Test Document

Cardinal Project for Blood Bank Resource Management

SWE 3313 | Group 1

ANDUJAR BRUTUS | PAMIR AHMAD | AUDREY ALLEN | EL ARBI BELFARSI | THOMAS BREITUNG

VER 1.0

11/21/2021

1. Testing Strategy

1.1 Overall Strategy

While developing Cardinal, many things will be taken into consideration during the testing phase. For low-level features and requirements, white-box testing methods will be used, as well as black-box methods for high-level requirements. The performance of the application will be determined through adequacy criterion such as percentage of accuracy, successful input/output and storage of data, and successful and accurate loading of the map API. Throughout the testing process, bugs and enhancements will be dealt with via assigned tiers of severity, which will then be the basis of their resolution.

MySQL will be used for the database portion of this application, and there are several automated testing tools that support it, two of which are the Database Performance Monitor by Solarwinds and the DTM Data Generator. For the portions of the application that use a language other than SQL, Atlassian's Crucible can be used to review code and parse through bugs.

1.2 Test Selection

For unit and integration testing, white-box methods: statement coverage and branch coverage will be employed to ensure quality at the lowest levels. System and acceptance testing will utilize black-box methods such as boundary value analysis and decision table testing to ensure functional and non-functional requirements are met.

1.3 Adequacy Criterion

ID	Requirements Description	Use Case	Test Cases	Adequacy Criterion
1	Center locator	Find donation center	TC_1	The requirement is met when the map API can consistently locate & display donation centers for users with 85 percent accuracy.
1.1	Map API	Find donation center	TC_1.1	The requirement is met when the map API loads and displays accurate geographical information.
2	Blood availability per location	Identify blood availability per location	TC_2	The requirement is met when a blood donation center can post their supply availability and that information can be read from a qualified user account.

2.1	Private Database	Identify blood availability per location	TC_2.1	The requirement is met when necessary data can be stored and pulled by qualified sources for use in the main software interface.
3	Donation center prioritization	Donation center priority detection	TC_3	The requirement is met when multiple donation center's may display their supply and have that supply be compared to other centers for prioritization. The center with a lower supply will be prioritized.
3.1	Public Database	Donation center priority detection	TC_3.1	The requirement is met when necessary data can be stored and pulled by public sources for use in the main software interface.
4	Personal information input	Input personal information	TC_4	The requirement is met when a registered user's data can be read and pulled to identify their information with little to no errors.
4.1 (2.1)	Refer to ID 2.1	Identify blood availability per location	TC_2.1	
5	Organize resources	Organize and Optimize blood resources	TC_5	The requirement is met when donation centers can store, pull, and organize resource information in a private database. This information will be made available to qualified users and administrators.
6	Resource request	Request resources	TC_6	The requirement is met when qualified users can send requests to donation centers for resource information. Perhaps to retrieve or collaborate on supply.

1.4 Bug Tracking

Bugs and enhancement requests are an important part of any software as it helps keep the quality, security, efficiency of the software at the upmost par. We will be using a bug tracking system that assigns a number to each bug/enhancement which is then briefly gone over to establish the severity of the bug. Then it will be organized by first come basis starting with the most severe as well. The severity will be determined by a simple 3 tier ranking system, 3 being most severe. Enhancements will also be given a number to which it will be dealt with in first come first serve as well.

1.5 Technology and Tools

Cardinal is a software that relies heavily on its included database, so technology used specifically for testing databases is a must. While developing Cardinal, MySQL will be used as the basis for creating the database.

Solarwinds offers a database performance monitoring tool for MySQL that allows users to monitor SQL queries and examine database performance metrics and analytics, making it a useful testing tool for this project. Updates to the database performance monitor (DPM) are made to be quick and efficient through the cloud, while keeping the entire program at a usage of under 1% of the computer's CPU. The DPM is also made to support a team, not just an individual user. Team members can efficiently share updates and even chat through the DPM software while using it, which it efficient, considering Cardinal's team works remotely. Solarwinds offers a 14-day free trial on their DPM, which will be a testing period long enough to carry Cardinal's team to the due date of the final product.

As Cardinal has not been made public for use, it would be difficult and time-consuming to create large amounts of database records without an automated tool. The DTM Data Generator, which also supports MySQL, is a tool that can generate random data for testing within a database. This software also offers a free demo period that can be utilized by Cardinal's team to develop the final product.

While the Cardinal's database is a core part of its development, there will be other components to the software that are just as essential for it to function. The code used in this project must be reviewed thoroughly to ensure the final product's quality. A great tool for code review is Crucible, a collaborative product developed by Atlassian. Not only does Crucible allow team members to review code, but it also helps to identify bugs and allows communication among team members within the program. Like the tools mentioned above, Crucible also offers a (30 day) free trial for Cardinal's team to utilize while developing the application.

2. Test Cases and Test Results

Test Case	Test Purpose	Test Steps	Expected Result	Actual Result	Pass/Fail Information
Request type: **Blood**	This test has a goal of	- Parametrizing	Center name:	Center's name:	**Pass**: as the expected result

Resources Inquiry Resource type: **Plasma** Resource amount: **15 bags** Maximum Radius: **10 miles** Requesting Client: **Greenwood Center**.	providing the best choice for the client center to make respond to its needs.	the search function with the given input. - Creating a subgraph of the centers that meet the criteria. - Running a graph search algorithm (BFS in our case for its completeness property). - Exhibiting the best-fit center information.	**Vital Blood Bank.** Requested Resource Balance: **21 Bags.** Radius: **7 miles**	**Vital Blood Bank.** Requested Resource Balance: **21 Bags.** Radius: **7 miles**	meets the actual one, this functionality has been partially verified.
Request type: **Blood Resources Inquiry** Resource type: **Platelets**	It is a second assessment to the main function the program performs, which is finding the	- Parametrizing the search function with the given input. - Creating a subgraph of the centers	Center Name: **P&R Blood Reserve**. Requested Resource Balance: **23 Bags.**	Center's name: **Vital Blood Bank.** Requested Resource Balance: **15 Bags.**	**Fail**: We have a mismatch of the actual result with the expected one. After additional debugging, we came to the conclusion that

Resource amount: **4 bags** Maximum Radius: **5 miles** Requesting Client: **Blood for Life**.	best-fit center.	that meet the criteria. - Running a graph search algorithm (BFS in our case for its completeness property). - Exhibiting the best-fit center information.	Radius: **4 miles**	Radius: **7 miles**	there must be an issue with the subgraph creating function.
Request type: **Suggestion's request** Resource type: **Plasma + Red Blood Cells** Requesting Client: **PlasmaX Center**. Number of centers: up to **5**	This test is to assess how effective is the suggestion system we built to print out the potential blood banks to make a request from.	- Setting a radius by default to 5 miles. - Creating a table that contains all the centers within that radius. - Scoring each row in the table using a coefficient that uses radius, days	Center Name: **P&R Blood Reserve**. Center Name: **Vita Center**. Center Name: **Sanguis Reserve**	Center Name: **P&R Blood Reserve**. Center Name: **Vita Center**. Center Name: **Sanguis Reserve**	**Pass**: as the expected result meets the actual one, this functionality has been partially verified.

		left to expire, amount of resource requested. - Exhibiting the best-fit centers' names.			
Request type: **Suggestion's request** Resource type: **Platelets** Requesting Client: **Vita Center**. Number of centers: up to 2	For further investigation of the suggestions system, we run another test with a different number of centers.	- Setting a radius by default to 5 miles. - Creating a table that contains all the centers within that radius. - Scoring each row in the table using a coefficient that uses radius, days left to expire, amount of resource requested.	NONE	Center Name: P**&R Reserve.**	**Fail**: While the program should have printed out NONE, given no center meets within the default radius. The bug could be originating from am implementation error in the scoring function.

		- Exhibiting the best-fit centers' names.			

Resources and References -

https://www.solarwinds.com/database-performance-monitor/integrations/mysql-monitoring

https://sqledit.com/dg/

https://www.atlassian.com/software/crucible