475435)

[1.1 Application Overview 4](#_Toc47475436)

[1.2 Purpose 4](#_Toc47475437)

[1.3 Scope 4](#_Toc47475438)

[2. Testing 4](#_Toc47475439)

[2.1 Test Report Strategy 4](#_Toc47475440)

[2.1.1. Unit Testing - JUnit 5](#_Toc47475441)

[2.1.2. Integration & System Testing - SonarQube/JUnit 5](#_Toc47475442)

[2.1.3. Acceptance Testing – Selenium 5](#_Toc47475443)

[2.2. Test Objective Table 5](#_Toc47475444)

[2.3. Tests Conducted 7](#_Toc47475445)

[3. Test Execution Findings 8](#_Toc47475446)

[3.1 Metrics 8](#_Toc47475447)

[3.2. SonarQube Overview 9](#_Toc47475448)

[3.3. Selenium Overview 10](#_Toc47475449)

[4. Exit Criteria 10](#_Toc47475450)

[**APPENDIX A –** Sample QR Code for Testing 10](#_Toc47475450)

**Tables of Figures**

[Figure 1. SonarQube Report 7](#_Toc47475450)

**Tables of Tables**

[Table 1 Test Guidelines 3](#_Toc47475450)

[Table 2 Test Case Group 5](#_Toc47475450)

[Table 3 Test Case Metrics 8](#_Toc47475450)

# 1. Introduction

The VLOL Application will utilize the following Test Report to outline the purpose, scope, application overview, test strategies, and test results findings that developers gathered throughout the SDLC process.

## 1.1 Application Overview

The Virtual Letter of Life (VLOL) is a web application that was created to improve emergency patient care at remote location in which the traditional Letter of Life information is not readily available. The VLOL system main purpose is to provide users and first responders with a web-based version of the letter of life. At its core, the application allows authenticated users to retrieve User’s Letter of Life information from a data source (an external database) using a desktop browser or a mobile device, and to create, update and delete records based on roles and privileges they are granted within the system. Unauthenticated users (users without an account or credentials) can access the application’s contact page, information page and registration page.

## 1.2 Purpose

The purpose of this Test Report is to outline the on-going test activities that were performed on the Virtual Letter of Life system (VLOL) application. The implementation of these test strategies ensured that all functional and non-functional requirement established on the SRS documentation were properly working. In this Test Report we will provide the following:

* The list and description of test strategies used to create test cases and performed analysis against the VLOL application in accordance to the Master Test Plan
* Provide table reports with results from each test cases
* Discuss test finding and potential test cases that failed

## 1.3 Scope

The scope of this document is to discuss specific test cases that validate and verify the VLOL functional and non-functional requirements that are presented in both the Software Requirement Specification (SRS) and Master Test Plan documentation. The report will summarize the data for the client and project manager explaining the applied test methodologies, the results gathered from the test cases, as well as the exit criteria.

# 2. Testing

The Virtual Letter of Life software will be subjected to various forms of testing procedures to ensure each functional and non-functional requirement established in the SRS and Master Test Plan are met and working properly.

## Test Report Strategy

This section outlines the different testing strategies utilized for the VLOL application.

### **2.1.1. Unit Testing - JUnit**

The back-end portion of the VLOL application was written in the Java programming language. Testers have implemented the White-Box strategies technique. Testers will utilize Junit via the Bottom Up approach, which is the conventional testing framework that most developers use when building an application in Java. The JUnit framework uses assert methods to perform proper back-end Unit Testing. Lastly, in order to reach a code coverage range of 80% - 100%, testers will utilize the open-source software SonarQube. This software tool will analyze the project source code and generate a code coverage measuring metrics that testers will check to ensure the code coverage reaches the 80% -100% range.

### **2.1.2. Integration & System Testing - SonarQube/JUnit**

For the VLOL application, developers will execute the Integration and System Testing in sequential phases. For this instance, testers will implement the Big Bang approach where internal unit test will be conducted, and then test the system as a whole. Thus, the joint usage of JUnit and SonarQube will help handle the integration and system testing. The test classes and test suit created through JUnit will be uploaded and analyze by the SonarQube. This on-going transition will provide a continuous integration and continuous development phase (CI/CD). In return, SonarQube will generate a detailed report, which will inform developers and testers of the existence of bugs, security vulnerabilities, potential duplicates, and code coverage analysis & percentage.

### **Acceptance Testing – Selenium**

The acceptance testing will be completed using Black-Box technique. VLOL developers will follow the Black-Box technique using the front-end automated testing tool Selenium. All completed functional requirements stated in the SRS will be validated using Selenium software tool.

**Note.** All the deletion functionality testing using Selenium must be performed at the end.

## Test Objective Table

Table 1 below provides the guidelines that developers will utilize to conduct testing based on the strategies described in the section 2.1.

*Table 1 Test Guidelines*

| **Test Type** | **Testing Objective** | **Technique** | **Completion** |
| --- | --- | --- | --- |
| **Unit Test** | Utilize the JUnit framework to conduct the back-end testing and use the SRS non-functional requirement to create test cases. | White-Box | All of the test cases successfully pass without prompting any failures with the following results:   * Correct error message gets display upon invalid data entry. * The valid data yield the correct output. * All use cases are applied. |
| **Integration & System Test** | The joint usage of JUnit and SonarQube will test whether the application is working well internally as a unit, and externally as a system. | N/A | The Code Coverage percent completion will be predicated by the report generated by the SonarQube. The following metrics will be listed in the report:   * Reliability – Bugs * Security – Vulnerabilities * Coverage – Unit Test Code * Duplications |
| **Acceptance Test** | Ensure that the functionalities of an application are working properly by inserting inputs and verify that the output returns as expected. | Black-Box | All the test cases created based on the SRS functionality are run through the Selenium testing with the following results:   * Correct error message is displayed upon invalid data entry. * The valid data yield the correct output. * All use cases are applied. |

The standards below are outlined to ensure best user experience and protection of personal data,

* Personal Health Data will be date and time stamped inclusive to medication management and continuity of care
* Access is monitored by recording who is accessing patient information, which can be used for data audit
* Reduced time spent searching through accurate patient information
* Enhance security to ensure the privacy of patients’ sensitive data storage and accessibility
  + Allowing information to be viewable to the users with regard to their level of access based upon their level of position.
* Accessibility to patients to easily update their medical information to ensure accurate information sharing between first responders and other authorized personnel
* Ease of accessibility for patients to easily and continually update their medical information to ensure that accurate information is shared between first responders and other authorized personnel

## Tests Conducted

Developers and testers used JUnit to create 189 test cases to cover the non-functional requirements, and 28 test cases to cover the functional requirements. Table 2 provides the descriptions of each test groups.

*Table 2 Test Case Group*

| Test Case ID | Test Description |
| --- | --- |
| Test-REQ-1.1 – Test-REQ-1.39 | The test cases in this group focus on the functional requirements in which authenticated and unauthenticated user interacts with various web components of the VLOL application. The Selenium software will record all tester interaction with the application (test transaction) to ensure that the UI screens projected are displaying the correct content. |
| Test Class name:   * Allergytest.java * ConditionTest.java * MedicationTest.java * RoleTest.java * UserTest.java | The following test classes will provide the test cases used for the backend portion of the application. Developers will use the JUnit framework and assert methods for the model classes in the application. |

# 3. Test Execution Findings

## 3.1 Metrics

The tests cases have been divided in four test suits. The tests will be grouped in suits based on the type of user that is allowed to perform the functionality according to the SRS document. Below are the suits name and the tests that each suit will contain:

* [**Suite ID-**](chrome-extension://mooikfkahbdckldjjndioackbalphokd/index.html)**Admin** – Test-Req-1.26 – Test-Req-1.39
* [**Suite ID-Agent**](chrome-extension://mooikfkahbdckldjjndioackbalphokd/index.html)– Test-Req-1.18 – Test-Req-1.25
* [**Suite ID-Participant**](chrome-extension://mooikfkahbdckldjjndioackbalphokd/index.html)– Test-Req-1.5 – Test-Req-1.10
* [**Suite ID-Provider**](chrome-extension://mooikfkahbdckldjjndioackbalphokd/index.html)– Test-Req-1.11 – Test-Req-1.17
* **Suit ID-Unidentified User** – Test Req-1.1 – Test-Re1-1.4

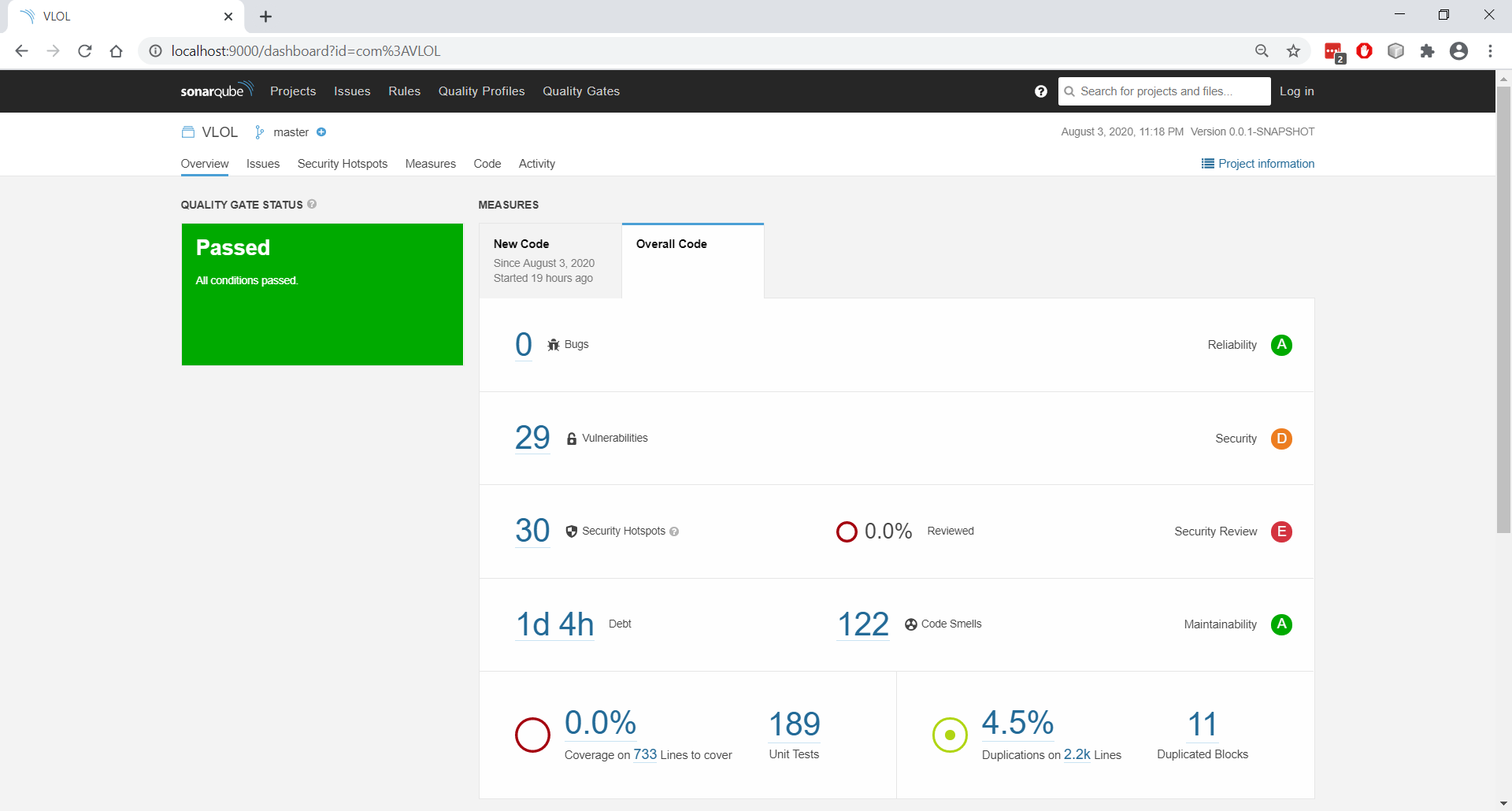
As stated in the previous segment, each “Suite ID-xxx” will host test cases that satisfy the functional requirements in the SRS document. To review the tester interaction recording, locate the Selenium file “VLOL Test Project – Selenium”.

Test metrics will provide developers the diagnostic report needed to determine the quality of the software application. These metrics will help developer ensure that all functional and non-functional requirements are tested and covered. The functional requirements will use four test metrics: Planned Test, Executed Test, Passes Test, and Failed Test Cases. SonarQube provided the statistical number from the result gathered by running the JUnit test cases and the Selenium test result. Table 3 displays the number of test cases planned, the number of test cases executed, the test cases that passed, and the test cases that failed.

*Table 3 Test Case Metrics*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test tool | Planned Test Cases | Executed Test cases | Passed Test Cases | Failed Test Cases |
| Selenium | 39 | 28 | 28 | 0 |
| JUnit | 189 | 189 | 189 | 0 |

Figure 1 provides a screen capture of the SonarQube report as of August 4, 2020. It shows the results obtained regarding the Reliability – Bugs, Security Vulnerabilities, Code Coverage, and Duplications.



*Figure 1. SonarQube Report*

## SonarQube Overview

The following is an overview of the current issues identified by SonarQube scanning, adjusted by the team to reflect ground truth (see the SONARQUBE markdown file in the repository for instructions on how to use the scanner with Maven).

* Lines of Code Analyzed: 2,200
* Total Issues: 151
  + Detailed screenshot of 151 issue output from SonarQube can be reviewed in the file ‘issues.pdf’.
* Current Technical Debt Owed to Quality Improvement: 28 hours
* Bugs: 0
* Potential Vulnerabilities: 29
* Potential Security Hotspots: 30
* Potential Code Smells: 122
* Unit Tests: 189
* **Unit Test Coverage:** SonarQube reports 0% code coverage, but the 189 tests protect the five Model classes, which account for 216 of the 245 total methods, for 88.1% method coverage. These counts include untestable code that do not return values or throw exceptions.
* Duplications: 4.5%

## Selenium Overview

The development team was able to successfully develop 28 of the 39 functional requirements outlined in the Software Requirement Specification document. The test cases for the 28 requirements can be found in the Selenium Test suit file. Below is a list of the requirements that were tested based on user privileges:

* Unauthenticated and Unauthorized User – 3 requirements
* Program Participant – 4 requirements
* Medical Services Provider – 5 requirements
* Patient Agent – 5 requirements
* System Administrator – 11 requirements

For further information, refer to the VLOL Application Development Turnover document in the section 4.2 and 4.3.

# 4. Exit Criteria

The Virtual Letter of Life Application has a set of pre-defined Exit Criteria that are dependent on the completion of all level of testing (Unit, System, Integration, and Acceptance). The following list outlines the exit criteria:

* Test cases execution should be within a range of 80-100% code coverage.
* Execution test coverages for functional and non-functional requirement listed in the SRS document.
* The test report should not have any sever or high priority error, including critical bugs within the application.
* If any open defects remain in the application, developers should make annotation for the upcoming team who will take over the ongoing development of the application.

**APPENDIX A – Sample QR Code for Testing**

