

Computer Science and Engineering
University of Nevada, Reno

NRDC Quality Assurance Application

Team 9:

Brianna Blain-Castelli
Christopher Eichstedt
Matthew Johnson
Nicholas Jordy

Instructors:

Dr. Sergiu Dascalu
Devrin Lee

External Advisors:

Connor Scully-Allison
Vinh Le

11/19/18

1. Table of Contents

2.	Abstract	3
3.	Introduction	3
4.	High-level and Medium-level Design	4
5.	Detailed Design	12
6.	Initial hardware design	15
7.	User interface design	16
8.	Glossary	20
9.	Contribution of team members	21

2. Abstract

The NRDC QA Application is a cross-platform mobile and web application. It is being designed as a generalized, user-facing tool for recording metadata, but with a more specific goal of aiding NEXUS site technicians for better efficiency in the field. The current process has NEXUS site technicians physically recording data in notebooks before transferring it into a central database. This paper describes how such an application could be designed through detailed UML diagrams and descriptions of necessary classes and functions.

3. Introduction

The 2018-2019 Senior Project's Team 9 has begun development on a web-based Quality Assurance (QA) application for NEXUS affiliated technicians who visit remote research sites throughout Nevada. Currently, the only option available to technicians for recording QA information is using physical notebooks. Technicians are then required to manually enter data collected into a computer upon their return. This can lead to inaccurate data entry due to factors such as human or documentation error. This application will allow a technician to enter information regarding quality assurance into given data fields, regardless of a connection to the internet. When the application is connected it will synchronize the data recorded to a central server. This will make the entire process of recording data more efficient.

The application will use an ontology which acts as a graph data structure that tells the application how to format its data fields. When the technician enters information and the application is not connected, the data will be stored locally. If the user is connected, the application will then test for a network connection and provide a notification that it is ready to send information.

The application will be developed using the Ionic framework which is used for creating mobile applications. Ionic uses multiple programming languages including TypeScript, HTML, and CSS. The server will use a parser to load a given ontology, and the application will connect to a Flask Microservice which will provide it with the hierarchy necessary to generate the interface.

Progress made since specification report includes planning implementation details based upon high, medium, and detailed design requirements. Progress was also made on the back-end towards parsing the ontology for specific aspects that can then be used to generate a hierarchy for the front-end.

Significant changes or updates to the requirements since specification have included more specific implementation details such as utilizing SHA256 encryption for login, separating metadata and ontology servers, specifics on how the search and search results pages would be

handled, and adding extra functionality to a number of classes. A very important change was planning out the methods of creating a dynamically generated hierarchy page, rather than the assumptions of always having locations/sites.

4. High-level and Medium-level design

The NRDC QA Application is designed to communicate between a client and server. The QA Mobile Application will query JSON information from the server using the Ontology Manager. The Metadata Management Application handles the metadata directly, by submitting it to, and retrieving it from the database. Figure 1 illustrates this process in detail below.

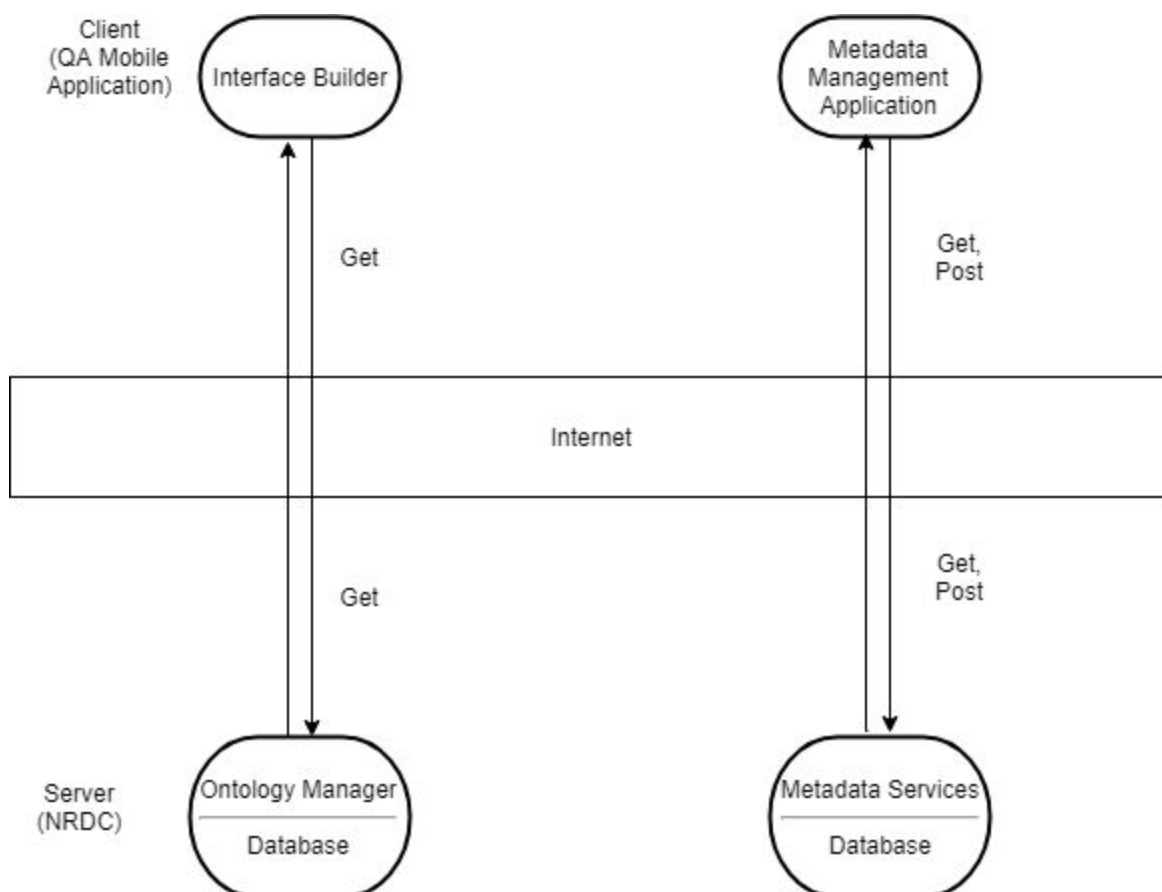


Fig 1. Client Server Block Diagram of the NRDC QA Application

The tables below contain the classes found within the NRDC QA Application. Each individual table contains functions that are described in greater detail as they pertain to the design.

Table 1. A description of the class “About”

Class: About	Description: An about page. It populates the necessary information about the development team and loaded project
getTeam()	This function populates the information regarding the development team.
getProject()	This function will populate information regarding the current project attached to the verified user.

Table 2. A description of the class “Home”

Class: Home	Description: The main page for the application. It displays notifications that updates have taken place and also displays the last site updated. It displays online status.
getNotifications()	This function will populate information on recent updates to the home page. It will display the number of updates that have taken place since the user last logged in.
getOnlineStatus()	This function will populate information on the current connection status. It will be denoted using a symbol for either “connected” or “disconnected.”
syncService()	A function called by a user pressing a synchronization button on the home screen. This function allows for the application to synchronize with the database
displayLoadingStatus()	This function visualizes the loading process on the home screen for the user by providing updates to the user on the loading process in the form of a progress bar with captions.
getOT()	Calls the flask service for a list of all organizational tiers and their parent-child relationships
getDataFields()	Calls the flask service for the characteristics of the tiers that will take the form of data fields in the application. This data is then stored in Characteristics.

Table 3. A description of the class “HierarchyManager”

Class: HierarchyManager	Description: Displays available hierarchical tiers according to the ontology assigned via login.
getUserNavigation()	Gathers hierarchical navigation data and populates the hierarchy manager page to display the appropriate level of navigation per user.
getLowerTiers()	This function gathers lower organizational tier information in order to structure pages following the broad location page.

Table 4. A description of the class “Search”

Class: Search	Description: Queries the application for appropriate search items defined by the user that can be keywords, locations or objects.
keywordSearch()	Takes the user submitted information and checks against database to populate interface with similar results.
locationSearch()	Similar to keywordSearch() but checks against the locations available to the user.

Table 5. A description of the class “SearchResults”

Class: SearchResults	Description: Populates a page based on items defined by the user that can be keywords, locations or objects.
returnKeywordResults()	Returns a populated list regarding information checked for when using the function keywordSearch() in the Search class.
returnLocationResults()	Returns a populated list regarding information checked for when using the function locationSearch() in the Search class.

Table 6. A description of the class “Login”

Class: Login	Description: Authenticates user login information and password to properly display information based on user credentials.
checkUser()	This function will check against the user’s login for validity.
checkPassword()	This function will check against the user’s password for validity.
encryptData()	This function will take in a string and encrypt it to using SHA256 encryption to ensure secure data transmission.

Table 7. A description of the class “Updates”

Class: Updates	Description: A page that displays most recent update logs and allows the user to check for updates.
checkOntologyUpdates()	Performs a checksum against the ontology to check for changes and modify information stored within the application accordingly.
checkVersionUpdates()	Performs a version check to search for new iterations of the application.
displayUpdates()	Display a list of information below the update option with changes made following the updates that took place. In the event no updates took place, an option stating no updates available is displayed instead.

Table 8. A description of the class “Settings”

Class: Settings	Description: A page that displays, holds and allows for the configuration of user settings, including accessibility, display, and more.
getUserSettings()	This function populates the user settings page with the current user settings.
updatePreferences()	This function updates the interface with user defined settings.

Table 9. A description of the class “DataValidation”

Class: DataValidation	Description: A class to hold validated data on the application. This class allows for validated user input to be saved if offline, and allows the user to send data when online.
validataData()	Checks the user entered information against the expected data types in the class Characteristics. This function copies over the data and deletes the information saved in Characteristics in order to allow the user to save validated data.
displayValidationErrors()	Returns which data was copied over successfully and which was not validated due to errors in the data. This information is populated on the page by highlighting successful fields in green and clearing them while highlighting unsuccessfully validated fields in red and keeping their information present in the field.
sendData()	If the user is online, sends the data to JsonConversion to send data to the database. If the user is offline, keeps the user entered data local.

Table 10. A description of the class “Parser”

Class: Parser	Description: Parses the ontology in order to determine data fields and hierarchical navigation. This class converts the parser results to JSON from XML and organizes this data to send to the interface.
convertOntology()	Converts the ontology XML file to JSON while parsing the data. The data is then stored in a graph to be organized.
sortOntology()	Sorts the ontology information into fields for easier reading. This includes gathering all organizational tiers, and linking characteristics to objects in the ontology such as sites or sensors.

Table 11. A description of the class “JsonConversion”

Class: JsonConversion	Description: A class to handle conversions of data types.
convertGPS()	Converts GPS coordinates to a format accepted by Json. This may include strings or integers.
convertImage()	Converts images to a format accepted by Json. This may include strings or integers.
convertTimeSeriesFromJson()	Converts a string provided in a query to date/time form for the application’s use
convertTimeSeriesToJson()	Converts time/date to a string accepted by Json.

Table 12. A description of the class “Characteristics”

Class: Characteristics	Description: A class to hold data fields as wells as data entries. This class organizes ontology information for the front end. It works in tandem with the DataValidation class.
gatherInput()	This function will accept user input and organize the data for later validation according to the corresponding data field.
generateFields()	This function will generate form fields based on the ontology assigned characteristics.

Figure 2 is a class diagram and describes the relationship that each class has with one another. In addition to the functions, the class diagram also displays the variables used by each class. The various diamonds and arrows notate the relationships the classes have with one another. The Parser class is not related to any other class because it derives from the back end while the rest of the classes derive from the front end.

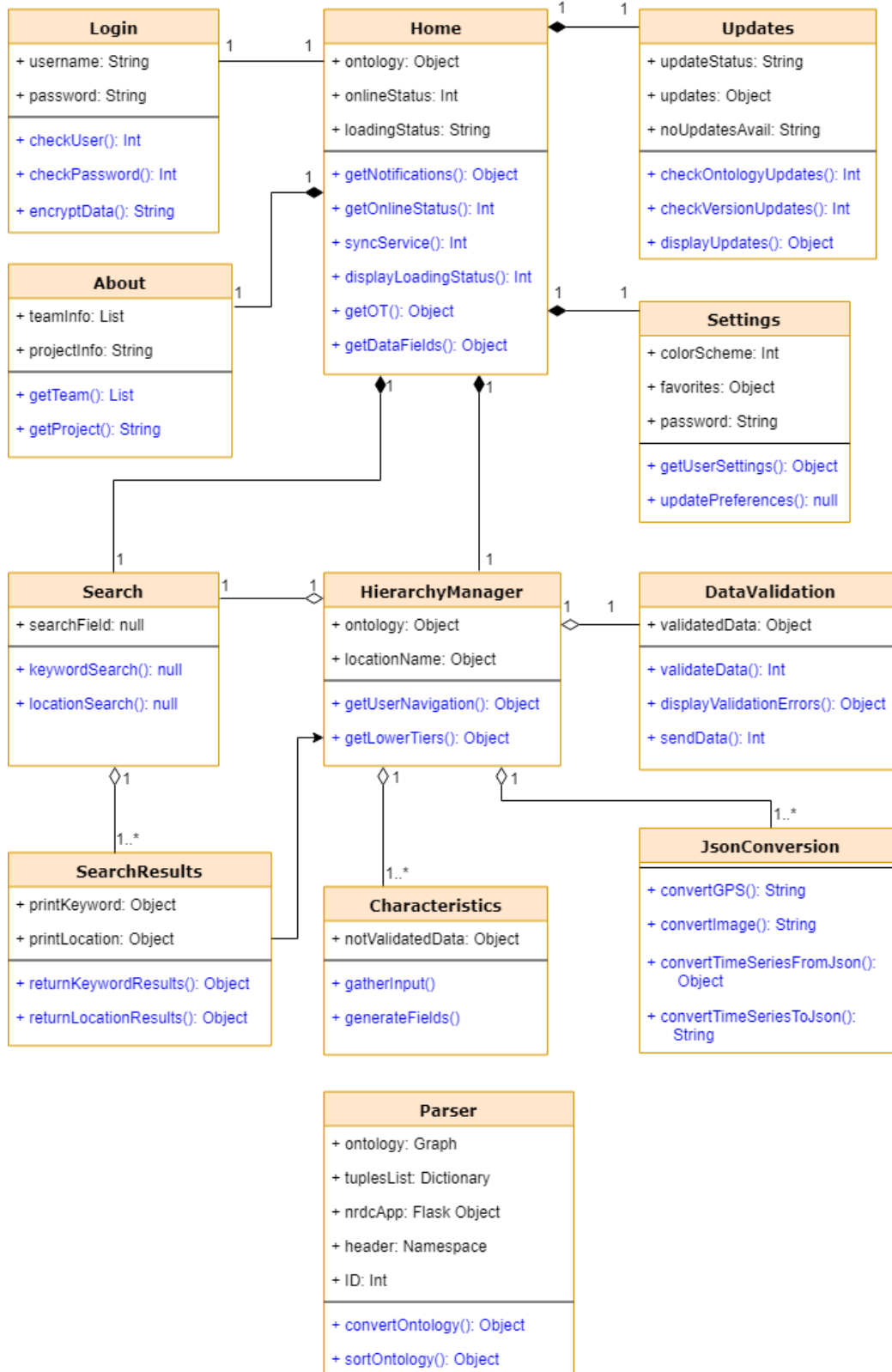


Fig 2. Class Diagram showing the relationship between critical classes in the application.

The main data structures utilized in the application's architecture include graphs, trees and a database. After parsing the ontology, the data gathered is stored within a graph. This graph is organized into a tree based upon relationships between the members of the graph and sent to the front end of the application through a series of queries. Metadata collected by the application is then stored within a database. A database table detailing this data type is displayed in Figure 3.

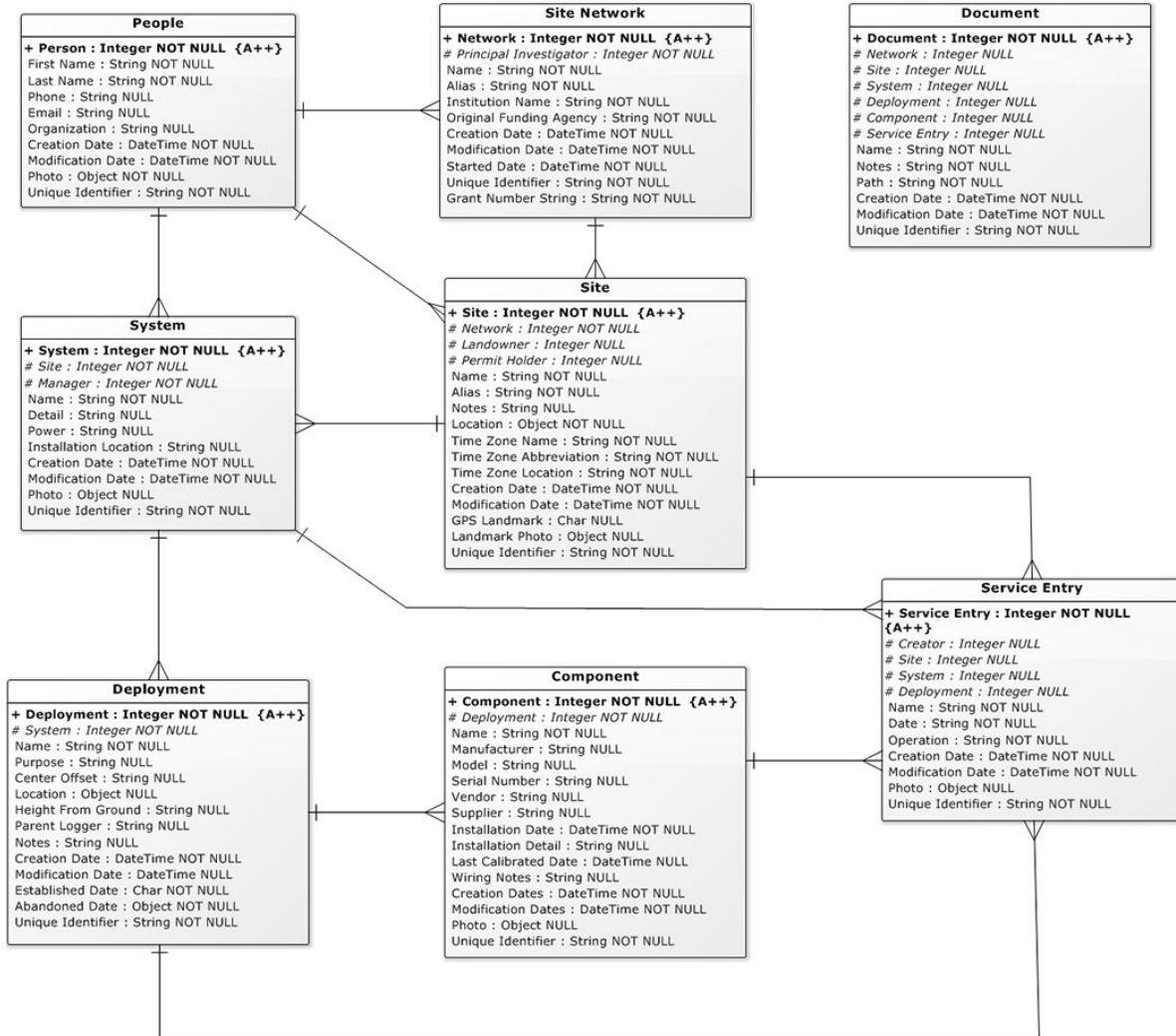


Fig 3. Database Table for the NRDC QA Application. Provided by Vinh Le and Connor Scully-Allison.

5. Detailed Design

The following section describes the detailed design of the NRDC QA application. Figure 4 shows the necessary steps necessary for the main front-end functionality to work. The first main point to take note of is that there are several functionalities that require internet access for initialization of the application. The second point to note is that there will be a hierarchy page, which will be where the dynamic parts of the application will be generated using the ontology.

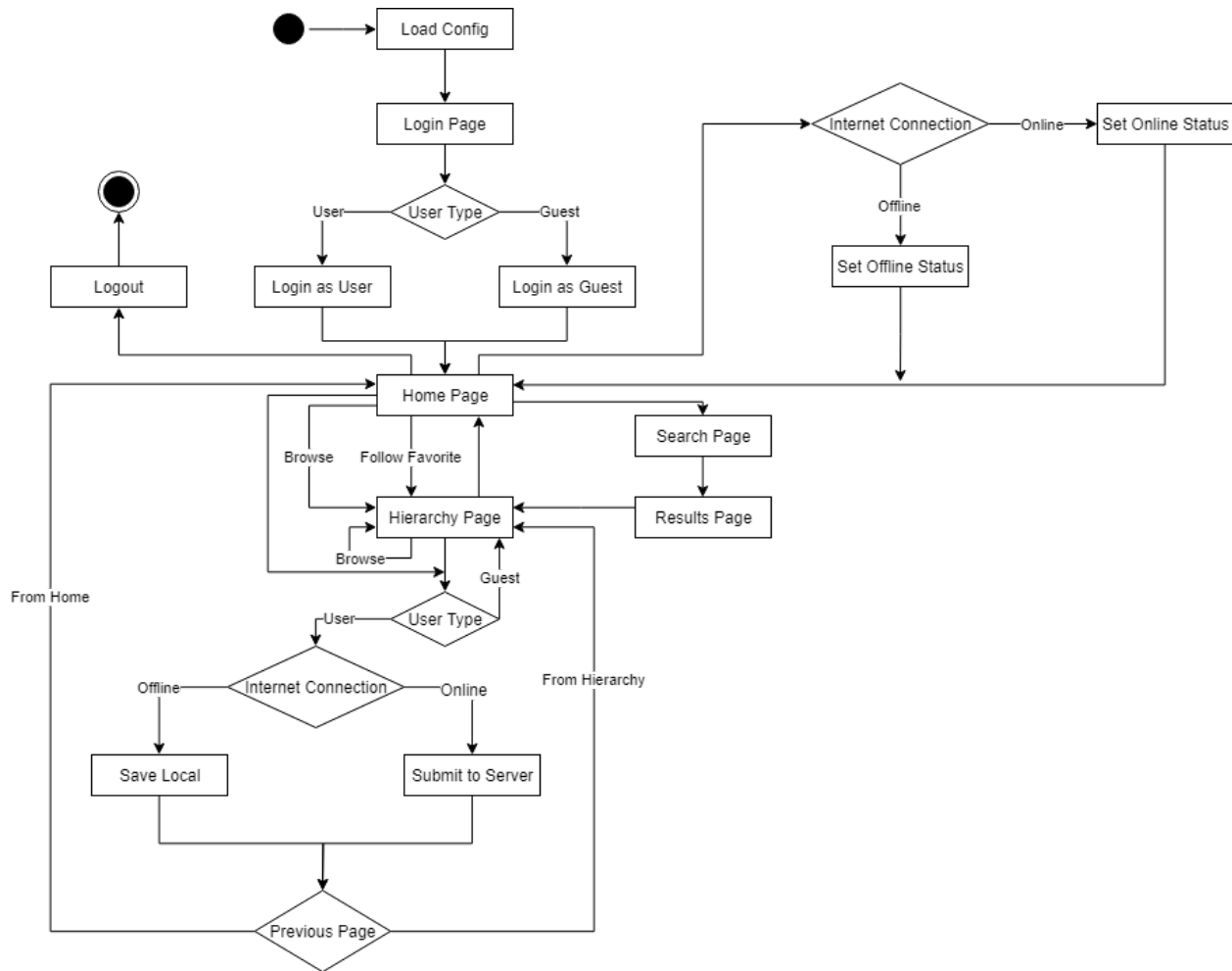


Fig 4. Activity Diagram of the front-end user interface

Figure 5 expands the login section of Figure 4, starting at the login page and continuing down to where the Home Page is created. It shows the more detailed aspects acquiring the data necessary to generate the application, including login information, template generated from a pre-processed ontology, and the metadata gathered from the database.

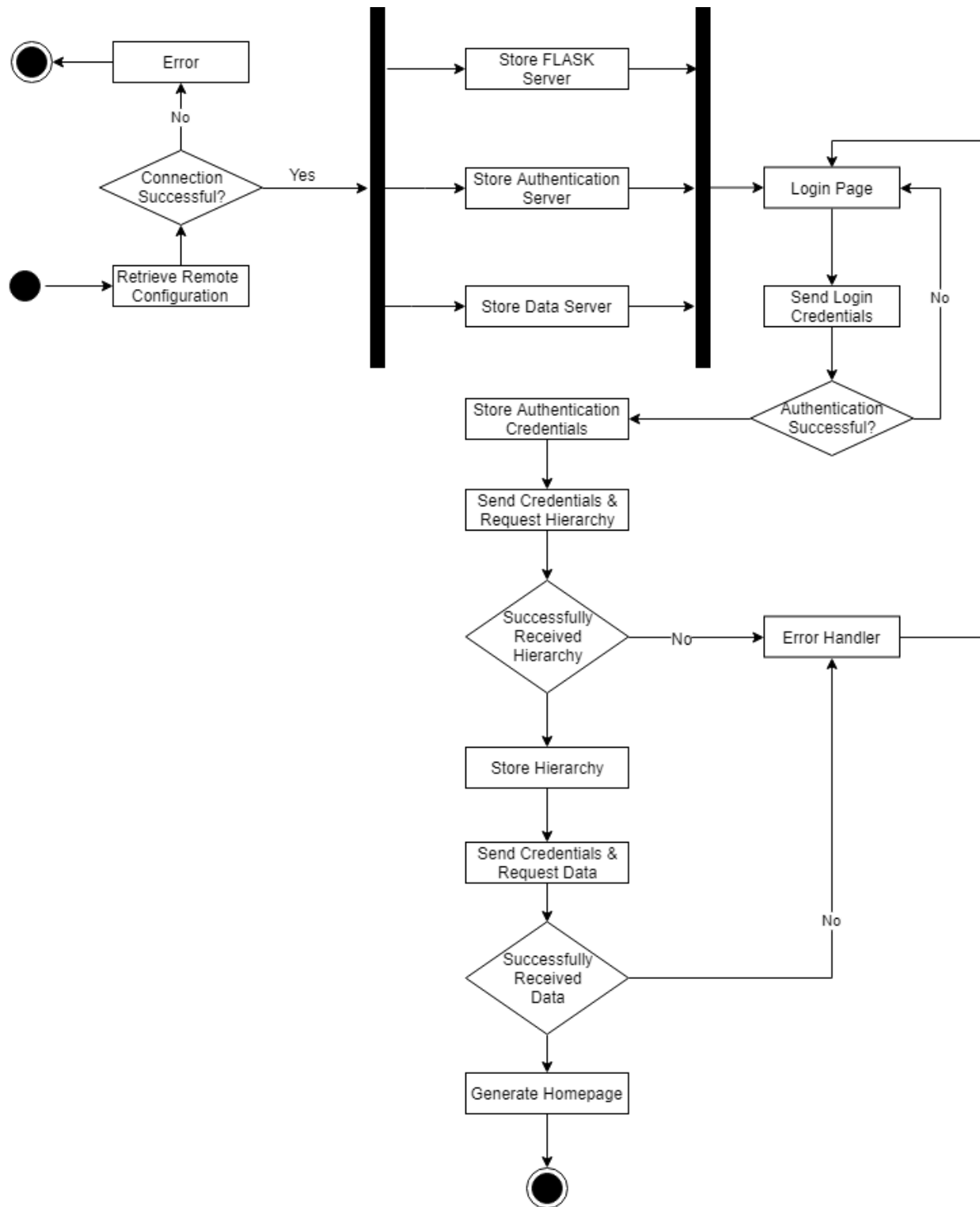


Fig 5. Activity Diagram of the login and initialization process

Navigation simplicity is an important feature of the NRDC QA application. Figure 6 is a state diagram that describes the flow of the page navigation for the user when they use the NRDC QA Application. The slide menu, search, and a hierarchy page that shows only the current section being used for data entry are the key features used to simplify navigation. In the state diagram, each transition is a button press of some object, and each state is a page. Each state describes what is shown on that page.

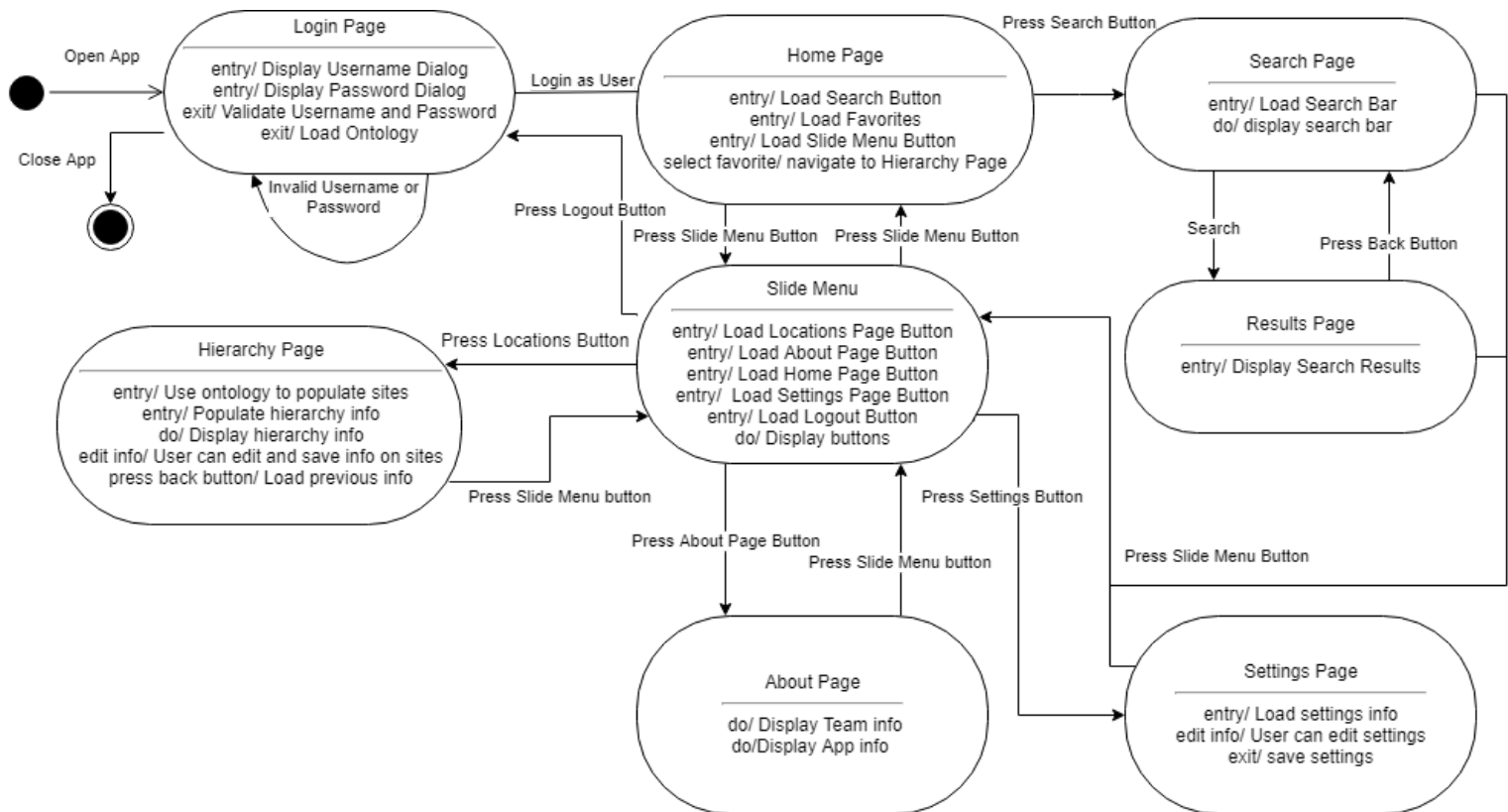


Fig 6. A State Diagram showing the application's page navigation.

The back-end utilizes several Flask services that are used for functions such as connecting to the database and storing metadata. Figure 7 is a state diagram that describes the functionality of a back-end FLASK service which is specifically designed to assist the front-end with dynamic page generation. It parses the ontology and creates a design template that it provides to the front-end application. Python has several tools that make the use of ontologies easier, and as a service, allows the parsing process to be done in a more efficient and compartmentalized way.

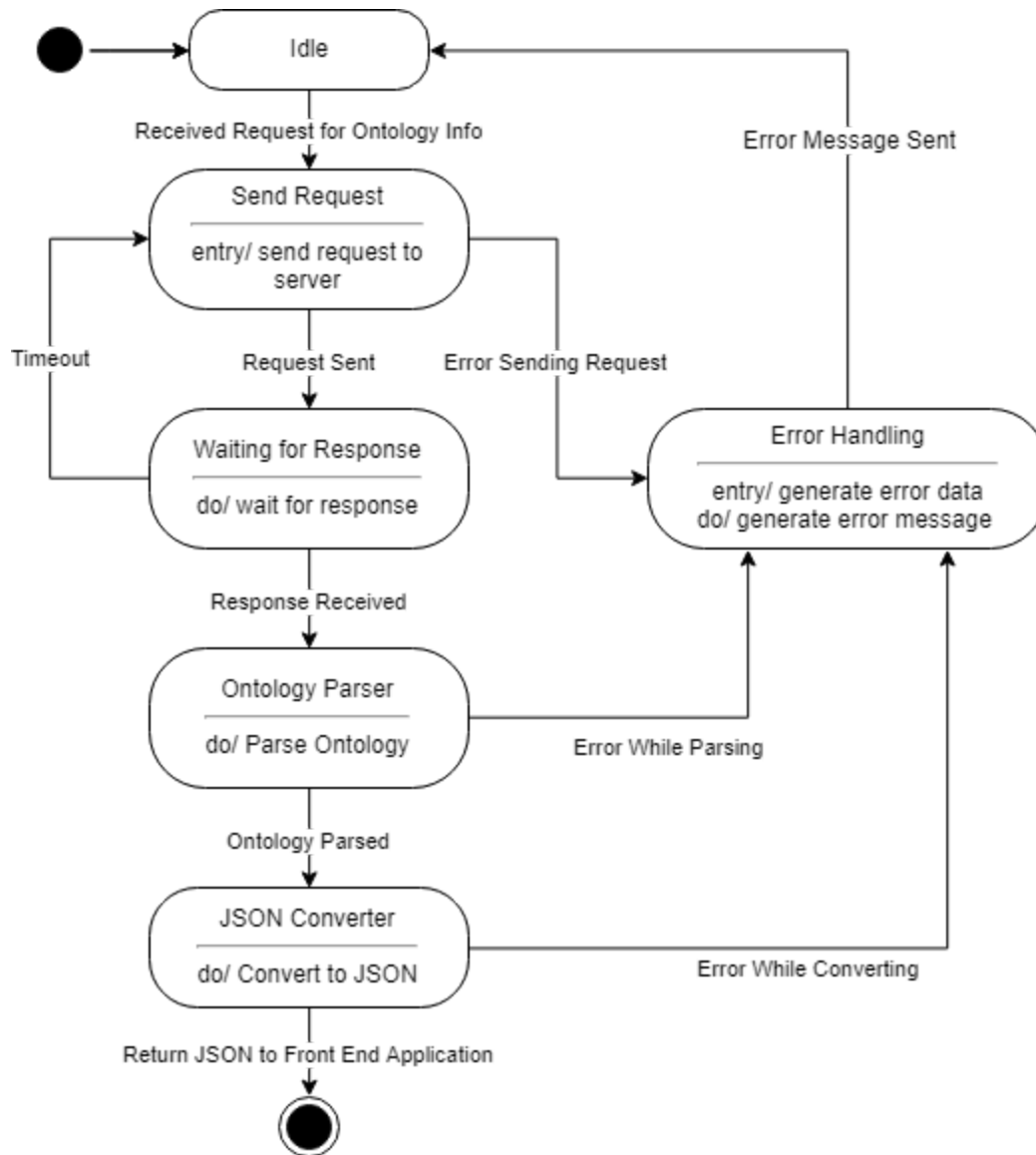


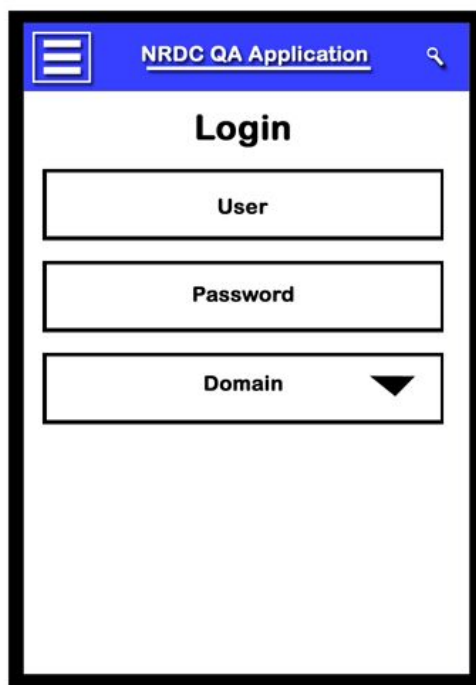
Fig 7. State diagram of a back-end FLASK service for parsing ontologies.

6. Initial hardware design (optional)

Does not apply to current project.

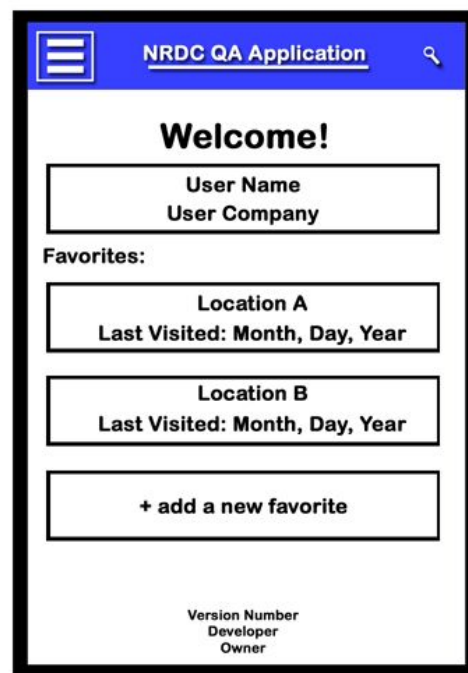
7. User interface design

The following three pages show the NRDC QA application interface designs. Figures 8 and 9 are the login and welcome views. Figures 10 and 11 display the “hamburger” menu and locations views. Figures 12 and 13 show the new and existing location views. Figures 14 and 15 are the updates and settings views. Figures 16 and 17 are the new search and search results view. The combination of these views gives a preliminary expectation for the design. Figure 18 shows the accordion menu that unfolds when changing sub settings. Figure 19 displays the update logs and its current contents.



The login screen features a blue header bar with a hamburger menu icon on the left, the text "NRDC QA Application" in the center, and a magnifying glass icon on the right. Below the header, the word "Login" is centered. There are three input fields: "User", "Password", and "Domain" with a dropdown arrow. The "Domain" field has a small black triangle pointing downwards.

Fig 8. The login screen the user sees when they open the application.



The welcome screen features a blue header bar with a hamburger menu icon on the left, the text "NRDC QA Application" in the center, and a magnifying glass icon on the right. Below the header, the word "Welcome!" is centered. There is a box for "User Name" and "User Company". Below that is a section titled "Favorites:" with two entries: "Location A" and "Location B", each followed by "Last Visited: Month, Day, Year". At the bottom of the favorites section is a button labeled "+ add a new favorite". At the very bottom of the screen, there is a small box containing "Version Number", "Developer", and "Owner".

Fig 9. The screen the user sees after they login.

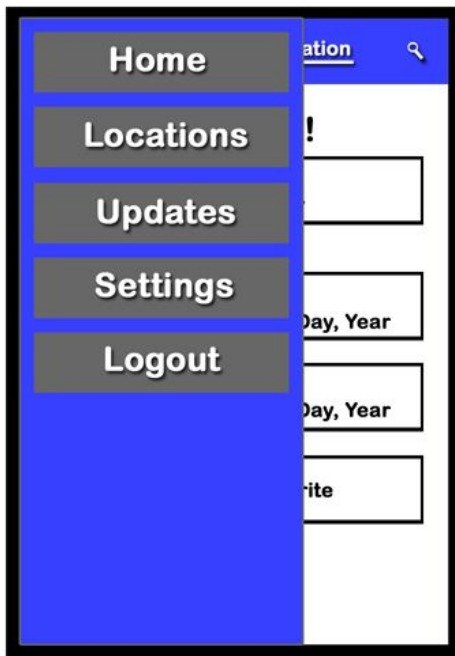


Fig 10. The “hamburger” menu that appears when the user accesses the menu button on the navigation bar.

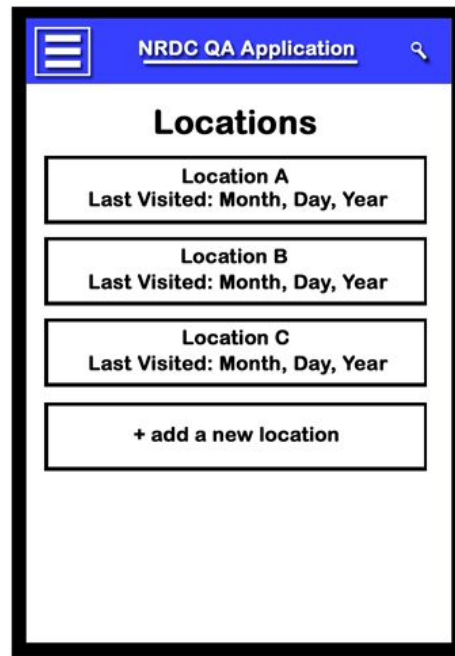


Fig 11. The locations screen that the user will use to navigate their permitted sites.

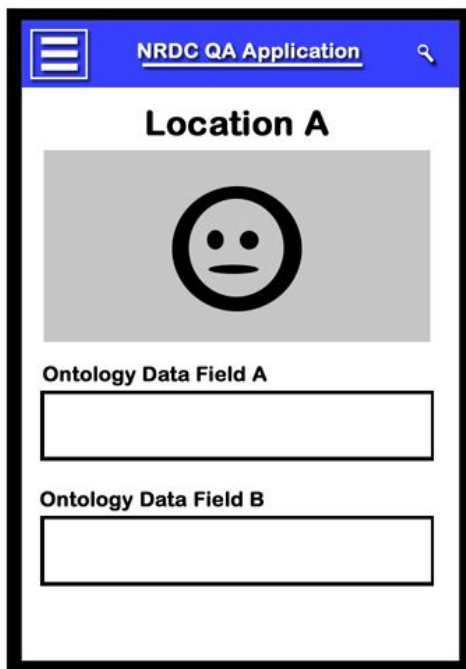


Fig 12. An existing location screen that will display a user image and filled data fields.

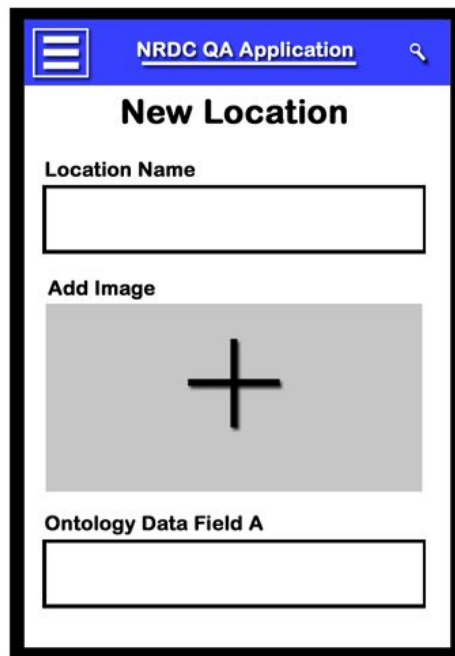


Fig 13. A new locations page that can allow for the addition of new sites. It contains a field for data and a user image.

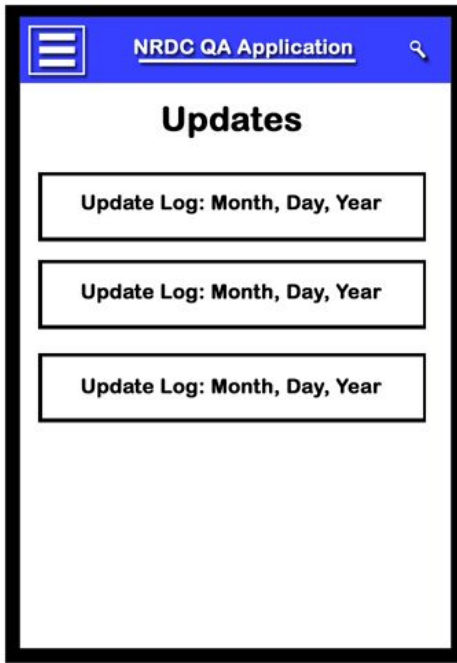


Fig 14. The updates screen will offer information regarding changes.

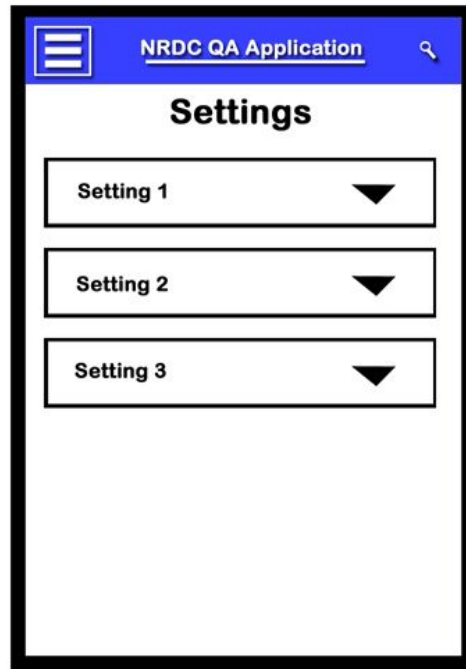


Fig 15. The settings screen that will allow a variety of user defined settings.

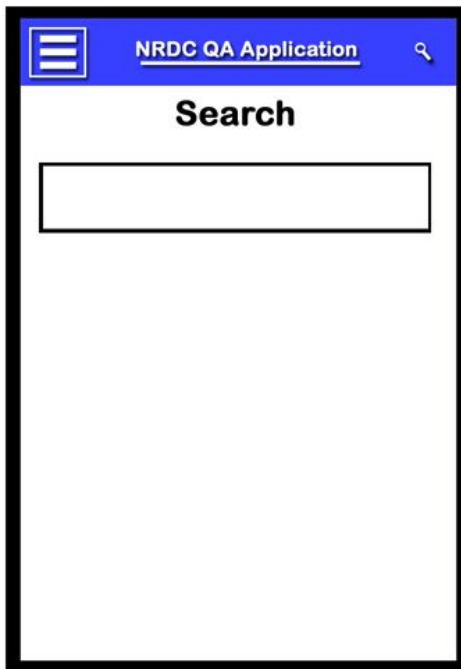


Fig 16. The new search screen that the user can access by selecting the magnifying glass on the navigation bar.

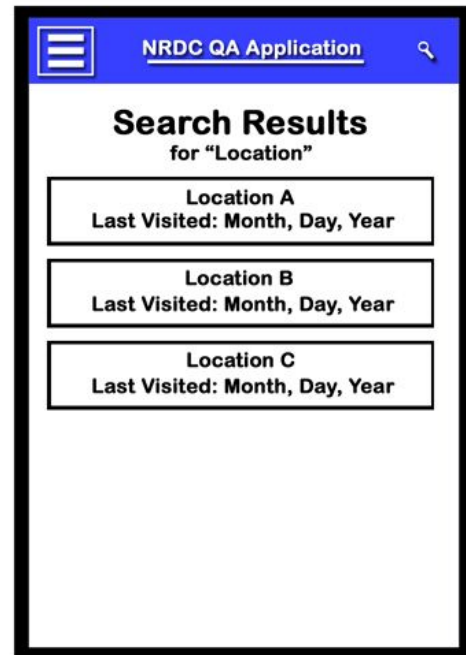


Fig 17. The search results screen that will display results from the users inquiry.

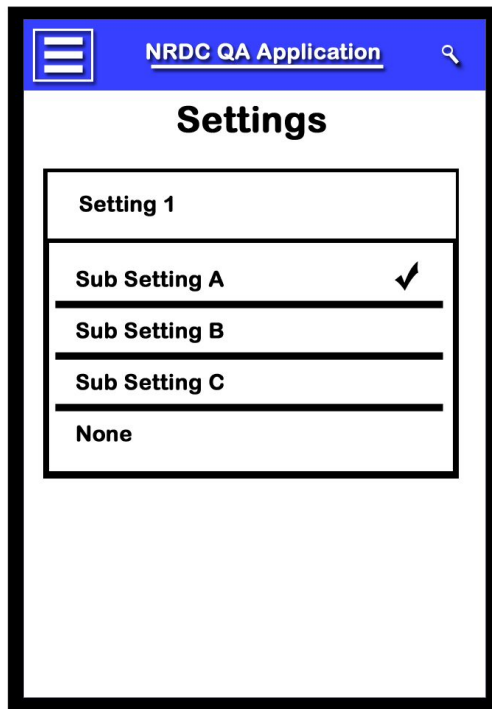


Fig 18. The submenu that will display available options when selecting from settings menu.



Fig 19. The updates log which will display the various changes and fixes for each build, every time the program is updated.

8. Glossary

1. Angular: An open-source framework for building web applications. It is based on Javascript.
2. API: Application program interface allows two software programs to communicate with one another.
3. Big Data: A large volume of structured or unstructured data that can be analyzed for patterns, trends, and associations.
4. Database: An organized collection of data designed to be easily accessed, managed and updated.
5. Discord: A chat room service and the main method of communication utilized during development.
6. Endpoint: A hardware device on a network designed to interconnect devices on the internet.
7. Environmental Science: The branch of biology concerned with the study of the natural world, including the conditions of the environment and the effects of these conditions on organisms.
8. Flask Service: A micro framework for Python used for managing website requests through HTTP with no data abstraction.
9. GPS: Global Positioning System used to find location data at any time of day in any weather conditions, using a radio navigation system.
10. Graph: An abstract data type used to define mathematical concepts and relations between objects through ordered pairs, nodes, and edges.
11. Hierarchical Navigation: The ability to navigate a website based on its organized structure that ranks items into levels of importance or relations.
12. In-situ Research: Research performed on site, allowing data to be collected without altering test conditions and without isolating measurements taken from the effects of other systems. In-situ Research is also a service specializing in training and assisting in mixed-methods.
13. Ionic: A platform for building mobile, desktop and web applications.
14. Metadata: A dataset used to describe other data.
15. Microframework: A minimalistic web application framework that lacks most of the functionality expected from a web application.
16. Nevada EPSCoR: A part of the Nevada System of Higher Education concerned with multidisciplinary learning within science, research, education, and technology.
17. NEXUS: A project under Nevada EPSCoR concerned with the effects of Solar Energy on the environment of Nevada. This is the most recent project under Nevada EPSCoR and the affiliation of the main group of technicians that would be utilizing the application.
18. NRDC: A research center concerned with sensor-based data management, and improving research cyberinfrastructure.

19. Ontology: A standard for the abstraction of data models in order to define the categories, properties and relations between data and entities .
20. Parser: A program, usually part of a compiler or interpreter, that breaks data into smaller elements for data management.
21. RESTful: A web service based on stateless operation that allow the requesting device to access and manipulate web resources through a textual representation.
22. Server: A program designed to provide services to other computer programs and its users.
23. Trello: A project management service designed to allow managers to turn large iterations into tasks and mark them as not started, in progress, and completed.
24. Triple Query: A request for information or data composed of subject-predicate-object.
25. XML: A language to define rules to encode documents. The format is designed to be easily readable by both humans and machines.

9. Contributions of Team Members

- Brianna Blain-Castelli
 - Contributions
 - 0. Cover Page
 - 4. High-level and Medium-level design
 - 8. Glossary
 - Extra Contributions
 - Time Worked: 13 hours
- Christopher Eichstedt
 - Contributions
 - 3. Introduction
 - 4. High-level and Medium-level design
 - 7. User Interface Design
 - Extra Contributions
 - Time Worked: 11 hours
- Matthew Johnson
 - Contributions
 - 1. Table of Contents
 - 3. Introduction
 - 2. Abstract
 - 5. Detailed Design
 - 7. User Interface Design
 - Extra Contributions
 - Helped create Class outputs in part 4.
 - Time Worked: 14 hours

- Nicholas Jordy
 - Contributions
 - 2. Abstract
 - 5. Detailed Design
 - 9. Team Contributions
 - Extra Contributions
 - Formatting of tables and figures
 - Time Worked: 12.5 hours