**Executive Summary (5):**

**Summarizes the scope, solution and evaluation of the project**

A software suite is designed to allow Maintenance Instructions (MIs) to be submitted and stored in a digital format. The suite must allow engineers to create MIs on their computer and technicians will use a tablet to fill out the MIs in the field.

Our solution is a combination of an iPad for technicians, a web application for engineers and a NoSQL based database to store data. All components are connected through the use of a backend called NodeJS, which handles transmission of data.

The resulting solution was flexible and adequate to meet the customer’s request of a proof of concept. In order to ease burden of development after the project is passed off to the customer, switching to a SQL based database, adding SAP and Excel integration, as well as utilizing a more robust web framework to handle data is recommended.

**Scope & Purpose (5):**

**Statement of final RCGs with justification of any changes as compared to the proposal. Inclusion of system specifications / operating limits / applications of the product.**

The final requirements stemmed from 6 main use cases agreed upon by our team and the client. They are:

1. A technician can complete an MI on the tablet

2. An engineer can create a new template

3. An engineer can modify an existing template

4. An engineer can search for existing MIs based on inputted criteria

5. Engineers are alerted when MIs are submitted with problematic values

6. Foremen are able to review MIs before approving them and uploading to server

For more details on our use cases, please see Appendix B.

From these, we expanded on our RCGs to fulfill these use cases. There were only updates to the requirements and constraints from our initial RCGs in the proposal. Our requirements list grew for multiple reasons:

1. Increased information gathered from customer

2. Better understanding of the scope of the project

3. Better understanding of functionality required from initial proposal to now

4. Requests and renegotiation of requirements from customer

The following list is a brief overview of our requirements. For a full list of functional requirements, please see Appendix C.

**A) iPad requirements**

1. User is able to login using their credentials
2. Able to pull a list of MI templates from server and display them
3. User is able to select an MI from the available templates and open it
4. The application displays each question on an MI on its own screen
5. Each question in the MI has an extra comment box, unique to that question
6. Each question has a media attachment button
7. For each question, you can add photos, videos, and sound files

**B)Web App requirements**

1. Be able to pull the list of all question type objects from database.
2. Allow users to create MI templates by selecting different question types.
3. Be able to fill up with question contents on each question added.
4. Be able to submit MI template newly created by users to server.
5. View previously created MI templates
6. View completed MIs, including their attachments (Images only for now)
7. Search for an MI template by its title

**C)Api/Backend requirements**

1. Able to serve the main web page for the MI creation WebApp
2. Able to respond to requests from clients asking about the question types available.
3. Able to respond to requests from clients asking for the available MI templates.
4. Able to respond to requests from clients in order to submit an MI template.
5. Able to respond to requests from clients asking for the completed MIs.
6. Able to search the MiTemplates table given a title.
7. Able to respond to requests from clients in order to submit a completed MI.

We had one additional constraint added on to us, which was that EC2 (our testing server service, Amazon’s Elastic Cloud Computing) only had an uptime of 15 hours per day. We took this into consideration when looking for a server to test deployments on. Ultimately, EC2 was a the best option and it was free for the purposes of our project, as we did not need any paid features. The following list is the final list of constraints.

**Constraints**:

1. No guarantee of internet connection except when submitting

2. MI submissions may have to be stored locally for an extended period of time if an internet

connection is not available.

3. No guarantee that the most up to date version of an MI will be used when viewing MIs

4. EC2 has a daily limit of 15 hours of uptime

The following list of goals are the same as those in the proposal.

**Goals**

1. Engineers can set trigger limits on MI questions to send an alert if threshold exceeded in any MI submission. The alert is directed to the engineer who made the MI

2. When a user fills out an MI, a question can be flagged for further viewing by an engineer. This will alert the engineer who made the MI when it is submitted

3. Tablet app notifies user when a pending MI update or MI submission is occurring

4. MIs on the tablet are updated when an internet connection is available in the background

5. Tablet app can link to a printer to print the MI at any point

6. Engineers can export MI submissions to an excel format from the web application

**Architectural Design: System (25) : Overview of the prototype functionality and interdependencies among the subsystems as completed in the final prototype. Provide justification for the global design choices over other alternatives.**

Our final architecture is shown in figure 1 below. We used a backend to handle communication between all our components, ranging from database communication to serving and handling requests from clients (web application/iPad). We also used an HTTP web server to serve the web pages to our web application. It was important for us to abstract our server functionality which is why we opted for this model over others. The reason for this abstraction is because BC Hydro wanted to use their own servers when the project is handed off, but our team was not given access to their servers. As such, we had to make our server implementation as independent from server type/specialization as possible. The standard client-server model allows to easily separate any code that runs on the server from code running on other platforms.

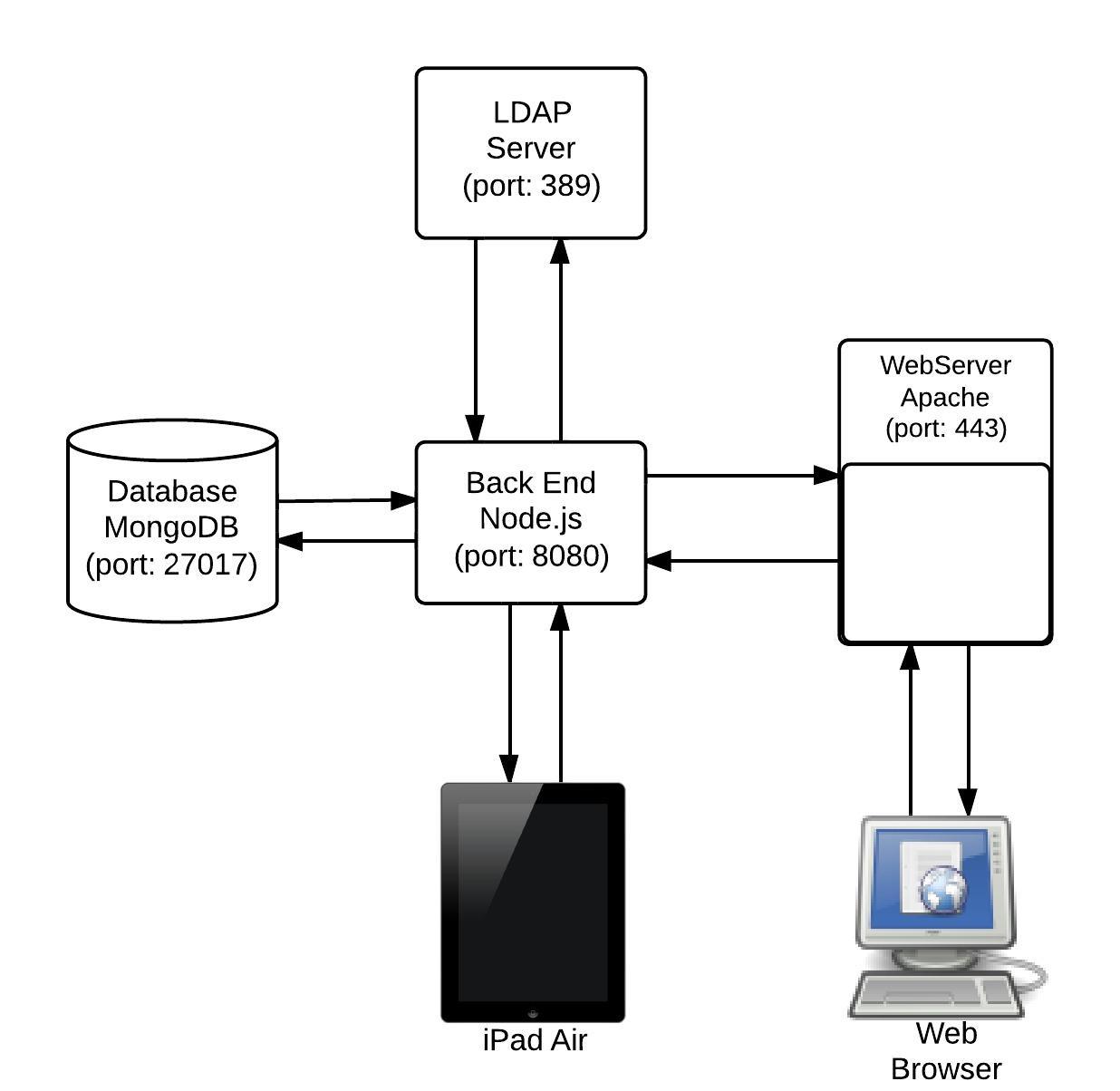


Figure 1: Architecture diagram for the Maintenance Instructions Application

**Architectural Design: Subsystem 25): Detailed description of subsystems and components as completed in the final prototype. Provide justification for design choices over other alternatives.**

**Database**:

The database we are using is MongoDB. It is a NoSQL type database based on JSON. The current instance of database is running on Red Hat Enterprise Linux (64-bit). Memory installed on the server is 0.615gb. Current hard drive capacity of the machine is 30gb. The database is accessible on the default port 27017 and admin panel is accessible on 28017. MongoDB version installed: 2.4.8

For more information on MongoDB, please see section 1 in Appendix A.

We had to make the decision of choosing between a SQL and NoSQL based database. A SQL database is much more popular and better documented and supported, but we made the choice of a NoSQL based database because it provided us with robust, built-in media management by allowing us to store media directly in the database. In addition, collections are used in place of table schema, meaning we had a lot more flexibility in changing how we stored our data, by not having to adhere to strict, predefined schemas. This allowed to store data pertaining to MIs in any form, in addition to the structure of our questions and internal data structures.

**Node.JS:**

We are using Node.JS to communicate with the database. It is a middle man that accepts requests from iPads and web app and retrieves or modifies data in the database. Our implementation of Node.JS is running on the same machine as the database. It is running on a Red Hat Enterprise Linux (64-bit) machine with 0.615 gb of ram and 30gb of hard drive. Node.JS listens on port 8080 for requests. The installed version of Node.JS: v0.10.21

For more information on Node.JS, please see section 2 in Appendix A

We had the choice of going down the more popular route and using a language such as PHP to handle our backend, but we realized that we were using JavaScript for our front end development, which would have meant that another programming language in addition to the paradigms associated with backend programming. By using Node.JS, we were able to leverage the single language, JavaScript, and the more modern features it provides, such as less context switching and better handling of 3rd party frameworks.

Another very important feature of Node.JS is the ability to integrate with MongoDB seamlessly. The integration is painless and supported out of the box, whereas PHP would require a 3rd party driver.

**LDAP Server:**

The LDAP server is currently performing authentication for the iPad. In the future, BC Hydro will use Active Directory on a Windows Server using Windows Server 2008 so our implementation has taken this into account and does not have any platform specific code. The hardware is will be deployed on a Intel 64 bit 2.33 GHz Core 2 Duo machine with 4GB of memory and 500GB of hard drive.

Any directory service can be used for authenticating users. Check chosen server operating system documentation for requirements to set up directory service.

Our decision to integrate LDAP authentication over something more modern like claims-based authentication was part of the requirements from BC Hydro. Because they already used LDAP internally, it would be much easier for them to integrate our applications into their ecosystem.

**Web Server:**

We are using Apache for our HTTP server. The version of Apache we have currently installed is 2.2.15 on a Red Hat Enterprise Linux (64-bit) machine with 0.615GB of ram and 30GB of hard drive. We use SSL on port 443 to encrypt traffic between client and server.

Either an Apache server on an unix machine or IIS on a windows server can be used.

For our purposes, an Apache server was selected simply out of convenience, as our EC2 server is unix-based. Had our server been Windows based, we would have likely used IIS instead.

**iPad:**

The app is currently supported on iOS 7. It is designed for an iPad with a 9.7 inch display and the 7.9 inch iPad mini. The iPad communicates with the Node.JS server on port 8080 and requires the internet to do so.

The decision to use an iPad over Android tablets came down to the fragmentation of Android tablets.With the iPad, the OS and hardware were unified and consistent meaning that BC Hydro could pick up the latest iPad that supports iOS 7 and they would all be the same, and would continue to be supported in the future. Android, on the other hand, have multiple versions from different vendors in terms of hardware. The OS on the device is based on the choice of the vendor, with more modern tablets not able to run the latest Android OS. For BC Hydro, this is a big issue as newer versions of a tablet OS can have fixes to serious security flaws.

**Web Browser (Client for web app):**

The client will access the web app using a web browser on their local machine. To access the web app, the client needs to be connected to the internet and they communicate securely using HTTPS on port 443. Currently the browsers supported are Chrome and Internet Explorer, as these are what BC Hydro use internally.

We had the option of creating a native desktop application instead of a web application. However, this would have meant using a different platform, either Java or C#, to create a native Windows application. Sticking to a JavaScript based web application meant we could leverage JavaScript in more of the project. Also, a web application is platform agnostic, meaning engineers could run it anywhere they could get an internet connection. This means engineers can access MIs if they are out of the office or on a different device. Ultimately we felt these two reasons made the web application more appealing over a native desktop application.

**Formal Testing (20): MOE**

**Description of how functionality was verified. Summary of test results which should match the previously stated system specifications.**

Formal testing and verification is vital to any software project. The goal of testing the BC hydro maintenance instruction application was to ensure the implementation satisfied the requirements. To test the application, we used black-box , manual and automated testing techniques to write test cases that satisfy the functional and nonfunctional requirements of the project.

**Black-box and manual testing**

Black-box testing is a method of software testing that examines the functionality of an application without looking into its internal structure. This testing method was relied upon heavily for this project and used to test the iPad app, Web app and the Node.js backend. To perform Black-box testing a set of documents were created to test each of the functional requirements of the project. Each requirement, had at least 1 test case that ensures it works properly. Figure 1 below illustrates the format of a black box test. The preconditions section lists the conditions that should hold true prior to conducting the test. The Expected Results section explains the result that the tester should see granted the test runs successfully. All the Black-box test cases are executed manually by testers and simulate behaviours of normal users, in this case, BcHydro Engineers. The tools used to perform testing included “RoboMongo” to verify the database contents; “Postman” to make requests to the Back end and verify the correct response. Finally, a web browser and its attached console are used to test the Web app.

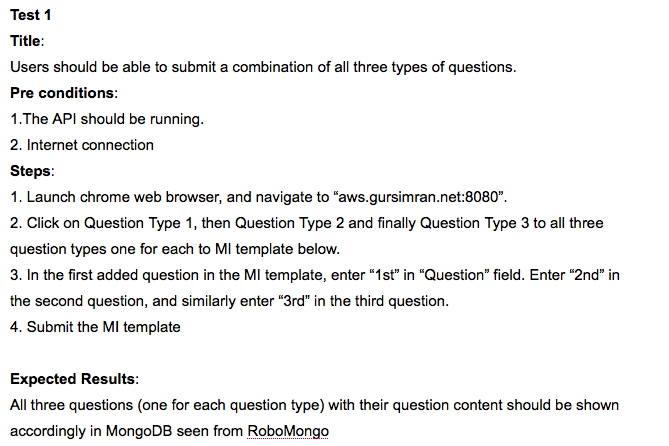


Figure 1: A sample Black-box test case.

**Automated testing**

Automated testing was used to test parts of the iPad application. An automated test suite was created using a tool called OC-unit, which is a tool that allows users to create automated tests for iOS applications. Figure 2 below is a sample automated test. The test shown below tests LDAP authentication on the iPad by supplying the username and password and checking for successful login.

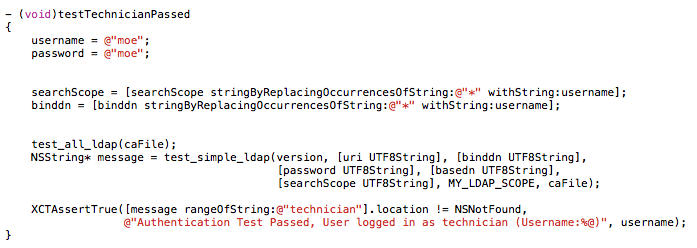


Figure 2: A sample automated test

**Summary of test results**

The complete test suite is maintained in the form of a github repository and new tests are added as new functional requirements are implemented. When new features are implemented, regression testing is performed and old tests are re-run to ensure that nothing was broken. As of now, all the test cases run successfully and return the expected results.

**Risk Management (10):**

**Summary of how risk to the project was mitigated through technical decisions, engineering processes and / or technical management**

Over this project, we encountered several risk that needed to be handled. The full list of risks can be found in the Risk Assessment Chart in Appendix D. The following will be a breakdown of our mitigation strategy by prototype.

Prototype 1:

For prototype 1, we identified 3 known risks that posed a high risk: iPad physically breaks, database security breach and database corrupted. We attempted to mitigate the first two risks, as the third risk was out of our hands.

To mitigate the first risk, we saved incomplete MIs to the database alongside completed MIs. This will still mean that any MIs in progress but not marked as incomplete will be deleted. In this way, even if the iPad breaks, the technician would be required to complete the MI from where he left off, not from the beginning (assuming he marked it as incomplete).

The second risk is a matter of security. While the matter needs more research and validation, we mitigated this risk by using the LDAP protocol for authentication (currently used by BC Hydro). General security of the server is not part of the scope for our project, and we use pre-existing communication security protocols to connect to our database.

Prototype 2:

For prototype 2, we had 3 new risks to mitigate. The first risk, with our client moving to a different location further away, we reduced the impact by more liberal use of emails as well as scheduling online meetings over Skype.

The second risk was our lack of familiarity with handling media attachments. We tried to reduce the impact by increasing the amount of time we spent researching media handling on the iPad, the Web App and our database. Unfortunately, this did add to the time taken to deal with tasks pertaining to this risk.

The final new risk was changing availability of team members. To reduce impact, we adjusted schedules and tasks given to members that would be unavailable to ensure that the most important work would be finished before the member(s) became unavailable. Also, we tried to keep everyone (including missing members) up to date with status updates and recorded meeting minutes.

Prototype 3:

For prototype 3, we had 2 risks to manage: new client requirements and large size of media attachments. We negotiated the first risk by reviewing the new requirements and picked the ones we could implement in time without too much effort. We then explained to the client that implementing the rest of the requirements would impede the overall progress of the project. We managed the second risk by compressing the larger files before they were sent through the iPad Application.

**Client Hand-Off Protocol (5):MOE**

**Description of how the client will use the final prototype. Summary of documents prepared for the client and the significance of the documents. The instructor will verify these items have been provided to the client.**

The project is divided into 3 major components, the iPad App, the Web App, and the supporting Back end. Three github repositories host the code for each of these components along with their list of bugs, issues and suggested enhancements. The client will be given these repositories along with 3 documents. The first document is a setup guide which explains how to install and setup the various tools needed to run the project. The second document is list of software artifacts which explains in detail the various components of the project, such as functional/non-functional requirements, use cases and risk management. The final document contains the test suite for the project.

**Conclusions (5):**

**Summary of what was achieved, outstanding goals and / or requirements, recommendations for further product development.**

In the last 7 months, our team has put together a software suite, consisting of a web application, an iPad application and a backend service which will allow BC Hydro to move their current maintenance instructions process from paper to digital. Our application also uses a NoSQL based database in order to handle all the data and media that our applications utilize. We also integrated LDAP authentication in order to match the authentication services in place at BC Hydro. Our codebase has been made as easy as possible for BC Hydro to integrate into their existing infrastructure.

We were able to satisfy all requirements agreed upon with the customer. There is only 1 goal which was not done; the ability to export data from our application to Excel. This goal has been part of our backlog and our client is aware that it will be a feature they will have to develop if they still want it. There are recommendations for the client from our team based on our experiences to this point:

1. Add an audit trail to keep track of who made changes. Our applications only track who made the latest changes.
2. Add SAP integration
3. Utilize a modern web framework to handle data management such as AngularJS or EmberJS
4. Consider bringing in a consulting firm to perform a heuristic evaluation
5. Incorporate technicians in the design process for the iPad application UX. Get their feedback on what works and what is confusing.
6. Consider adding more sophisticated security on top of what our applications have

As BC Hydro uses and better understands what they wish to get from this software suite, they will continue to add their own features, these are just some suggestions to get them started.

We, Group 5, have had a fortunate opportunity to work on a project so important to BC Hydro. We thank our client, Mattias Gibbs, and BC Hydro for the opportunity and hope our realization of their concept will bring continued success.

**Bibliography & Appendices : Detailed information needed to justify statements from other sections in the report. No explicit mark is given.**

**Appendix A - System Requirements**

**A1: MongoDB Requirements**

MongoDB Version:

* 64-Bit build to store than 2gb of data.

OS Requirements:

* 64-Bit OS
* Any of
  + Linux
  + Mac OS X
  + Windows Server 2008 R2 64bit
  + Windows 7, vista
  + Solaris platform

Hardware Requirements:

* Processor
  + 64-BIt (to run a 64-bit build)
  + Faster the better (there is no minimum)
* Memory
  + More the better (no minimum)

Accessibility:

* Database access - Port 27017 (can be changed to as required)
* Admin panel access - Port 28017 (1000 more than database access port)
* If running on a separate machine than node.js, make sure to disable internal firewall of linux machine that is enabled by default.
* If this machine is on a separate network, open the database access port for other machine to communicate.

**A2: Node.Js Requirements**

Node.js Version:

* Latest stable release (currently v0.10.21)

OS Requirements:

* 32/64-Bit OS
* Any of
  + Linux
  + Mac OS X
  + Windows

Hardware Requirements:

* Processor
  + 32/64-BIt
  + Faster the better (there is no minimum)
* Memory
  + More the better (no minimum)

Accessibility:

* Request access - Port 8080 (can be changed to as required)
* Request access port needs to be open in the firewall to allow external connections.
* If using linux, add a firewall rule to allow external connections.
* Talks to LDAP Server on port 389.

**Appendix B - Detailed Use Case List**

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 1  Mechanic/Electrician |
| Use Case Name: | **Completing MIs** |
| Description: | An electrician on site is doing a routine inspection of a piece of equipment on site. He is instructed by the foreman that he needs to fill out an MI. He grabs a tablet and opens the MI application. While still maintaining an internet connection, he logs in and is presented with the home screen for electricians. He types in the name of the equipment or the MI template id in the search bar and the relevant MI is highlighted, which he opens. After the relevant MI is downloaded, he heads down to the powerhouse (there is no internet connection in the powerhouse), he fills out the questions, but does not submit the MI. He logs out of the application, heads back to the office and returns the tablet to the foreman. |

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 2  Engineer |
| Use Case Name: | **Creating a New MI Template** |
| Description: | An engineer needs to create a new MI template for a new generator being installed. He goes to the web application and logs in. He then clicks the “new MI template builder” button and he then builds the new MI template from a given set of question types. After finishing it, he names it and saves it. When he clicks “submit”, the template is uploaded as a JSON object to the server . When tablets connect to the server the next time, this new template will be available. |

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 3  Engineer |
| Use Case Name: | **Modifying Existing MI Template** |
| Description: | An engineer wishes to modify an existing MI template to reflect recent policy changes. He logs into the web application, uses the “search existing MI templates” feature to select the template to be modified. If/When the template appears, he clicks “edit” and proceeds to make any required changes to the template. Once finished, he saves and clicks “submit” and a new version of the template is uploaded to the server, superseding the old version. The version number of the template is incremented. After this, the new MI template can be used to create MIs. |

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 4  Engineer |
| Use Case Name: | **Search Completed MIs by a given criteria** |
| Description: | An engineer wishes to check recently completed MIs on a piece of machinery located at a plant in Mission. He logs in to the web application and selects the search completed MIs option, then selects the plant name in the location box and the piece of machinery in the equipment box. He then clicks the “search” button and is presented with the results. |

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 5  Mechanic / Electrician |
| Use Case Name: | **Alert Engineer about problems with the MI** |
| Description: | The mechanic/electrician fills out an MI and notices something peculiar about one of the readings on the equipment. He flags the appropriate portion of the MI by checking a box that reads “alert” to let the engineer know about the issue. He completes the rest of the MI and hands it over to the foreman. The server notices an alert has been set in the MI and sends an email to the engineer notifying them about the issue. |

|  |  |
| --- | --- |
| Use Case Number:  Actor: | 6  Foreman |
| Use Case Name: | **Review MI and upload to server** |
| Description: | After the electrician/mechanic has completed (but not yet submitted) the MI on the tablet, he gives the tablet to the site foreman. The foreman logs in separately, searches for and reviews the completed yet unsubmitted MI, and while having an internet connection, submits the MI. Once he gets the confirmation of successful submission, he turns off the tablet and resumes his daily work. |

**Appendix C - Full Requirements List**

**A) iPad requirements**

1. User is able to login using their credentials
2. Able to pull a list of MI templates from server and display them
3. User is able to select an MI from the available templates and open it
4. The application displays each question on an MI on its own screen
5. Each question in the MI has an extra comment box, unique to that question
6. Each question has a media attachment button
7. For each question, you can add photos, videos, and sound files
8. Can take new photos and videos or select them from photo gallery
9. Can manage media for each individual question
10. Questions can be in the following format
    1. Short answer
    2. True/False
    3. Multiple Choice (single answer)
    4. Multiple Choice (multiple answers)
    5. Tables
11. Users are able to navigate to previous questions
12. The application displays an error attempting to move to the next question without answering the current question or entering a comment
13. All questions and answers are displayed at the end for review
14. While reviewing, you can click on a question to go to that question and change the answer
15. Thumbnails of attached media for each question is displayed when reviewing
16. On the review screen, you can click on the submit button to mark it as completed for the foreman to review
17. User can logout
18. Foreman can select a completed MI from the list of completed MIs
19. Foreman can review all the answers and also edit them
20. Foreman can add an additional comment to the form
21. The submission sends the MI to the server as a JSON object with media encoded as base64.
22. Incompleted forms are deleted after 7 days
23. Forms started by some user should not be displayed to another
24. User can save forms and complete them later
25. User can search forms on multiple fields
26. User can delete incomplete forms
27. User can print a completed MI on the review page by clicking on the print button.
28. Each question has a help button that has instructions on how to answer that question.
29. Foreman can assign a MI started by one technician to another technician.

**B)Web App requirements**

1. Be able to pull the list of all question type objects from database.
2. Allow users to create MI templates by selecting different question types.
   1. True/False
   2. Multiple Choice
   3. Short Answer
   4. Table
3. Be able to fill up with question contents on each question added.
4. Be able to submit MI template newly created by users to server.
5. View previously created MI templates
6. View completed MIs, including their attachments
7. Search for an MI template by:
   1. Title
   2. Upload Date
   3. Original Issue Date
   4. Edit Date
   5. MI Number
   6. Revision Number
   7. Discipline
   8. Location
   9. Equipment
8. Allow users to add section groupings to MI templates

**C)Api/Backend requirements**

1. Able to serve the main web page for the MI creation WebApp
2. Able to respond to requests from clients asking about the question types available.
3. Able to respond to requests from clients asking for the available MI templates.
4. Able to respond to requests from clients in order to submit an MI template.
5. Able to respond to requests from clients asking for the completed MIs.
6. Able to search the MiTemplates table given a title.
7. Able to respond to requests from clients in order to submit a completed MI.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Appendix D : Risk Assessment Chart**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Category | Risk | Occurrence Chance | Occurrence Severity | Risk Factor | Status | Notes |
| Product/ Service |  |  |  |  |  |  |
|  | Wireless internet down | Low | Low | Low | Mitigated | Cannot submit. Cannot access latest version of MIs |
|  | Local server down | Low | High | Medium | Closed | Cannot submit. Cannot create new templates. Engineers cannot view submissions. No notifications |
|  | Database security breach | Low | Very High | High | Mitigated | Sensitive data could be leaked. |
|  | Database corrupted | Low | Very High | High | Closed | Must rollback server. All data lost from last backup to moment corruption occurs |
|  | iPad breaks | Medium | High | High | Mitigated | iPad used in harsh conditions. Broken iPad means loss of data that has not been submitted |
| User |  |  |  |  |  |  |
|  | Saved MI not submitted within allotted time | Low | Medium | Low | Closed | Submission must be discarded. Need to re-submit MI |
|  | Incorrect values used in template | Low | High | Medium | Open | Incorrect values in template means all submissions made with template may become invalid. |
|  | Incorrect values entered in MI submission | Low | Low | Low | Mitigated | Submission is invalid. Must be re-done. |
| Technical |  |  |  |  |  |  |
|  | Data corrupted while being sent to server | Low | Medium | Low | Closed | Submission must be discarded. Need to re-submit MI |
|  | Server only receives partial data for submission | Low | Medium | Low | Closed | Either server discards submission or tries to request a re-submission from iPad |
|  | Large media attachments taking too long to transfer | Med | Med | Med | Mitigated | Files are compressed before they are attached and sent |
| Other |  |  |  |  |  |  |
|  | Mattias moved to a different campus | Occurred | Medium | Medium | Mitigated | Contact with our client has become more challenging. Decisions and reviews are being delayed |
|  | Unfamiliar with handling media attachments | Medium | Medium | Medium | Mitigated | Increased development time to handle media attachments |
|  | Teammate(s) unavailable | Low | Medium | Medium | Mitigated | Delays development and increased communication difficulty |
|  | Client sends new requirements | Low | Med | Med | Closed | Implemented requirements that required little effort |