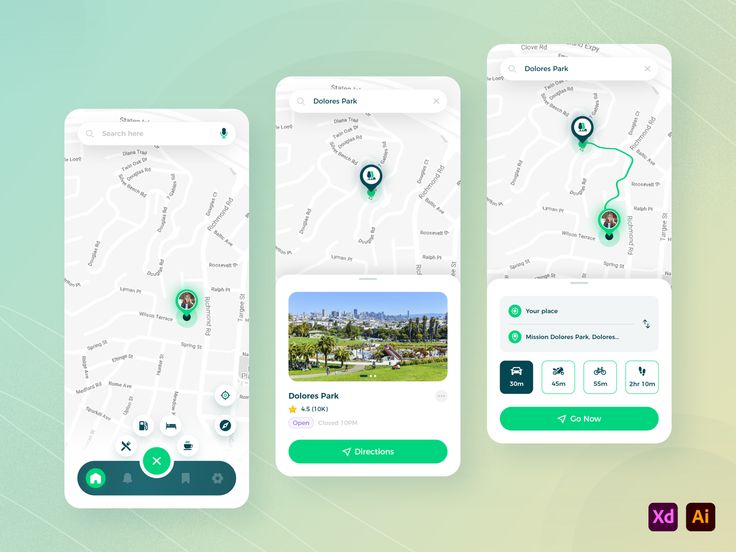
ADA-FRIENDLY NAVIGATION APP



by: Tracy Mai

June 6, 2024

CSC 3150 – Systems Design – Dr. Andy Cameron

Document Name: Mai Tracy – Final System Specification – ADA Friendly Navigation App – v2

Contents

[**1.** **Executive Summary** 3](#_Toc168610499)

[**2.** **Introduction** 3](#_Toc168610500)

[**2.1.** **Problem Statement / Project Vision** 3](#_Toc168610501)

[**2.2.** **System Capabilities** 3](#_Toc168610502)

[**2.3.** **Non-functional Requirements and Design Constraints** 4](#_Toc168610503)

[**2.4.** **System Evolution** 5](#_Toc168610504)

[**2.5.** **Document Outline** 6](#_Toc168610505)

[**3.** **Structural Model** 7](#_Toc168610506)

[**3.1.** **Model Introduction** 7](#_Toc168610507)

[**3.2.** **Class Diagrams** 7](#_Toc168610508)

[**3.3.** **Metadata** 8](#_Toc168610509)

[Application Class Diagram 8](#_Toc168610510)

[Location Class Diagram 9](#_Toc168610511)

[Map Class Diagramd 10](#_Toc168610512)

[MapOptions Class Diagram 12](#_Toc168610513)

[RegisteredUser Class Diagram 13](#_Toc168610514)

[Route Class Diagram 14](#_Toc168610515)

[UnregisteredUser Class Diagram 15](#_Toc168610516)

[User Class Diagram 16](#_Toc168610517)

[**4.** **Architecture Design** 17](#_Toc168610518)

[**4.1.** **Architecture Overview** 17](#_Toc168610519)

[**4.2.** **Infrastructure Model** 17](#_Toc168610520)

[**4.2.1.** **Deployment Diagram 1 – Architecture Overview** 17](#_Toc168610521)

[**4.2.2.** **Deployment Diagram 2 – Nodes and Artifacts** 18](#_Toc168610522)

[**4.3.** **Hardware and Software Requirements** 18](#_Toc168610523)

[**4.3.1.** **Hardware Components** 18](#_Toc168610524)

[**4.3.2.** **Required Software Components** 19](#_Toc168610525)

[**4.4.** **Security Plan** 19](#_Toc168610526)

[**4.4.1.** **Security Overview** 19](#_Toc168610527)

[**4.4.2.** **Security Plan** 19](#_Toc168610528)

[**5.** **User-Interface** 20](#_Toc168610529)

[**5.1.** **User-Interface Requirements and Constraints** 20](#_Toc168610530)

[**5.2.** **Window/Screen Navigation Diagram** 20](#_Toc168610531)

[**5.3.** **UI Wireframes** 20](#_Toc168610532)

[**6.** **Appendices** 21](#_Toc168610533)

[**6.1.** **Glossary** 21](#_Toc168610534)

[**6.2.** **References / Bibliography** 21](#_Toc168610535)

[**6.3.** **Supporting documentation** 21](#_Toc168610536)

# **Executive Summary**

ADAFNA enables people with disabilities to make use of tools widely available for able-bodied people, which do not currently adequately support their mobility restrictions. Developed by a small team from Seattle, WA, this team has worked hard to design a product to meet everyone's needs, regardless of their abilities. This document establishes the major advancements in the design of the application over the past few weeks, including addressing security concerns, implementation details, and UI/UX mockups. As the development of the product continues into the next quarter, we hope to provide a roadmap for what has been already done and planned for, and what efforts still need to be taken. In the future, the team hopes to provide an easy-to-use prototype to be able to accurately benchmark operating costs and load restrictions and test all functionality.

# **Introduction**

This System Specification will focus on the technicalities of the ADA-Friendly Navigation App (ADAFNA). The ADAFNA routes physically disabled people directly to their destination through ADA-friendly routes, ramps, pathways, and appropriate building entrances. The phone app will feature an easy interface, allowing users to choose their route based on their physical abilities, and notify the users if the path does not seem to be 100% compatible with their needs. The app will also show ADA-friendly bathrooms, access ways, and regular rooms in any building the user desires. More details about the creation of the ADAFNA can be found within the System Proposal.

## **Problem Statement / Project Vision**

The development of the ADA-Friendly Navigation App came from the lack of support the physically disabled have to wander the world. Originally built for cars, navigation apps have since transformed for bikers, walkers, bus riders, and even air transport. However, there is still no solution for physically disabled people. If they use the walker’s path, they are presented with stairs that cannot be traversed with wheelchairs. If they use the biker’s path, there is no promise that the ramps are ADA-accessible. The ADAFNA solves these problems by routing users through ADA pathways they can take, helping people of all abilities discover the world and enriching everyone’s lives with more experiences.

The ADAFNA is a first-of-its-kind, life-changing application. Those involved would benefit not only financially but would be able to make a substantial difference in the world. The creators’ goal behind the ADAFNA is simply to help physically disabled people travel the world. Stakeholders involved would be viewed more positively and respected in the public eye, creating more trusted corporate values from the public. As the ADAFNA becomes more successful and the number grows, customer satisfaction and loyalty enhancement will grow alongside it. This can result in long-term viability as users will trust and return to the company for similar products. Since this project involves many teams, stakeholders working on this project themselves, will have increased cross-team organization as teams learn to work together and get better through it.

## **System Capabilities**

This section will briefly present the 8 Functional Requirements for the ADAFNA. More details about the requirements can be found on pages 14 to 15 of the System Proposal. A visual representation of how these requirements interact with one another can be found on page 17 of the System Proposal. To preface, there are faces of various use cases that belong both the Functional and Non-Functional Requirements and are shown accordingly in both this System Specification and the System Proposal.

UC-3, Use Saved Addresses: Users should be able to save routes and addresses to easily access them, regardless of whether they choose to log into the ADAFNA with their email.

UC-5, Show ADA Accessible Rooms and Elevators: The ADAFNA should show all ADA-Friendly accessways, rooms, bathrooms, elevators, and wide doors.

UC-6, Save Addresses:

UC-7 Choose/Change ADA Type of Path: The ADAFNA should always show the most optimal path dependent on the user’s physical ability, ensuring the path is 100% ADA accessible and never show a path that the user is unable to take.

UC-8, Show ADA Warnings: Users should be notified of any concerns related to the pathways to their destination.

UC-9, Route to Destination: The application must not stop routing just outside the building but will route inside the building through ADA-accessible entries through elevators and directly to their destination when necessary.

## **Non-functional Requirements and Design Constraints**

The constraints related to this project are mostly based on time, budget, and technical limitations, as outlined on page 5 of the System Proposal. It is most important that this project is completed on time; if the project runs into a problem that may run over time, its functional requirements should be prioritized to align with the original vision of the product for the minimum viable product (MVP). The budget established was to make sure that people would be compensated correctly for their work. However, this does put technical limitations on what software the developers may use. For the MVP, the software developers should choose software within the budget. As resources grow over time, software can be updated. The Feasibility Assessment showed that the project is very feasible with a relatively low to medium risk in all facets, leaving no overwhelming concerns, as shown on pages 11 through 13 of the System Proposal.

This section will briefly present the 8 Non-Functional Requirements for the ADAFNA. More details about the requirements can be found on page 16 of the System Proposal. A visual representation of how these requirements interact with one another can be found on page 17 of the System Proposal. To preface, there are faces of various use cases that belong both the Functional and Non-Functional Requirements and are shown accordingly in both this System Specification and the System Proposal.

UC-1, Normal Usage of the App: This requirement defines that the app and its interface should be: easy-to-use, have a quick response time, different visual options, and function seamlessly regardless of whether the user chooses to log into the app

UC-2, Initial Log Into the App: When the users are given the option to log in with their email, the interface should easily link other email forums for easy access.

UC-3, Use Saved Addresses: The application should not store direct information about people’s identity, and users who choose to log into the app should be able to access their saved addresses when logging in on another device.

UC-5 Show ADA Accessible Rooms and Elevators, UC-7 Choose/Change ADA Type of Path

UC-8 Show ADA Warnings: Every route, room, etc., that the app presents should be ADA-accessible or follow the user’s request and ADA legalities.

UC-9, Route to Destination: The application will have a visual and voice option to tell users the directions and when they have reached their destination.

UC-10, Help Page: Provides a necessary help page for user inquiries.

## **System Evolution**

The basic functional requirements will be delivered in Version 1 of the ADA-Friendly Navigation App, as outlined above. Version 1 is the MVP focuses on having every necessary app function. Keeping the necessary functions to a minimum makes the app simple and easy for users. The future expansion and growth of the ADAFNA will prompt a Version 2, in which the app will be updated to support multiple languages aside from English, reaching users from different walks of life.

* + 1. **Version 2 Changes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name**: Multiple Languages | | **ID**: UC-11 | **Importance**: Should Have |
| **Primary Actor**: Application User | **Use Case Type**: Overview, Essential | | |
| **Supporting Actors:** Language Database | | | |
| **Stakeholders and Interests**:  Marketing Team:  The release of various in the ADAFNA should be made public for users to know. The marketing team must be notified to go forward with their marketing strategy. | | | |
| **Brief Description**:  The Application User will now be able to choose what language the ADAFNA functions in. | | | |
| **Trigger**:  **Type** (mark one): \_X\_\_ External \_\_\_ Temporal | | | |
| **Relationships**:  **Association**:  **Include**:  **Extend**:  **Generalization**:  The generalization that connects to this use-case is from UC-1, Normal Usage of the App, and UC-2, Initial Log Into App. Once the Application User chooses their desired language, the user should be able to use the rest of the functionalities of the ADAFNA like normal (UC-1). If the user chooses to change their desired language whilst being logged into the ADAFNA with their email (UC-2), this change will be seen in other devices where they have logged into the ADAFNA with their email. Resulting in the other devices presenting the ADAFNA in the user’s desired language. | | | |
| **The Normal Flow of Events**:   1. The Application User opens the ADAFNA to use its functionalities and be routed to their destination (UC-1). 2. The Application User changes the desired language that the ADAFNA should function in. 3. The Application User continues to use the rest of the functionalities of the ADAFNA however their heart desires. | | | |
| **Sub-flows**:  2.1. The Application User is logged into the ADAFNA with their email.  2.2. Other devices logged in with the same email will be notified of this language change and will run the ADAFNA in the preferred language as defined by the user.  2.3 The Application User continues onto step 3 of the Normal Flow of Events. | | | |
| **Alternate/Exceptional Flows**: | | | |
| **Special Requirements:** | | | |
| **To do/Issues:** | | | |

## **Document Outline**

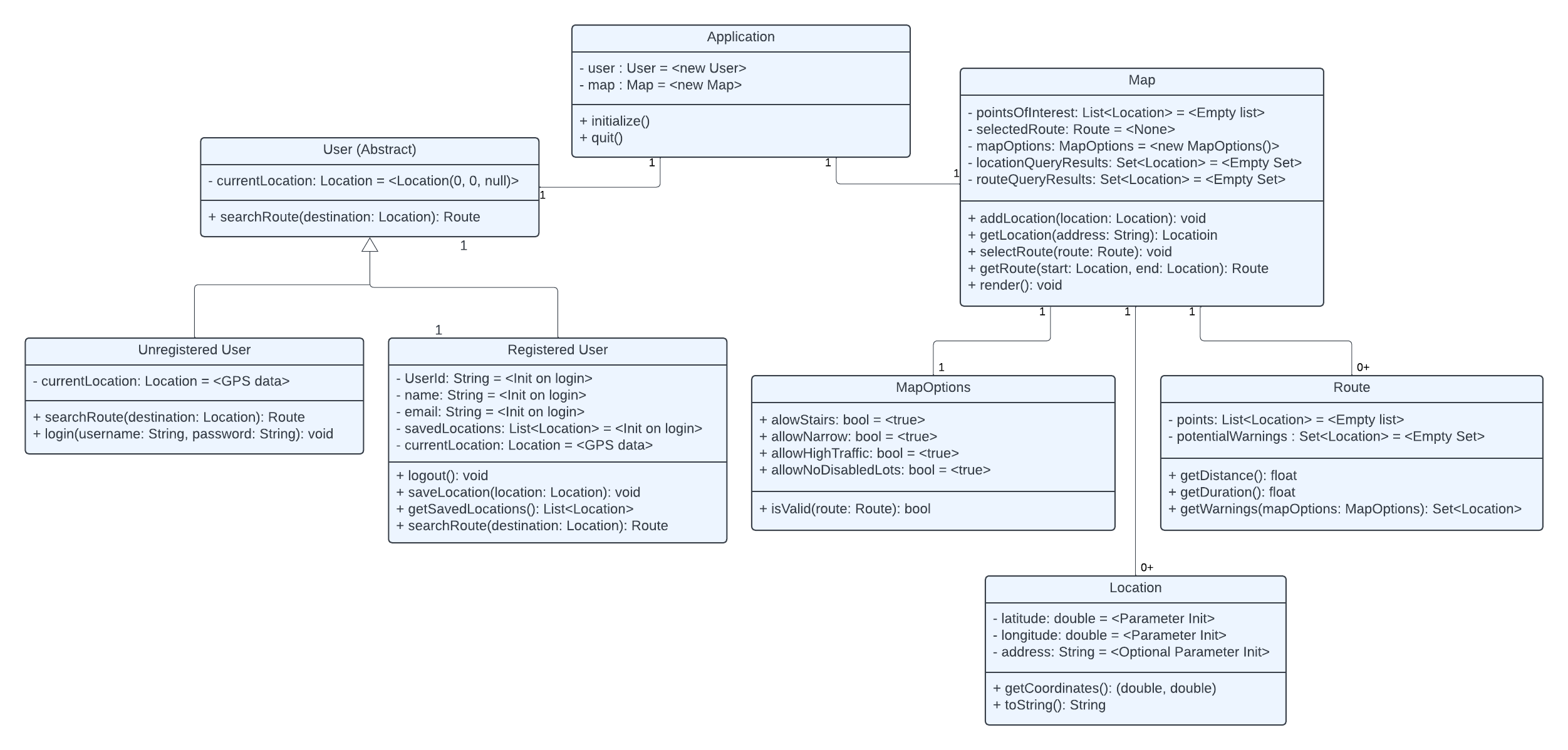
Throughout the rest of this System Specification, we explore the various aspects that make ADAFNA work, starting with the organization and flow of required app data in section 3. This is followed by an overview of the architecture in section 4, including network communication structure, interactions between various classes, and both hardware and software components. This section continues to then describe security concerns. Section 5 includes all front-end interface designs, and app navigation. Section 6 contains various helpful appendices.

# **Structural Model**

## **Model Introduction**

ADAFNA maintains a top level Application class, which then manages the various pieces of state which define function. The following sections in detail describe the interactions between the different classes, first providing a high level overview, then delving into details in section 3.3.

## **Class Diagrams**

The following diagram describes major pieces of state held by our top-level application object.

Note that all classes also come with default constructors. State marked with “Init” must be initialized following a relevant API request.

## **Metadata**

Application Class Diagram

Description: Top-level class that maintains all application state, namely an instance of a user and a map class.

Visibility: Public

Is Abstract: No

Additional Information:

* *Contained Attributes*

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| user | User using the current application instance | N | 1 |
| map | Object containing all information regarding the displayed map | Y | 1 |

* *Operations*

Constructor:

* Initializes an unregistered user object.
* Initializes default map object.
* Invoke Initialize()

Initialize():

* Perform a GET request to the backend to request all stored user data
* Populate map and user state.
* Invoke map.render() to display a default map

quit():

* Perform a PUT request to store all user and map state that required persistent storage to the database.
* Gracefully stop displaying the map, and invoke all destructors.

Location Class Diagram

Description: Packages state to represent a single world location, with an optional address.

Visibility: Public

Is Abstract: No

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| latitude | Latitude value of this location | Y | 1 |
| longitude | Longitude of this location | Y | 1 |
| address | Optional human-readable address to attach to the given location, if one exists. | N | 1 |

* *Operations*

Constructor:

* Initializes all state to their parameter set values (latitude, longitude, and optionally an address)

getCoordinates() -> tuple(double, double):

* Returns the stored latitude, longitude values in a single tuple.

toString() -> string

* Returns a human readable representation of the location, using the address if it exists, and if not, “(<latitude>, <longitude>)”.

Map Class Diagram

Description: Class containing all information regarding the displayed map, including selected options and points of interest.

Visibility: Public

Is Abstract: No

Additional Information:

* *Contained Attributes*

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| pointsOfInterest | List of all locations to display a pin at. Note this can be empty, depending on current application state. | N | 0.. |
| selectedRoute | Highlighted route to display to user | N | 0 |
| mapOptions | Current user-selected navigation options | N | 1 |
| locationQueryResults | Search results from a client query for locations | N | 0.. |
| routeQueryResults | Search results from a client query for a route from two locations | N | 0.. |

* *Operations*

Constructor:

* Initializes all state to their default values (to be changed via queries / user selections)

addLocation(location: Location):

* Adds the location parameter value to the points of interest list, to be displayed on the map as pins.

getLocation(address: String) -> Location:

* Send a GET request to the backend
  + Backend preprocesses the request and forwards it to the Google Maps API.
* Await the results from the GET request
* Convert results into a new Location object, and return (to be potentially added to locationQueryResults, or used in further queries, or to add to pointsOfInterest)

selectRoute(route: Route):

* Set the given route as the selectedRoute, to be highlighted on the displayed map.

getRoute(start: Location, end: Location) -> Route:

* Send a GET request to the backend containing the given start and end locations.
  + Backend processes the request, and forwards it to the Google Maps API to construct the route between the two locations
* Await the results, and return a new Route object

render():

* Update the displayed map to reflect the state in pointsOfInterests and selectedRoute.

MapOptions Class Diagram

Description: Packages the user selected options for various accessibility restrictions to apply to routes.

Visibility: Public

Is Abstract: No

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| allowStairs | Whether or not to show ADA warnings for stairs which would impede movement. | N | 1 |
| allowNarrow | Whether or not to show ADA warnings for narrow streets or spaces which would impede movement. | N | 1 |
| allowHighTraffic | Whether or not to show ADA warnings for high traffic spaces which would impede movement. | N | 1 |
| allowNoDisableLots | Whether or not to show ADA warnings for a lack of diabled parking which would impede movement. | N | 1 |

* *Operations*

Constructor:

* Initializes all state to their default values

IsValid(route):

* Iterates through the nodes in a route to determine whether any restriction would be applied to the route. Only restrictions which are marked as not allowed would apply to the validity check.

RegisteredUser Class Diagram

Description: Packages an ordered list of locations to be used as a route.

Visibility: Public

Is Abstract: No

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| userId | Unique identifier for the logged in user. | Y | 1 |
| name | Name of the logged in user. | Y | 1 |
| email | Email address of the logged in user. | Y | 1 |
| savedLocations | List of locations previously saved by this user | Y | 0.. |
| currentLocation | Current location of the user, as set by a periodic GPS ping | N | 1 |

* *Operations*

Constructor:

* Initializes all read-only state to parameter specified values. These values should be populated when the user logs in.

logOut():

* Sends a PUT request to the backend, to store all (potentially changed) user data to the database.
* Change application stored user to become an UnregisteredUser.

saveLocation(loc: Location) :

* Stores the parameter location to the maintained savedLocations piece of state.

getSavedLocations () -> List<Location>:

* Returns the list of savedLocations, potentially to populate the pointsOfInterest map state, to be displayed on the map.

searchRoute(destination: Location) -> Route:

* Sends a GET request to the backend with the passed destination and the stored currentLocation
  + Backend processes the request, and forwards it to the Google Maps API to determine all routes from the currentLocation to the destination.
* Await the request, and construct a Route object to return (potentially to be passed to the map object to store in routeQueryResults)

Route Class Diagram

Description: Packages an ordered list of locations to be used as a route.

Visibility: Public

Is Abstract: No

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| points | Ordered list of points to connect together into this route. | Y | 1.. |
| potentialWarnings | A set of locations which contain an ADA warning, according to the specified MapOptions, to show a warning pin on. | Y | 1 |

* *Operations*

Constructor:

* Initializes all states to their default values, points initialized with a parameter, and potential warnings with the MapOptions determined warning points.
  + Note that the points list comes from a backend GET query.

getDistance() -> float:

* Iterates through points, getting the sum of the Euclidean distance between adjacent points in the list.
* Returns the computed sum.

getDuration() -> float:

* Gets the distance using the above function, then determines the amount of travel time required to traverse that distance.
* Returns the computed time.

getWarnings() -> float:

* Returns the set of potentialWarnings, to be displayed as points of interest on the selected route.

UnregisteredUser Class Diagram

Description: Packages the state of a user which has not yet logged in.

Visibility: Public

Is Abstract: No

* *Operations*

searchRoute(destination: Location) -> Route:

* Sends a GET request to the backend with the passed destination and the stored currentLocation
  + Backend processes the request, and forwards it to the Google Maps API to determine all routes from the currentLocation to the destination.
* Await the request, and construct a Route object to return (potentially to be passed to the map object to store in routeQueryResults)

login(username: string, password: string):

* Sends a GET request to the backend after salting and hashing the password
  + Request gets processed by the backend, which retrieves the stored salted and hashed password from the database and validates the login request.
  + Backend determines if the login was valid or not, and responds with either a LoginOK (containing the users information) or LoginNotOK
* Await result, ignore if the result was a LoginNotOk
* If the result was a LoginOk, creates a RegisteredUser object with the returned user values
* Sets the Application user state to be a RegisteredUser instead.

User Class Diagram

Description: Abstract class which is used by the application to store user information. Can be an instance of either RegisteredUser or UnregisteredUser

Visibility: Public

Is Abstract: Yes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Description | Read Only? | Multiplicity |
| currentLocation | Abstract (however both types of users periodically ping GPS data to set this value) | N | 1 |

* *Operations*

searchRoute(destination: Location) -> Route:

* Sends a GET request to the backend with the passed destination and the stored currentLocation
  + Backend processes the request, and forwards it to the Google Maps API to determine all routes from the currentLocation to the destination.
* Await the request, and construct a Route object to return (potentially to be passed to the map object to store in routeQueryResults)

# **Architecture Design**

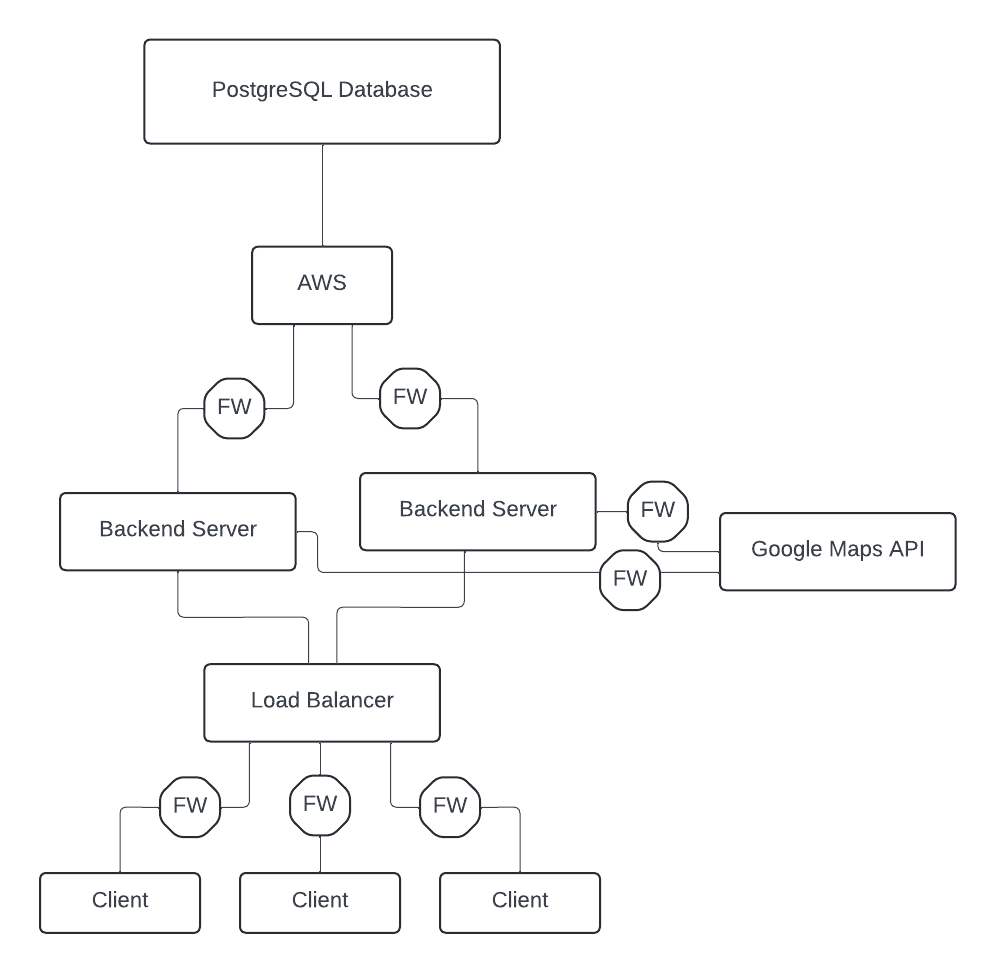
## **Architecture Overview**

As with any product, there are a multitude of software, hardware, and network requirements to make the overall system function as intended. Within this section, we provide an overview of the required actors and outline their interactions. In short, ADAFNA is a standard three-tier client-server system. This sort of design provides a highly available service with decent system consistency while also allowing for future scaling by increasing the scope of each tier (load balancing as necessary).

## **Infrastructure Model**

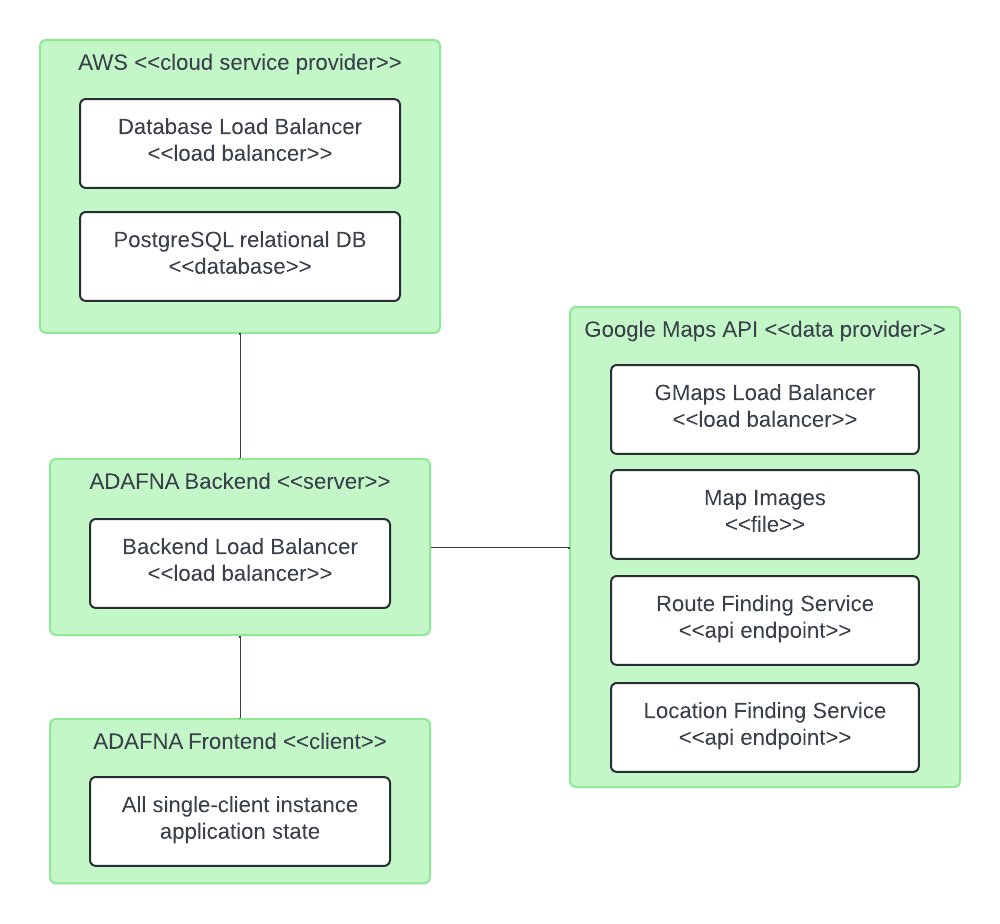
The following diagrams describe the messages between different system components, to ensure safe and secure communication and Maps API / Database usage.

* + 1. **Deployment Diagram 1 – Architecture Overview**

The following diagram summarizes the communication between various nodes in the system. Note that any octogon marked with FW represents a firewall, to keep any server nodes and clients protected.

Note, this assumes that all database sharding, replication, and load balancing is handled by AWS.

* + 1. **Deployment Diagram 2 – Nodes and Artifacts­**

The below diagram provides a component level view of the system architecture.

## **Hardware and Software Requirements**

As previously discussed, ADAFNA uses a three-tier client-server model. Each tier can be scaled independent of the others.

* + 1. **Hardware Components**

Clients will only require a (GPS-supported, internet enabled) device to download and use the service. On initial release, we only support Android platforms, however this can easily be later scaled to include desktops, laptops, and other mobile devices. On the other side, ADAFNA requires at least one server to host the backend service to respond to various client queries (including login, navigation, etc.). Note that if client demand increases, this may potentially need to be distributed and scaled to include multiple backend servers. Clients would send queries to a load balancer; the load balancer would distribute the request amongst the multiple servers. A database will be required to store client information. Similar to servers, depending on client traffic, the database may need to be replicated and sharded to provide highly available access to stored client data. Note that if the database is distributed in such a manner, consistency is a low priority for ADAFNA due to a lack of competing client requests

* + 1. **Required Software Components**

Client software will be limited to Android OS version 13+. Servers will be running an Ubuntu Linux distribution. Servers will communicate with a relational PostgreSQL database hosted on Amazon’s AWS. Google provides a Google Maps API with the required functionality to enable the backend to properly service client requests.

## **Security Plan**

* + 1. **Security Overview**

ADAFNA leverages a multitude of sensitive and personal user data; protecting this data from adversaries becomes the primary concern of the security protocol. Overall, all data must be end to end encrypted, through a standard RSA protocol. A front end application will establish a secure channel to communicate with the backend, and then use that channel for all messages sent. All passwords are properly salted and hashed to avoid storing plain text. Past that, we only use trusted third party services to aid in ADAFNA’s operation. The firewalls described in the above diagrams also help to keep the network safe to use. To ensure system durability, all databases are properly replicated, through AWS.

* + 1. **Security Plan**

The below table summarizes different potential threats and vulnerabilities that must be addressed by the following controls. Boxes marked N/A do not apply, as offering further protection is unfeasible.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Environmental | Circuit Failure | Virus | External Adversary | Internal Adversary | Man in the Middle |
| Users | N/A | N/A | 2, 4, 9, 13 | 1, 3 | 1, 3 | 1, 2, 3, 5 |
| Backend Servers | 7, 11, 12 | 6, 7, 10, 11 | 2, 4, 9, 13 | 1, 3, 7, 8 | 1, 3, 7, 8 | 1, 2, 3, 5 |
| External Endpoints | 13 | 6, 13 | 2, 4, 9, 13 | 1, 7, 8, 13 | 1, 7, 8, 13 | 1, 2, 3, 5 |
| Data | 6 | 6 | 1, 2, 4, 5, 9, 13 | 1, 7, 8, 13 | 1, 7, 8, 13 | 1, 2, 3, 5, 13 |

*Controls*

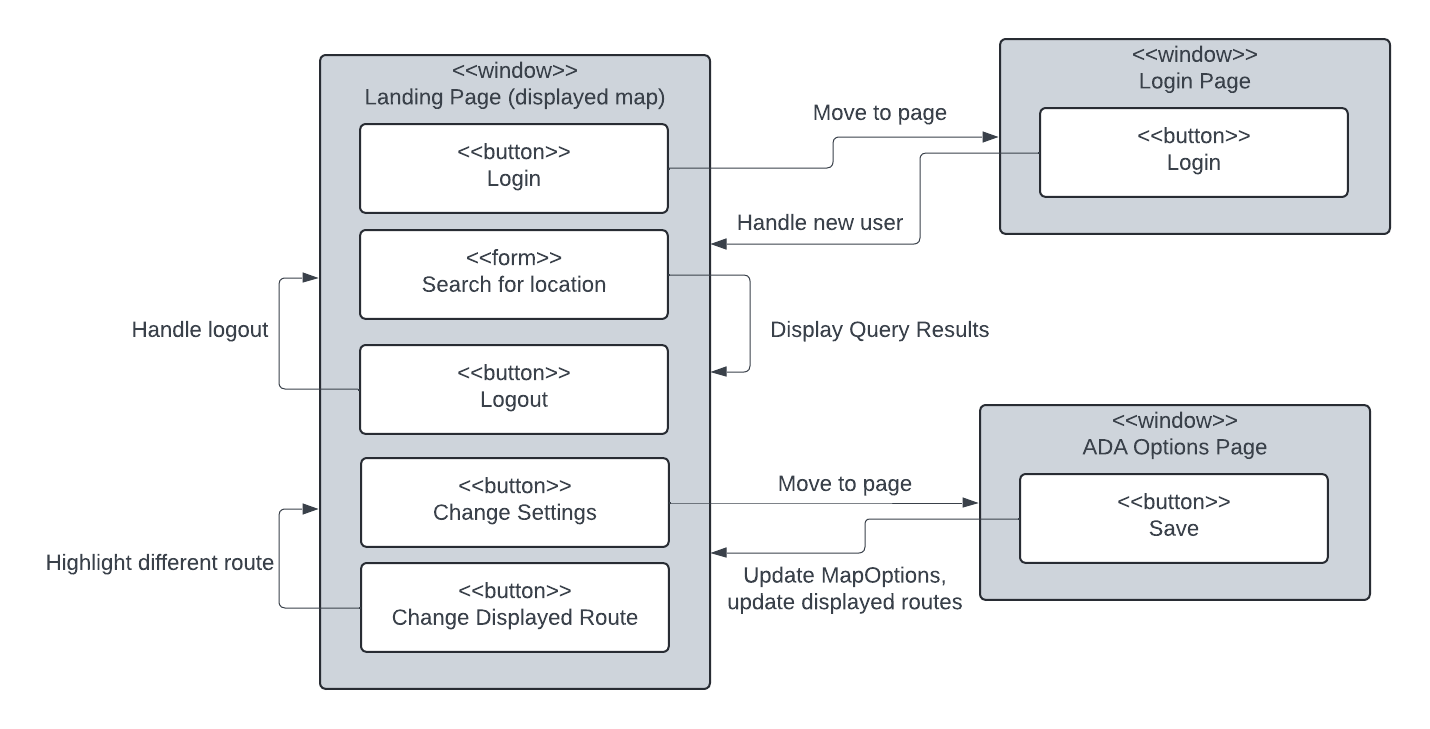
1. End-to-end encryption
2. Network firewalls
3. Authentication and authorization
4. Third-party network security
5. Salted and hashed password storage
6. Server replication
7. Onsite employee response
8. Badge access
9. Anti-virus and Anti-malware software (on both hosts, and on the network firewalls)
10. Seperate and stable data center power supply
11. Fire suppression (air exhaust, sprinklers), evacuation procedures, and other environmental protections
12. Good data center locations (low natural disaster rate)
13. Secure and trusted service providers (GMaps API and AWS)

# **User-Interface**

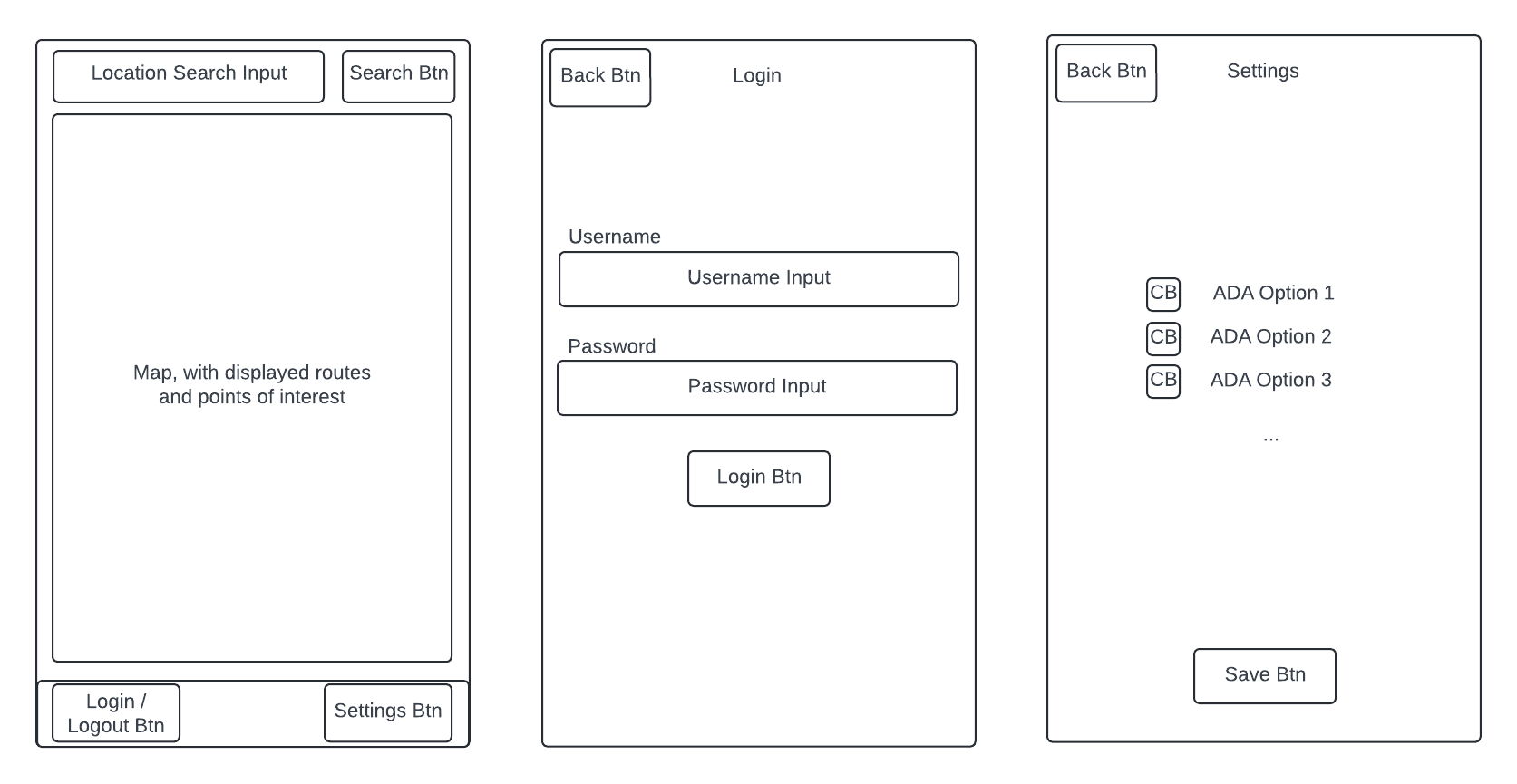
## **User-Interface Requirements and Constraints**

ADAFNA’s UI design is similar to that of standard navigation applications like Waze, Google Maps, or Apple Maps. It however extends the functionality of all of these precursors via ADA friendly options. In section 5.2, we provide a high level navigation diagram of how a user would interface with the UI. In section 5.3, a screen-by-screen wireframe is shown. These two diagrams combined represent the overall experience of using ADAFNA to log in, perform queries, and display routes.

## **Window/Screen Navigation Diagram**

The below diagram describes the different buttons, forms, and functions provided to the user when using the application. Note that certain form information is omitted; those forms would include textboxs, checkboxs, or other input elements as required to set the given data classes defined in section 3. For instance, the login page would provide inputs for the requested username, password combination. The ADA options page would provide checkbox inputs to select the desired ADA options to store in the maintained MapOptions object. If a button loops back to the same displayed window, then that operation simply updates what is displayed on the same screen.

## **UI Wireframes**

The below diagrams display a minimal representation of the three aforementioned windows. Note that the Login / Logout button would change in function and label depending on if the user has previously logged in or not (checked via the type of the stored user field in the top level application class). If the button is currently a login button, then it will redirect the user to the login page. If it is a logout button, then it will simpy call the stored users logout() function, and not perform any redirects. Back buttons return the user to the landing page. In the settings page, a multitide of ADA options could be displayed and selected via checkboxes (CBs), as specified in section 3 of this document. Note that the large central map component of the home page displays map data (updated via the render() function). This includes points of interests and ADA warnings on the currently selected route, which would appear as different pins. Beyond that, it would also display any queried routes as a line across the map following the list of locations stored in that route. All routes appear grayed out, except the currently selected route, which appears in blue. Selecting a grayed-out map would change the selected route to that route, updating the ADA warning markers as necessary. Refer to the above diagram, combined with the state descriptions in section 3, to determine specific operations performed by a given button / checkbox.

# **Appendices**

## **Glossary**

Controls: Plans for how to interact with security threats. Deployment Diagrams: kind of structure diagram used in modeling the physical aspects of an object-oriented system. Feasibility Assesment: How freassible a system is. Hardware: the external and internal devices and equipment that enable you to perform major functions such as input, output, storage, communication, processing, and more. Software: the programs and other operating information used by a computer. Threats: Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation),

## **References / Bibliography**

1.  Visual Paradigm. "What Is Deployment Diagram?" Visual Paradigm, <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>. Accessed 6 June 2024.
2.  SolarWinds. "Computer Hardware Definition." SolarWinds, <https://www.solarwinds.com/resources/it-glossary/computer-hardware#:~:text=and%20diagnostics%20methods-,Computer%20Hardware%20Definition,computer%20hardware%3A%20external%20and%20internal>. Accessed 6 June 2024.
3.  "Software noun - Definition, Pictures, Pronunciation and Usage Notes." Oxford Advanced Learner's Dictionary, [https://www.oxfordlearnersdictionaries.com/us/definition/english/software#:~:text=%2Fˈsɔːftwer%2F,computer%20for%20doing%20particular%20jobs](https://www.oxfordlearnersdictionaries.com/us/definition/english/software#:~:text=%2F%CB%88s%C9%94%CB%90ftwer%2F,computer%20for%20doing%20particular%20jobs). Accessed 6 June 2024.
4.  "Threat." NIST Computer Security Resource Center, <https://csrc.nist.gov/glossary/term/threat#:~:text=Any%20circumstance%20or%20event%20with,and%2For%20denial%20of%20service>. Accessed 6 June 2024.