# **Introduction**

# **Discussion**

# **Test Cases**

# **Unit Testing**

(See Appendix A for test case tables)

# **Test Scenario 1**

This series of test cases was used to test the unit function that stored the Guest and Child Name of the Guest User. For this we used both Equivalence Class Partitioning and Boundary Value Analysis to test varying data lengths entered into the input boxes, which would eventually be sent to another function that stores the data into the system database. In our first two test cases, we used Equivalence Class Partitioning to verify what would occur if the Guest Name and the Child Name were less than one character and if they were one or more characters. Using the formulae below, we were able to derive two test cases that would satisfy the goals intended.

The goal for Test Scenario 1-1 was to confirm that the unit function would store both Guest Name and Child Name fields as long as they are one character or more. After entering a three character Guest Name and a four character Child Name, the function was allowed to execute successfully. As a result, our goals were met in this instance. The goal for Test Scenario 1-2 was to confirm that the unit function would not allow either Guest Name or Child Name to be stored as empty fields. After refusing to enter values for Guest Name and Child Name, the function prompts the user to enter a value for each field. Since the function never stores the Guest Name and Child Name unless values are written, then our goal was satisfied. Since these two test cases satisfied four conditions, less test cases needed to be executed, thus saving valuable testing time and allowing an increase in productivity.

*Let Guest Name = G and Child Name = C*

# Equivalence Class Partitioning (Test Scenario 1-1 to 1-2)

G = 0, C = 0 satisfies G < 1 and C < 1

G = 3, C = 3 satisfies G >= 1 and C >= 1

Afterwards, we used the formulas below that were derived using Boundary Value Analysis, to create five more test cases. Unlike Equivalence Class Partitioning, these test cases were able to test the range of valid amounts of entered data. In this scenario, the lowest valid value amount allowed is 1 and the highest allowed value amount is 20. The goal of Test Scenario 1-3 was to confirm that 10 characters is a valid amount for both Guest Name and Child Name. After using the test data from the table, the function executed successfully. Therefore, our goal was achieved. The goal of Test Scenario 1-4 was to confirm that 1 character is a valid amount for both Guest Name and Child Name. After using the test data from the table, the function executed successfully. Therefore, our goal was achieved. The goal of Test Scenario 1-5 was to confirm that two characters is a valid amount for both Guest Name and Child Name. After using the test data from the table, the function executed successfully. Therefore, our goal was achieved. The goal of Test Scenario 1-6 was to confirm that 20 characters is a valid amount for both Guest Name and Child Name. After using the test data from the table, the function executed successfully. Therefore, our goal was achieved. The goal of Test Scenario 1-7 was to confirm that 19 characters is a valid amount for both Guest Name and Child Name. After using the test data from the table, the function executed successfully. Therefore, our goal was achieved. These five test cases had high adequacy since they were able to satisfy ten Boundary Value Analysis conditions in just five test cases. This saved valuable testing time and allowing an increase in productivity.

# Boundary Value Analysis (Test Scenario 1-3 to 1-7)

G = 10, C = 10 satisfies nominal values for G and C

G = 1, C =1 satisfies minimum for G and C

G = 2, C = 2 satisfies just above minimum for G and C

G = 20, C = 20 satisfies maximum for G and C

G = 19, C = 19 satisfies just below maximum for G and C

# **Test Scenario 2**

This series of test cases was used to confirm that the unit function would only allow a successful login when receiving a valid Admin Name and Password. In these scenarios, Boundary Value Analysis would not be usable since we are dealing with Boolean expressions. Therefore, we used the Equivalence Class Partitioning formulae below to derive two test cases. The goal of Test Scenario 1-1 was to confirm that the unit function would allow successful access after receiving a valid Admin Name and Password from the Access Control List. After the test data was entered, the function successfully executed to completion, allowing us to navigate to the Admin homepage. Therefore, our goal was achieved. The goal of Test Scenario 1-2 was to confirm that the unit function would not allow successful access after receiving an invalid Admin Name and Password from the Access Control List. After the test data was entered, the function prompted for us to enter valid Login data. Since we were unable to enter the website without valid Login data, our goal was achieved. These test cases proved very adequate since they allowed us to achieve 4 conditional goals in just two test cases. This allowed us to reduce testing time, thus improving productivity.

*Let A = Admin Name and P = Password*

# Equivalence Class Partitioning

A = Valid, P = Valid satisfies A = Valid and P = Valid

A = Invalid, P = Invalid satisfies A = Invalid and P = Invalid

# **Test Scenario 3**

This series of test cases was used to confirm that the unit function would only allow a successful login when receiving a valid Guest ID and Password. In these scenarios, Boundary Value Analysis would not be usable since we are dealing with Boolean expressions. Therefore, we used the Equivalence Class Partitioning formulae below to derive two test cases. The goal of Test Scenario 1-1 was to confirm that the unit function would allow successful access after receiving a valid Guest ID and Password from the Access Control List. After the test data was entered, the function successfully executed to completion, allowing us to navigate to the next page, where the user would enter their Guest Name and Child Name. Therefore, our goal was achieved. The goal of Test Scenario 1-2 was to confirm that the unit function would not allow successful access after receiving an invalid Guest ID and Password from the Access Control List. After the test data was entered, the function prompted for us to enter valid Login data. Since we were unable to navigate to the next page without valid Login data, our goal was achieved. These test cases proved very adequate since they allowed us to achieve 4 conditional goals in just two test cases. This allowed us to reduce testing time, thus improving productivity.

*Let G= Guess ID and P = Password*

# Equivalence Class Partitioning

G = Valid, P = Valid satisfies G = Valid and P = Valid

G = Invalid, P = Invalid satisfies G = Invalid and P = Invalid

# **Test Scenario 4**

This series of test cases was used to confirm that the unit function is able to accurate determine whether a MySQL database was created or not. In the case where a database was not created, the function would create one. In these scenarios, Boundary Value Analysis would not be usable since we are dealing with Boolean expressions. Therefore, we used the Equivalence Class Partitioning formulae below to derive two test cases. The goal of Test Scenario 1-1 was to confirm that the unit function would recognize that a database was already created. Since the function executed without displaying a message, then a database was already created. Therefore, our goal was achieved. The goal of Test Scenario 1-2 was to confirm that the unit function would recognize that a database was not created and therefore create one. Since the function notified us that a database was created, then a database was successfully created. Therefore, our goal was achieved. These test cases proved very adequate since they allowed us to have simple and accurate tests that can be executed in reasonable time. This allowed us to reduce testing time, thus improving productivity.

*Let D = Database*

# Equivalence Class Partitioning

D = Created

D = Not Created

# **Test Scenario 5**

Let A = Access Control List and L = User Logs Table

# Equivalence Class Partitioning

A = Created, L = Created satisfies A = Created and L = Created

G = Not Created, L=Not Created satisfies A = Not Created and L = Not Created

# **Test Scenario 6**

Let F = FAQ Table

# Equivalence Class Partitioning

F = Created

F = Not Created

# **Integration Testing**

(See Appendix B for tables)

# **Alpha Testing**

(See Appendix C for tables)

# Decision Table

|  |  |  |  |
| --- | --- | --- | --- |
| Admin Name Entered | Y | - | N |
| Password Entered | Y | N | - |
| User Log Updated | X |  |  |
| Can Update FAQ Table | X |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Guest Name Entered | Y | Y | - |
| Password Entered | Y | Y | N |
| Guest/Child Name Entered | Y | N | N |
| User Log Updated | X | X |  |

# **3. Rapid Application Development (RAD)**

Our team chose to use the Rapid Application Development model to tackle the requirements of this Assignment. This model was ideal since it allows for rapid development of a smaller project like ours, whereas the Spiral Process Model would have been more ideal for a larger project which involved continuous enhancements. RAD allowed us to have more flexibility during our web application design and implementation. For example, we were able to build a rapid prototype of the web application, however quick changes were needed consistently, to allow the application to continuously adapt to the requirements of the assignment. Since our team was made up of a mix of developers of varying experience, this agile method was perfect to allow each other to compliment the lacking experience in each area, whether that being during design and implementation of the website, or during the testing and documentation phase.

Since we needed to have this project completed in a short period of time, RAD was the perfect choice. Without the restrictive spiral model, we were able to have a working prototype of the web application completed within a few days. Since the requirements allowed for flexibility in the design and look of the application, we were able to continually implement more user friendliness to our web application with each update. It was very important for us to have an application that was adaptable to changes. The RAD model usually has less risk associated with it than the Spiral Model, and considering that this is an application to be used in the Health industry, low risk web application development was a must.

One of the other reasons we chose to use RAD, was because it allowed us to quickly have a visual aspect of the requirements needed. This visual representation allowed us to identify the flaws easier, resulting in less flaws that needed to be addressed with each porotype cycle.

# **4. Rapid Application Development Phases**

# **Requirements Planning (Definition Stage)**

First we analysed the requirements and was able to development a business model for this project. We identified what was the most vital and necessary information needed and made sure that a heavy focus was made. For instance, in an Accident & Emergency, the most important information to a patient would be how and when they would be cared for. We realized that it had to be made easily apparent to any guest using the website, and also for any authorized staff that needed to update that information.

We also continued to analyse the other requirements, making sure that they were clearly defined and understood amongst our team. It was vitally important that the goals were clearly evaluated, in order to have the most correct build of the first stage prototype. Although a prototype can be altered later, with Rapid Application Development, speed and accuracy are the defining attributes. The less errors that are apparent in the first stage of the prototype, the less time would be needed to have to fix them before addressing the actual, correct requirements.

# **User Design (Design Stage)**

During this stage is when we got the opportunity to take the defined requirements, and turn them into a visual display. Since realistically we were not able to meet with clients for this project, we all at some stage took on the roll as a client by sharing ideas with one idea and continuing to consistently improve the design. Our main focus was usability, mobile responsiveness and accuracy to requirements. It was at this stage that we continually tested and tweaked each design until we were able to settle on a design that best fit the requirements of the assignment. At the end, we believe that we came to the more satisfying design for this project.

# **Rapid Construction (Development Stage)**

During this stage we took the web application prototype and transformed it into a working model. Although at this stage changes could still be made, due to the iterative design phase, we were able to quickly and smoothly finalize the development of this application. It was during this phase that we were able to conduct unit testing of each function, then integration testing of each functionality. Finally, we were able to conduct alpha testing of the entire web application.

# **Implementation (Deployment Stage)**

At this stage the final tests were done to confirm full functionality before finally launching the web application fully. Upon completion, we had a functioning application that could be used with node.js to satisfy the requirements outlined in the assignment.

# **5. Maintenance**

One of the main concerns of the development team after the application is fully launched is the subject of maintaining the application. Maintenance, as it applies to this specific instance, would be modifications and updates applied to the application to keep it up-to-date and tuned to the ever-changing world of technology and the environment it is operating in, this case being a web environment. In other words, we mainly concerning ourselves with Adaptive Maintenance.

The development team had multiple discussions as to why an Adaptative approach would be best and our reasoning as to the importance of Adaptive Maintenance is as follows: The software ecosystem, especially those concerning web applications is in a constant state of flux. Regular maintenance is required to adapt to these never-ending changes. Whether it changes in work patterns of staff, software platforms, compilers, updating the hardware (servers, desktop machines, etc), they all affect the functionality of the application. To ensure that the maintenance goes as smoothly as possible, we intend to do the following:

1. Study the current landscape in which the application operations for any sign of changes.
2. Analyze any changes to obtain a thorough understanding of their direction and timeline.
3. Deduce how our customer base will be affected at predictable points in the future or if they are being affected now.**]**

The most likely cause of maintenance would be updates to the web browser the application operates in. For example, say that our team discovers that the latest version of Google Chrome, Firefox or Safari released in developer-only beta is/will be incompatible with the current build of the application. Looking at the trend of releases in the past, we will know that within the span of a few months since the beta's release, our application will be incompatible with the latest versions of these web browsers when they go live. Thus we can work on updating our website to work with these new versions instead of waiting until they are released, customers run into issues, they complain and we have to spend extra time and resources performing Corrective Maintenance to develop a patch to solve these issues.

# **Conclusions**

# **References**

[1] The Industrial Advisory Board of the Guide to the Software Engineering Body of Knowledge (SWEBOK) - Institute of Electrical and Electronics Engineers, Inc. *Software*

*Engineering Body of Knowledge (SWEBOK)*, 2001. www.swebok.org - current February 2006.

[2] Adrion, W.R. , Branstad, M.A. , Cherniavsky J.C. *Validation, Verification, and Testing of Computer Software*. ACM Computing Surveys (CSUR), 14 (2), 1982. ACM Press.

[3] “4 Phases of Rapid Application Development Methodology,” *Lucidchart Blog*, 10-Aug-2018. [Online]. Available: https://www.lucidchart.com/blog/rapid-application-development-methodology. [Accessed: 21-Apr-2020].

[4] “SDLC - RAD Model,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc\_rad\_model.htm. [Accessed: 21-Apr-2020].

[5]

# **Appendix A: Unit Test Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 1-1 | Check if Guest Name and Child Name >= one character | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “Sam”  Child Name = “Gary” | User would successfully enter Guest homepage | User successfully entered Guest homepage |
| 1-2 | Check if Guest Name and Child Name < one character | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “ ”  Child Name = “ ” | User would never leave the current security page as the input boxes prompt for at least one character to be entered in each input field box | The input boxes await for at least one character to be entered in each input field box |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 1-3 | Check if Guest Name and Child Name = ten characters | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “Clementine”  Child Name = “Joan White” | User would successfully enter Guest homepage | User successfully entered Guest homepage |
| 1-4 | Check if Guest Name and Child Name = one character | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “S”  Child Name = “G” | User would successfully enter Guest homepage | User successfully entered Guest homepage |
| 1-5 | Check if Guest Name and Child Name = 2 characters | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “Sa”  Child Name = “Ga” | User would successfully enter Guest homepage | User successfully entered Guest homepage |
| 1-6 | Check if Guest Name and Child Name = 20 characters | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “Samantha Eve Jackson”  Child Name = “Chrissio Eve Jackson” | User would successfully enter Guest homepage | User successfully entered Guest homepage |
| 1-7 | Check if Guest Name and Child Name = 19 characters | User must have successfully entered their Guest ID and password | 1. Enter test data for Guest Name 2. Enter test data for Child Name 3. Await results | Guest Name = “Sam Annette Jackson”  Child Name = “Chrissy Eve Jackson” | User would successfully enter Guest homepage | User successfully entered Guest homepage |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 2-1 | Test when Admin Name and Password are both valid | The user must have Node.js running and have selected to login as an admin | 1. Enter Admin Name 2. Enter Password 3. Await Response | Admin Name = “Jake”  Password = “entering” | Admin User should have a successful login | Admin User had a successful login |
| 2-2 | Test when Admin Name and Password are both invalid | The user must have Node.js running and have selected to login as an admin | 1. Enter Admin Name 2. Enter Password 3. Await Response | Admin Name = “Karl”  Password = “lake” | Admin User would be notified of a username or password error | Admin User would be notified of a username or password error |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 3-1 | Test when Guest ID and Password are both valid | The user must have Node.js running and have selected to login as a guest | 1. Enter Guest ID 2. Enter Password 3. Await Response | Guest ID = “Jake”  Password = “entering” | Guest User should be directed to security page where they would enter their Guest Name and Child Name | Guest User was directed to security page where they would enter their Guest Name and Child Name |
| 3-2 | Test when Guest ID and Password are both invalid valid | The user must have Node.js running and have selected to login as a guest | 1. Enter Guest ID 2. Enter Password 3. Await Response | Guest ID = “Ruth”  Password = “klare” | Admin User would be notified of a username or password error | Admin User would be notified of a username or password error |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 4-1 | Test when Database exists | The user must have Node.js running | 1. Attempt to access Database 2. Await response | A select query to the Database | The Database is now available for use | The Database is now available for use |
| 4-2 | Test when Database does not exist | The user must have Node.js running | 1. Attempt to access Database 2. Await response | A select query to the Database | The database is created | The database is created |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 5-1 | Test when Access Control List and User Log Tables exist | The user must have Node.js running | 1. Attempt to access Control List 2. Await response | A select query to the Access Control List table | The Access Control List table is displayed. Once the Access Control List exists, then the User Logs Table also exists | The Access Control List table is displayed. Once the Access Control List exists, then the User Logs Table also exists |
| 5-2 | Test when Access Control List and User Log Tables do not exist | The user must have Node.js running | 1. Attempt to access Control List 2. Await response | A select query to the Access Control List table | Tables are created for the Access Control List and the User Logs | Tables are created for the Access Control List and the User Logs |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 6-1 | Test when FAQ table exists | The user must have Node.js running | 1. Attempt to access FAQ table 2. Await response | A select query to the FAQ table | The FAQ table is available for updating | The FAQ table is available for updating |
| 6-2 | Test when FAQ tables does not exist | The user must have Node.js running | 1. Attempt to access FAQ table 2. Await response | A select query to the FAQ table | The FAQ table is created | The FAQ table is created |

# **Appendix B: Integration Test Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 1-1 | Testing the integration between the Database Check Unit and the User Access List/User Log Unit during a successful Admin login. | The user must have Node.js running and have selected to log in as an Admin | 1. Enter Admin Name 2. Enter Password 3. Click “Log In” 4. Await confirmation for the correct comparison of log in details to the valid log in data stored in the Access Control List table | Admin Name = “Jake”  Password =”entering” | The Admin User should successfully be taken to the Admin homepage and the User Logs Table updated with the Admin Name and timestamp. | The Admin User should successfully be taken to the Admin homepage and the User Logs updated with the Admin Name and timestamp. |
| 1-2 | Testing the integration between the Database Check Unit and the User Access List/User Log Unit during an unsuccessful Admin login. | The user must have Node.js running and have selected to log in as an Admin | 1. Enter Admin Name 2. Enter Password 3. Click “Log In” 4. Await confirmation for the correct comparison of log in details to the valid log in data stored in the Access Control List table | Admin Name = “Jan”  Password =”kong” | The Admin User would receive an error that they have entered incorrect log in details. | The Admin User received an error that they have entered incorrect log in details. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 2-1 | Testing the integration between the Database Check Unit, the Guest/Child Name storing Unit and the User Access List/User Log Unit during a successful Guest login. | The user must have Node.js running and have selected to log in as a Guest | 1. Enter Guest ID 2. Enter Password 3. Click “Log In” 4. Await confirmation for the correct comparison of log in details to the valid log in data stored in the Access Control List table 5. Upon successful comparison, then enter both the Guest Name and Child Name 6. Click “Submit” and await response | Guest ID = “Jake”  Password =”entering” | The Guest User should successfully be taken to the next page after entering Guest ID and password. Upon successful log in, their ID and timestamp will be sent to the User Logs Table. On the next page, the user will be taken to the Guest homepage after successfully entering a valid Guest Name and Child Name. The Guest Name will also be sent to the User Logs Table | The Guest User was successfully taken to the next page after entering Guest ID and password. Upon successful log in, their ID and timestamp were sent to the User Logs Table. On the next page, the user was eventually taken to the Guest homepage after successfully entering a valid Guest Name and Child Name. The Guest Name was also sent to the User Logs Table |
| 2-2 | Testing the integration between the Database Check Unit, the Guest/Child Name storing Unit and the User Access List/User Log Unit during an unsuccessful Guest login. | The user must have Node.js running and have selected to log in as a Guest | 1. Enter Admin Name 2. Enter Password 3. Click “Log In” 4. Await confirmation for the correct comparison of log in details to the valid log in data stored in the Access Control List table | Admin Name = “Jan”  Password =”kong” | The Guest User would receive an error that they have entered incorrect log in details. | The Guest User received an error that they have entered incorrect log in details. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 3-1 | Testing the integration when the FAQ table is checked, updated and data is displayed on the page. | The user must have Node.js running and have selected to log in as an Admin. They must currently be on the Admin FAQ page | 1. Enter FAQ Question 2. Enter FAQ Answer 3. Click “Submit” 4. Await response | FAQ Question = “How many doctors does ServeYou have?”  FAQ Answer=”Enough to satisfy everyone” | The function should check for the availability of the FAQ table. After verificaiton, the FAQ Question and Answer are added to the FAQ table. The new FAQ information is then displayed on both the Admin and Guest FAQ page | The function should checked for the availability of the FAQ table. Upon verification, the FAQ Question and Answer were added to the FAQ table. The new FAQ information was then displayed on both the Admin and Guest FAQ page |
| 3-2 | Testing the integration when the FAQ table needs to be created before being updated and data is displayed on the page. | The user must have Node.js running and have selected to log in as an Admin. They must currently be on the Admin FAQ page | 1. Enter FAQ Question 2. Enter FAQ Answer 3. Click “Submit”   Await response | FAQ Question = “How many doctors does ServeYou have?”  FAQ Answer=”Enough to satisfy everyone” | The function should check for the availability of the FAQ table. Since there will be none, one will be created. The FAQ Question and Answer are added to the FAQ table. The new FAQ information is then displayed on both the Admin and Guest FAQ page | The function should checked for the availability of the FAQ table. Upon creation, the FAQ Question and Answer were added to the FAQ table. The new FAQ information was then displayed on both the Admin and Guest FAQ page |

# **Appendix C: Alpha Test Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario | Test Case | Preconditions | Test Step | Test Data | Expected Results | Actual Results |
| 1 | Check that the program executes to completion successfully | The user must have Node.js and apache running | 1. Login as a Guest using a valid Guest ID, password, Guest Name and Child Name 2. Ensure that the User Logs table was created and updated correctly 3. Ensure correct navigation to each page on the web application 4. When currently on the FAQ page, ensure that it is not possible to update the page using guest privileges 5. Logout 6. Repeat Steps 1-5 nine times 7. Check that the User Logs table has all 10 guests represented 8. Login as an using a valid Admin Name and password 9. Ensure that the User Logs table was created and updated correctly 10. Ensure correct navigation to each page on the web application 11. When currently on the FAQ page, ensure that it is possible to update the page using admin privileges 12. Enter an appropriate Question and Answer 13. Click ”Submit” 14. Ensure that the FAQ table was created and updated correctly 15. Ensure that the Question and Answer can be seen on the FAQ page 16. Logout 17. Repeated steps 8-16 nine more times 18. Check that the User Logs table has all 10 staff members (admins) represented 19. Check that the FAQ table has all Questions and Answers represented 20. Simulate process for 6 months and store data to a file 21. Ensure that the file has the correct information | GUEST DATA   1. Guest ID: 789; Password: password; Guest Name: Carl; Child Name: Kim   ADMIN DATA   1. Admin Name: Jake, Password: password | The User Logs table should have all 20 (10 Guest and 10 Admin) Login information stored with timestamps  The FAQ datbase table should have all 10 Questions and Answers stored  The FAQ page should have all 10 Questions and Answers shown | The User Logs table had all 20 (10 Guest and 10 Admin) Login information stored with timestamps  The FAQ database table had all 10 Questions and Answers stored  The FAQ page had all 10 Questions and Answers shown |

# **Appendix D: Web Application Documentation**

This web application can be used by visitors to allow themselves to become more familiar with the Accident and Emergency Department. This application can also be used by authorized staff to update specific sections like the Frequently Asked Questions area. This web application was created using HTML, CSS, Bootstrap and Javascript/Node.js. Various features from Font Awesome and Google Fonts were also used. Bootstrap was the front-end framework of choice, since it design for responsiveness to varying screen sizes like mobiles, tablets and Desktops/Laptops. Node.js was the ideal choice used to enable back-end functionality like Login Authorization using the Access Control List and storing the record of each log in. This application was designed to be user friendly and functional for the requirements.

When a visitor starts the application, they are given the option to log in using a guest ID and password. In the case of an incorrect log in information being entered, the visitor will be notified and inferred to enter the correct log in details. Upon successful log in, they would be prompted to enter their name and the name of their child. The visitor will not be allowed to leave these fields blank. Upon successful submission of this information, a record of their name of time of logging on would be sent to a file on a server and the visitor is then taken to the homepage. Depending on what device they are currently using, the web application will have a different appearance. At this point they can view the information displayed on the homepage or navigate to another section. At any point, the user will be allowed to log out using the navigation bar.

Upon view of the Wards page, they will be shown varying procedures and rules of the department and information about the ward they would be staying in, if admitted to the hospital. Upon view of the Frequently Asked Questions page, the visitor can view all questions and answers pertaining to the department.

When an authorized staff member starts the application, they will be given the option to log in using the administrative page. They would also be notified of any incorrect details entered. Upon successful log in, a record of their name and time of logging in would be sent to a file on the server. The authorized staff member would be able to view the same responsive pages as the visitor, along with having similar features. The one difference would be the view of the Frequently Asked Questions page. On this page, the staff member would be allowed to update the page by entering new Questions and Answers. Upon successful submission of this information, the page would be updated for both authorized and guest users.