#### Software Engineering Project

**Final Report**

for

**AccrediLink: Matching Databases and Algorithms for Home Health Providers**

*December 6th, 2019*

**Team members**

Paul Espinoza *pxe150330*

Neda Khakpour *nxk144430*

John (Jack) Lasater *jbl160130*

#### 

#### 

#### **EXECUTIVE SUMMARY**

Throughout the nation, many have had to rely on family members to be both the provider and coordinators of home health care needs for our critically sick, elderly, and disabled loved ones, this approach is flawed and needs a viable alternative. Credentialing and background checking is a time consuming and expensive task that home health agencies must perform. This frequently leads to proper verification not being done, and non-credentialed persons being hired to take care of loved ones in home settings, creating significant risk for all parties involved. As the population expands and ages, home health care providers and organizations need the ability to keep pace with the increasing demand.

This report describes the Senior Design Project that was created as a solution to keep pace with the growing demand. Designed and implemented as part of the University of Texas at Dallas’ Fall 2019 Software Engineering Project, and sponsored by Accredilink Health, the product is the matching functionality to a membership-based social media technology for healthcare professionals and organizations who employ them. It addresses the industry needs for faster and more efficient job fulfillment, credential verification, and background verification. It consists of a web application made up of an Express backend and React frontend, and MongoDB database. This report describes the management plan, design, implementation, testing and usage of the product.

‘

#### **TABLE OF CONTENTS**

**EXECUTIVE SUMMARY 1**

**LIST OF FIGURES 4**

**LIST OF TABLES**   **5**

**1. INTRODUCTION 6**

1.1 Purpose and Scope

1.2 Product Overview

1.3 Structure of the Document

1.4 Terms, Acronyms, and Abbreviations

**2. PROJECT MANAGEMENT PLAN**

2.1 Project Organization

2.2 Life Cycle Model Used

2.3 Risk Analysis

2.4 Hardware and Software Resource Requirements

2.5 Deliverables and Schedule

2.6 Monitoring, Reporting, and Controlling Mechanisms

2.7 Professional Standards

2.8 Evidence of Configuration Management

2.9 Impact of the Project on Individuals and Organizations

**3. REQUIREMENTS SPECIFICATIONS**

3.1 Stakeholders for the System

3.2 Use Case Model

3.2.1 Graphic Use Case Model

3.2.2 Description of Use Cases

3.3 Rationale for Use Case Model

3.4 Non-functional Requirements

**4. ARCHITECTURE**

4.1 Architectural Style Used

4.2 Architecture Model

4.3 Technology, Software, and Hardware Used

4.4 Rationale for Architectural Style and Model

**5. DESIGN**

5.1 GUI Design

5.2 Static Model (Class Diagram)

5.3 Dynamic Model (Sequence Diagram)

5.4 Rationale for Detailed Design Model

5.5 Traceability from Requirements to Design Model

**6. TEST PLAN**

6.1 Requirements/Specifications-based System Level Test Cases

6.2 Traceability of Test Cases to Use Cases

6.3 Techniques for Test Generation

6.4 Assessment of Test Suite Used

**ACKNOWLEDGEMENT**

**REFERENCES**

**LIST OF FIGURES**

Figure 2.1.1 - Page 6

Figure 2.2.1 - Page 7

Figure 3.2.1.1 - Page 15

Figure 4.1.1 - Page 19

**LIST OF TABLES**

Table 2.3.1 - Pages 8, 9

Table 2.5.1 - Page 10

Table 2.5.2 - Page 10

Table 2.6.1 - Page 11

Table 2.6.2 - Page 11

Table 3.4.1 - Pages 18, 19

Table 5.4.1 - Page 25

Table 6.1.1 - Pages 26, 27, 28

Table 6.2.1 - Pages 29, 30

Table 6.4.1 - Pages 31 - 38

**1. INTRODUCTION**

**1.1 Purpose and Scope**

* Purpose: the purpose of this document is to thoroughly explain the structure, plan, design,implementation, and results of the project.
* Scope: includes details about: the project management plan,requirements specifications,architecture, design, and test plan of the project

**1.2 Product Overview**

Accredilink’s Matching Program will allow Healthcare agencies and providers to find and connect with each other with the assurance of Accredilink’s credentialing and verification. This system will allow agencies to search for specific Providers that are already credentialed by Accredilink allowing quicker and more secure hiring by removing the need for the agency to verify the provider.

**1.3 Structure of Document**

The sequence of this document begins with the executive summary, project management plan,requirement specifications, architecture, design, test plan, acknowledgement and references. This structuring of the plan allows the reader to more quickly access specific information about the document.

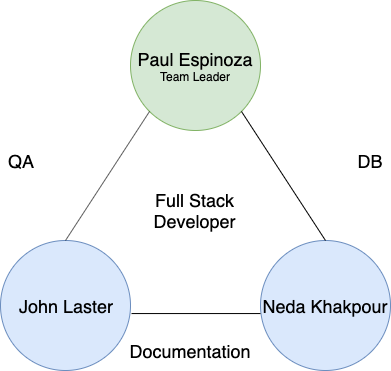
**1.4 Terms, Acronyms, and Abbreviations**

* Provider:
  + A person that provides health care or home health services, this includes both standard healthcare workers such as Physical Therapists, Registered Nurses, etc. Also includes individuals willing to offer home health services that do not require certifications.
* Agency:
  + A home healthcare agency, is a corporation that employs healthcare professionals to provide home health care services to customers who have a patient in need of the services.

**2. PROJECT MANAGEMENT PLAN**

**2.1 Project Organization**

The project organization allows for the facilitation and coordination of tasks between individuals, promoting interactions without conflict [1]. This project is split into two major parts in terms of the involved members: the project team and the project board.



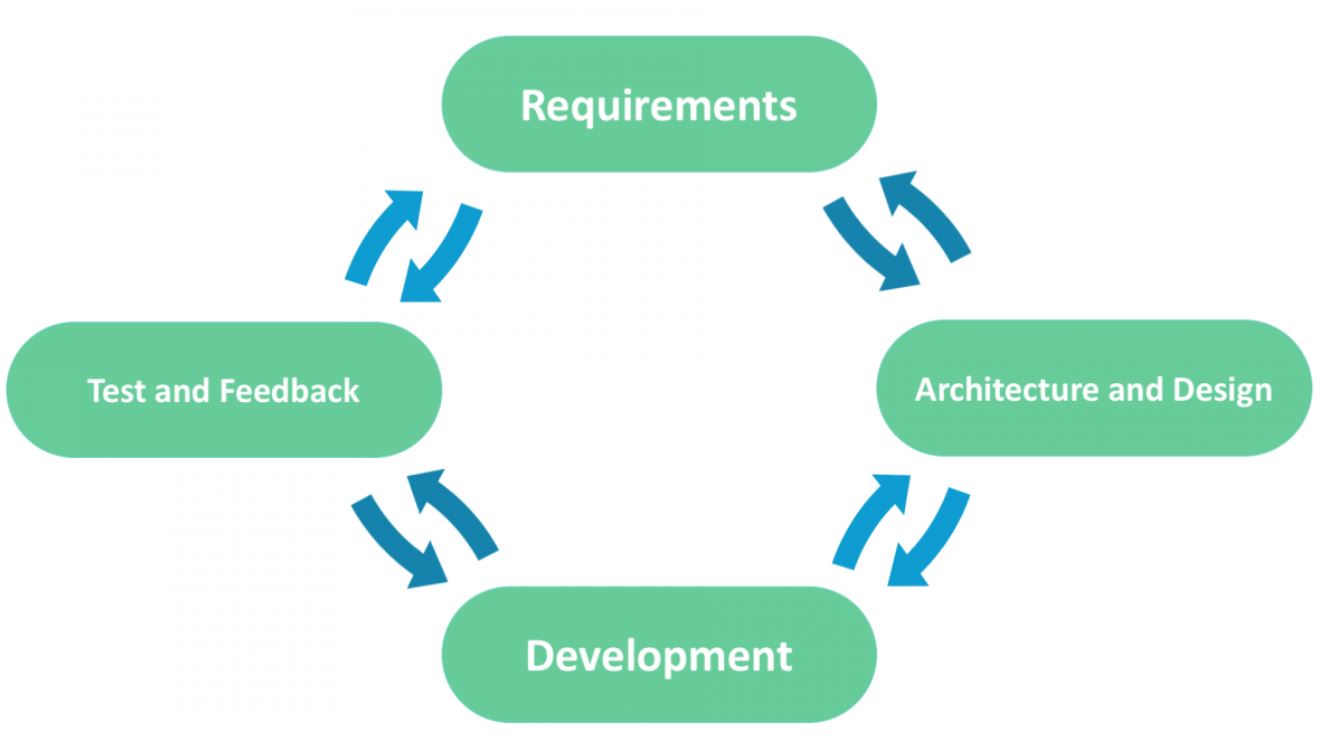
*Figure 2.1.1: Project Organization Chart*

The project team consists of Paul Espinoza, Neda Khakpour, and John Lasater. These members are responsible for the project management, requirements, architecture, design, implementation, and testing of the product. The team leader, Paul Espinoza, ensures deadlines are being met and manages the phases of the project. Every member is part of the development team. There are three teams that each have two members, the Documentation, Database, and QA teams shown above.

The project board is composed of AccrediLink management and employees, serving as the customer. These members are responsible for defining the desired product outcome for the project team to implement. The board interacts with the team in order to answer questions, provide advice, and make decisions regarding the project. The following list associates each member of the AccrediLink project board with their appropriate title:

* **Ralph Burns**, President & CEO
* **George Gundling**, Founder & Chief Solutions Architect
* **Robert Andreyka**, Director of Product Development
* **Boris Siperstein**, Chief Financial Officer
* **Baynes Eapen**, Senior Infrastructure Engineer
* **Paul Espinoza**, Project Leader & Project Development
* **Neda Khakpour**, Project Development
* **John Lasater**, Project Development

**2.2 Life Cycle Model Used**

****

*Figure 2.2.1: Agile Model*

An Agile approach will be implemented throughout the course of this project. Through this methodology, the development process will be divided into short iterations involving collaborations between the project team and project board. Agile life cycles allow for flexible response to change and adaptive planning efforts towards all phases of the project.

Similar to Scrum, daily checks and weekly meetings will be held. Weekly meetings will take place each **Wednesday at 2:00 PM**. The layout of the meetings is as follows:

* Weekly Meetings
  + Project progress discussion
  + Issue/conflict resolution
  + Define next steps
  + Create plan for upcoming week of work
* Daily Checks
  + Progress update
  + Define work to be done for next day

**2.3 Risk Analysis**

In order to avoid complications within a project, risk analysis allows for potential risks to be identified for prevention and reduction of unwanted circumstances. By evaluating the likelihood and determining strategies, the project team is able to be prepared for each case.

|  |  |  |  |
| --- | --- | --- | --- |
| **Possible Risk** | **Likelihood** | **Reduction Strategy** | **Action** |
| Inappropriate task delegation | Moderate | Impact prevention and reduction | Create detailed schedule, revise schedule |
| Inability to meet deadlines | Moderate | Impact prevention | Schedule tasks realistically, strict adherence to deadlines |
| Misunderstood customer requirements | Moderate | Impact prevention and reduction | Design review with customer, revise requirements |
| Absence of team member | High | Impact reduction | Inform team members in advance, review weekly meeting log, reschedule team meeting |
| Disagreement within project team | Moderate | Impact reduction | Appoint team leader for mediation, report if necessary |
| Inability for all team members to meet | Moderate | Impact prevention and reduction | Rescheduling, changing method of communication |
| Lack of responsibility towards tasks | Moderate | Impact prevention and reduction | Enforce daily checks and work deadlines, request for reassignment of work if necessary |
| Lack of knowledge towards software and tools | High | Impact prevention | Allocate time to learn about software and tools prior to development |
| Unreliable chosen solution | Moderate | Impact reduction | Reinforce design, revise if necessary |
| Numerous defects | Moderate | Impact reduction | Review and revise code |
| Loss of unsaved work or data | Low | Impact prevention | Using multiple methods for back-ups to be saved on and version control |

*Table 2.3.1: Risk Analysis Table*

**2.4 Hardware and Software Resource Requirements**

The search system we are designing shall be able to run on all operating systems and hardware. As a result, there will be no hardware resource requirements. There are several software requirements, we shall continue using technologies chosen in previous sections of the project.

Firstly, for building the backend for the search interface, we shall use Express. Express is a server-side web and mobile application framework. Express was chosen because it provides developers with a robust set of backend functions, a variety of HTTP utility methods and middleware to setup a dynamic API [3], as well as the flexibility to build either single-page, multi-page or hybrid, cross-platform web apps [4]. The system shall also feature a frontend for users to interact with. The frontend shall be built using React, a JavaScript library for building user interfaces. React was chosen for multiple reasons. Those reasons include, the ability to create user-friendly and intuitive UI and the small learning curve to use React with JavaScript experience [5].

Finally, our web application’s web server will be hosted using Amazon Web Services. AWS allows scalability, charges you for only the resources you use, and does not include upfront costs or long-term contracts [6]. Additionally, Amazon provides all the physical security and most of the data and network security for AWS.

**2.5 Deliverables and Schedule**

|  |  |
| --- | --- |
| **Deliverable** | **Deadline** |
| Project Management Plan | 09/06/2019 |
| Requirements Documentation | 09/20/2019 |
| Architecture Documentation | 10/04/2019 |
| Detailed Design Documentation | 10/25/2019 |
| Testing Plan | 11/15/2019 |
| Final Project Report | 12/06/2019 |
| Final Project Demonstration | 12/04/2019 |

*Table 2.5.1: Project Deliverable Schedule*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Duration** | **Start Date** | **End Date** | **Delegated Member(s)** |
| **1**. Planning | 13 days | 8/23/2019 | 9/6/2019 | **Neda Khakpour** |
| **2.** Requirements and Design | 14 days | 9/6/2019 | 9/20/2019 | **Neda Khakpour**, Paul Espinoza |
| **3.** Implementation |  | | | |
| *Implementation: Phase I* | 26 days | 9/20/2019 | 10/16/2019 | **Paul Espinoza**, Jack Lasater |
| *Testing: Phase I* | 10 days | 10/16/2019 | 10/26/2019 | **Jack Lasater**, Neda Khakpour |
| *Implementation: Phase II* | 23 days | 10/26/2019 | 11/18/2019 | **Paul Espinoza**, Jack Lasater |
| *Testing: Phase II* | 11 days | 11/18/2019 | 11/29/2019 | **Jack Lasater**, Neda Khakpour |
| **4**.Product Delivery | 4 days | 11/29/2019 | 12/3/2019 | Paul Espinoza, Neda Khakpour, Jack Lasater |

*Table 2.5.2: Task Delegation*

**2.6 Monitoring, Reporting, and Controlling Mechanisms**

To monitor the work and progress of the project, the project team utilizes collaboration tools to have effective communication and appropriate delegation of tasks. In addition, control mechanisms are implemented for the storage of documentation and programs. The table below shows the outline of the tools used, along with a short description.

|  |  |  |
| --- | --- | --- |
|  | **Tool** | **Description** |
| Monitoring | Trello | Organization of tasks |
| Microsoft Teams | Communication between project team and project board |
| GroupMe | Communication between project team members |
| Controlling | Google Drive | Storage of reports and  documentation |
| Github | Storage of programs |

*Table 2.6.1: Monitoring and Controlling Tool Table*

The following Management Report Chart organizes the reports that will be produced throughout the time span of the project. A description of the contents is provided, along with when to expect the report.

|  |  |  |
| --- | --- | --- |
| **Reports** | **Report Description** | **When?** |
| Status Report | Up-to-date log or summary of current standing on project | Weekly |
| Request for Review Report | Request input from project board regarding completed work | End of phase I & II (implementation) |
| Project Deliverable Reports | Project deliverable documentation (as assigned) | Based on deadline |
| Milestone Report | Completed phase information with deliverables attached | End of each major task |
| Issue or Bug Report | Listed issues with description that need to be examined by project board | As needed |

*Table 2.6.2: Management Report Chart*

**2.7 Professional Standards**

Adherence to professional standards can create a culture in which productivity is encouraged, and every team member is free to contribute ideas. A professional workplace enables employees to create better products, which helps drive the company forward. Adherence to standards also enables professional business relations on the corporation and individual level. A few specific standards are outlined below:

ACCOUNTABILITY

Team members are accountable for the quality of their work, and should be proud of the solutions they deliver. No one tries to deflect blame or make excuses for shortcomings in the deliverables.

PROFESSIONAL GROWTH

All team members should be constantly pushing themselves to be better engineers. Team members should always be trying to learn new skills and improve skills they already have.

DELIVER RESULTS

Team members will remain focused on delivering high quality solutions that meet the requirements of the client in an efficient way.

CONFLICT MANAGEMENT

Team members are expected to voice concerns in a respectful way when they find them. Any conflicts that arise will be handled in a fair and democratic manner.

EFFICIENCY

All deliverables are expected to run using minimal space and time constraints. Additionally, team members are expected to develop efficiently so as to get products delivered on time.

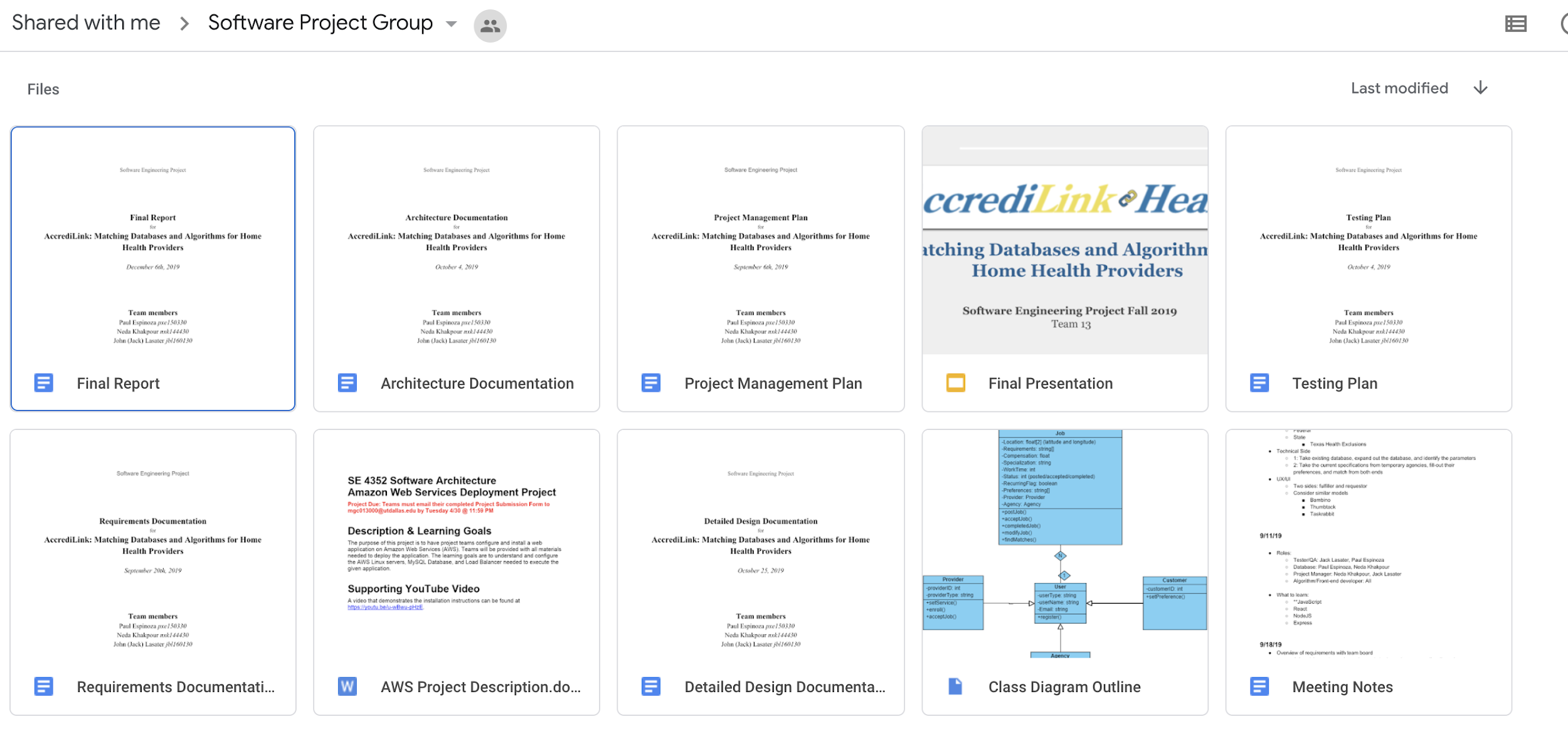
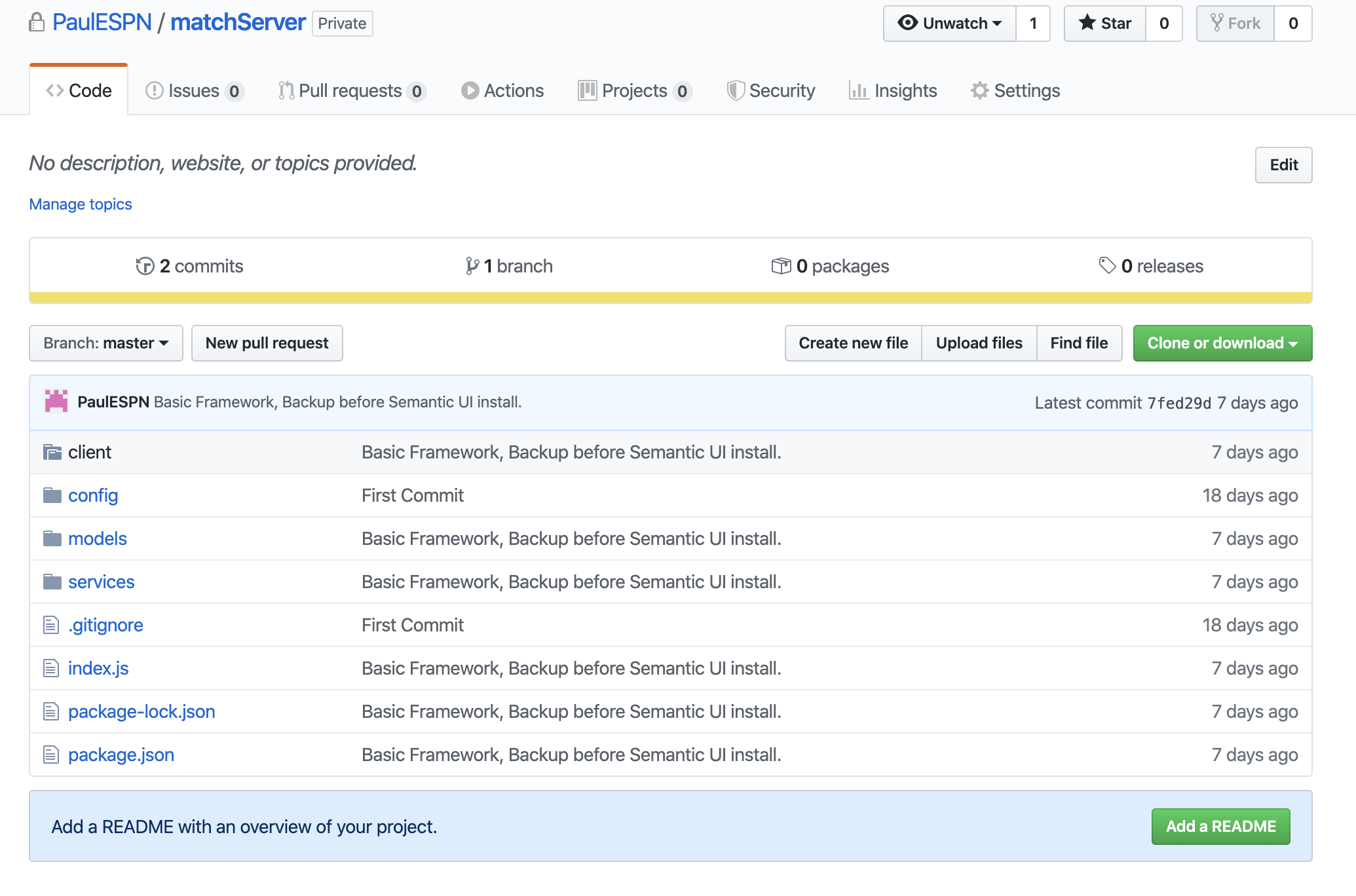
HIGHEST STANDARDS

The team members will have very high standards, which will drive them to deliver the highest quality products they can.

COMMUNICATION SKILLS

Team members will communicate with one another and with the client using professional language. All communications will be respectful for each other's time and treat others as equal.

**2.8 Evidence of Configuration Management**

****

**2.9 Impact of the Project on Individuals and Organizations**

The impact of the Accredilink Match program will allow home healthcare agencies to search for credentialed providers that have up to date documentation allowing them to immediately start working without the uncertainty of delayed background checks.

This search and quicker onboarding will speed up the process of providers finding work and agencies fulfilling customer’s needs.

**3. REQUIREMENTS SPECIFICATIONS**

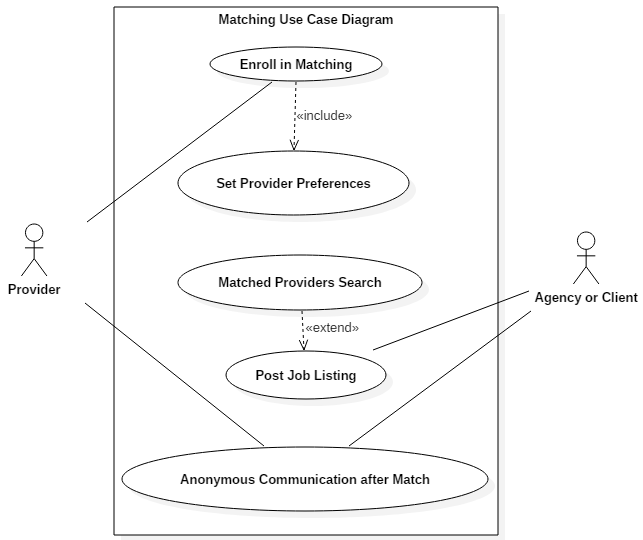
**3.1 Stakeholders of the System**

Stakeholders for the system include:

* Certified Healthcare Professionals
  + Benefit from the verification and ongoing monitoring of all licenses, credentials, and policies.
  + Allows them to easily find work through any Accredilink partnered agencies.
* Home healthcare agencies, in-home care agencies, nursing homes and hospices
  + Eliminates the costly and tedious work of collecting and tracking credentials.
  + Allows agencies to focus on the core business of providing critical services to patients when needed.
  + Allows for continuous verification and monitoring of all associated resources
  + Provides the ability to identify and interview pre-screened healthcare personnel before hiring.
* Unpaid caregivers
  + Benefit from quick and easier access to employers
  + Benefit from the ability to more easily find part time or occasional opportunities to help another family.
  + Benefit from the ability to easily demonstrate their credentials, skills, location and availability.
  + Benefit from the ability to get accreditation and training, making them more able to demand a higher wage.
  + Benefit from being able to be put on a payroll for the routine care of their loved ones.
  + Benefit from being able to be properly background checked.
* Patients and families
  + Allows families to more easily find qualified caregivers.
  + Alleviates the financial burden and time commitment associated with families conducting their own pre-employment, rudimentary background checks.
  + Family can have more comfort regarding the qualifications and capabilities of the caregivers who attend to their loved one.
  + Families can now have access to the background of the caregiver.

**3.2 Use Case Model**

**3.2.1 Graphic Use Case Model**

****

*Figure 3.2.1.1: Use Case Model*

**3.2.1 Description of Use Cases**

* Enroll in Matching:
  + **Participating Actors** - Provider
  + **Entry Condition(s)** - Provider must be logged into Accredilink application and navigate the the match enrollment.
  + **Normal Flow of Events** - After a provider logs into the application, they can select to enroll into matching.
  + **Exit Condition(s)** - Once they agree to enroll, the provider is taken to set their preferences.
  + **Exceptions** - If the provider does not agree to enroll, they will not be taken to the preferences page
  + **Special Requirements** - To access the enrollment page for the match program the provider must be accredilink verified.
* Set Provider Preferences:
  + **Participating Actors** - Provider
  + **Entry Condition(s)** - On enrollment into the matching program the provider is taken to the preferences page or if the user navigates to the preferences page after being previously enrolled.
  + **Normal Flow of Events** - A provider inputs their preferences for geography, time, speciality, etc. Then saves their preferences which will update the matching database and allow them to be matched.
  + **Exit Condition(s)** - Once the provider saves their preferences they will be taken back to the main application.
  + **Exceptions** - N/A
  + **Special Requirements** - The provider must be enrolled into the matching program.
* Matched Providers Search:
  + **Participating Actors** - Client
  + **Entry Condition(s)** - A client has posted a job in the system.
  + **Normal Flow of Events** - The system will search through the database of providers for those that meet the job criteria. The system shall then inform the matching providers for them to review.
  + **Exit Condition(s)** - The providers that were found are given an overview of the job to review and determine if they would like to take it.
  + **Exceptions** - If no providers match search criteria, the searching client will be notified.
  + **Special Requirements** - N/A
* Post Job Listing:
  + **Participating Actors** - Agency/Client
  + **Entry Condition(s)** - Client must be logged into Accredilink application
  + **Normal Flow of Events** - The client creates a job posting with details of the opportunity and requirements/preferences for potential providers.
  + **Exit Condition(s)** - Once the job posting is created the client approves it before it moves onto the rest of the system.
  + **Exceptions** - The client will not be able to submit a job posting if they are not logged in.
  + **Special Requirements** - N/A
* Anonymous Communication after Match:
  + **Participating Actors** - Provider and Agency/Client
  + **Entry Condition(s)** - Both participating actors have simultaneously matched based on their preference criteria.
  + **Normal Flow of Events** - The Provider and the Agency/Client will have access to communicate with one another without their identity being disclosed.
  + **Exit Condition(s)** - The Provider and Agency/Client have the ability to contact one another anonymously from both ends
  + **Exceptions** - The identity of each participating actor may be revealed at the discretion of the individual themself.
  + **Special Requirements** - Each participating actor must be logged in in order to communicate with the other party.

**3.3 Rationale for Use Case Model**

The two main parties in the use case model are the provider and the agency/client. The client is the party that is interested in receiving home-health services, and the provider is the home-health worker that is being considered.

The provider enrolls in matching, which means that they register with Accredilink to be entered into the provider database. Once registered, the provider can also set their provider preferences to indicate things like preferred working days, specialty, or any accreditations/qualifications.

The client uses the system to post jobs for providers. The client can specify job criteria in order to narrow potential matches. Additionally, clients can specify preferences in order to try and find providers that meet certain criteria without discarding those that do not.

Once a job is posted, providers can view the job if they meet criteria, and choose to accept the job if they so choose. After posting a job, the client will get a list of providers that accepted the job, and will be able to choose one to offer the job to.

Once a job has been offered, the client and provider are given an anonymous channel through which they can communicate. During this communication, the client and provider can get to know each other, and discuss any complications that were not stated in the job posting and/or provider description.

After a successful match, the details of the job should be logged so the data can be used by the Accredilink company to make business decisions in the future.

**3.4 Non-functional Requirements**

|  |  |
| --- | --- |
| 1 | The system shall be available for use 24 hours a day and each day of the week. |
| 2 | The system shall never allow a Social Security Number (SSN) to be viewable at any point in time, including during entry. |
| 3 | The system shall never disclose any patient information to a provider or agency, unless a signed release form which authorizes permission is provided. |
| 4 | The system shall have a load time of less than 10 seconds for all actions. |
| 5 | The system shall have a restart cycle time in less than 60 seconds. |
| 6 | The system shall only allow access permissions to be changed by the system data administrator. |
| 7 | The system shall record all timestamps in UTC (Universal Time Coordinated) when placed in permanent storage. |
| 8 | The system shall be accessible to individuals with disabilities in accordance with the American Disabilities Act of 1990. |
| 9 | The system shall be able to accomodate to increased volumes of users or work loads. |
| 10 | The system shall be compatible with all major web browsers |
| 11 | The system shall never give out caregiver information without consent |
| 12 | The system shall not show caregivers that have not indicated they are looking for matches |
| 13 | The system shall send email/phone notifications within one minute of the user requesting them to be sent |
| 14 | The system shall sanitize inputs to prevent SQL injection attacks |
| 15 | The system shall store user passwords in a secure way, to prevent account breaches |

*Table 3.4.1: Non-functional Requirements Table*

**4. ARCHITECTURE**

**4.1 Architectural Style Used**

****

*Figure 4.1.1: MVC Diagram*

The system will use a **Model-View-Controller** (MVC) architecture style which is linked to the user. The MVC is separated into three components: Model, View, and Controller [1]. The user makes an input request to the Controller. The Controller accepts the input request and converts it into an action for the Model or View components. The Controller is considered the interface between the Model and View, containing a combination of front-end and back-end platforms. Requests and responses of data are conducted between the Controller and the Model. The Model components is responsible for the data maintenance and logic of the system, interacting directly with the MongoDB database. Node and Express will work interchangeably as the back-end scripting languages for this system. Express is an extension of the Node framework with additional features. The View component is created by the collection of data processed in the Controller, which is provided by the Model. This component presents the output data to the user as a response. React, a front-end JavaScript platform, makes up the View component of this architecture style.

As a whole, the MVC architecture supports the application features of the project by providing a clear separation of front-end and back-end logic. The components are connected through an interface, allowing the system to function as a whole. This style allows the program to be easily readable, while changes to the code do not affect the entire architecture. In addition, the MVC supports a rapid development process as multiple programmers can work on different components of the architecture at once.

**4.2 Architectural Model**

Subsystems:

* + Front End: React
  + Back End: Express and Node.js
  + Database: MongoDB

**4.3 Technology, Software, and Hardware Used**

The system will use MongoDB databases for information storage. The backend will consist mainly of Java API calls to interact with the databases, and the frontend will be based in the React framework. Each team member will be free to use the IDE of their choice, and code will be shared via GitHub. Documentation will be kept on Google Drive and on Microsoft Teams, where it will be shared with Accredilink representatives. The application databases will be hosted on Accredilink hardware.

**4.4 Rationale for Architectural Style and Model**

*Architectural Style*

In regards to the architectural style, the Model-View-Controller (MVC) is the designated pattern for the system. The MVC architecture emphasizes the separation of front-end and back-end logic, which interconnect through an interface. The matching algorithm consists of multiple services between the provider and agency/client. The data presentation component of the system plays a major role in the development phase due to high user involvement.In relation to the MVC architecture, the front-end of the system corresponds directly to the View component. In addition to the View, the back-end platform must be able to manage, search, and match large amounts of data from the database. This is a vital aspect of the system, as the back-end corresponds directly to the Model component of the MVC architecture. In addition to the Controller interface, equal importance is held for both front-end and back-end platforms to correctly function. Since the matching algorithm places significance on these platforms, the MVC architecture suits the system. With the separation of components, programming changes do not affect the entire architecture. Programmers have the ability to work on different components simultaneously, resulting in a faster development process This is useful, as time is limited for the completion of the project. With separation of components, flexibility, and rapid development, the Model-View-Controller architecture style is the best fit for this software system.

*Architectural Model*

The provider and agency/client are the main external parties interacting with the system. They directly interact with the Angular frontend client that uses API calls to perform services. In the service layer, the API calls are connected to and interact with databases. These interactions include: adding jobs to active job database, adding matched jobs to match success database, and searching databases for jobs matching given criteria. Databases needed for this project include: jobs actively seeking providers, providers seeking active jobs, and completed jobs, which are recorded for future business purposes.

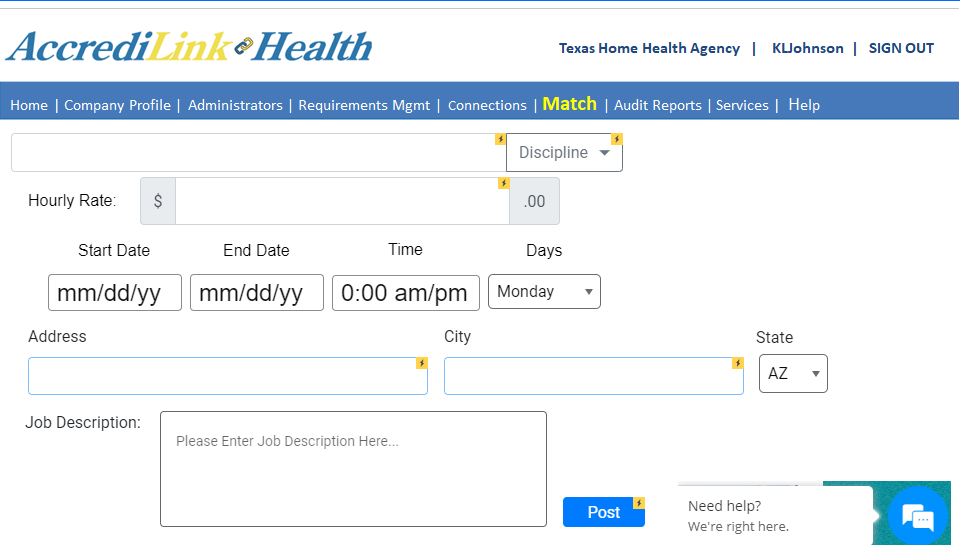
**5. DESIGN**

**5.1 GUI Design**

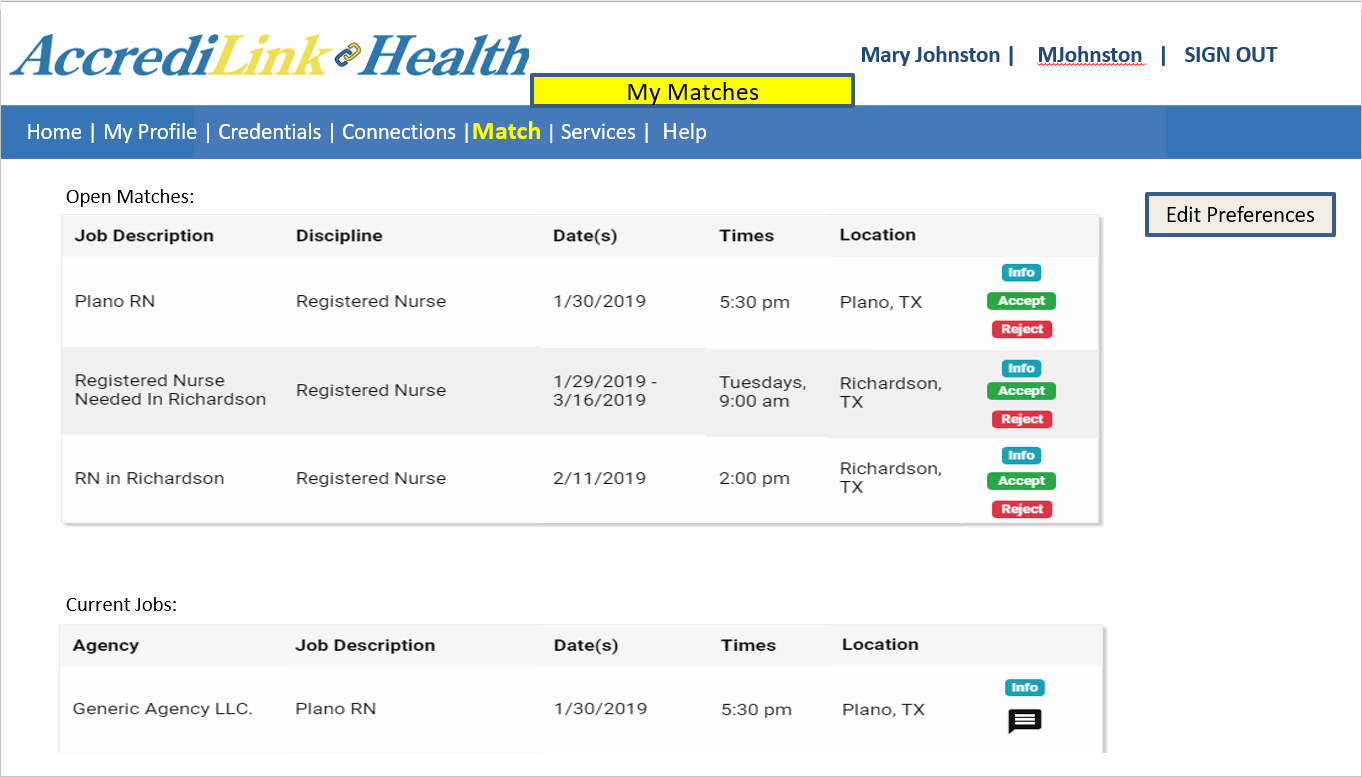
**Agency Landing Page**



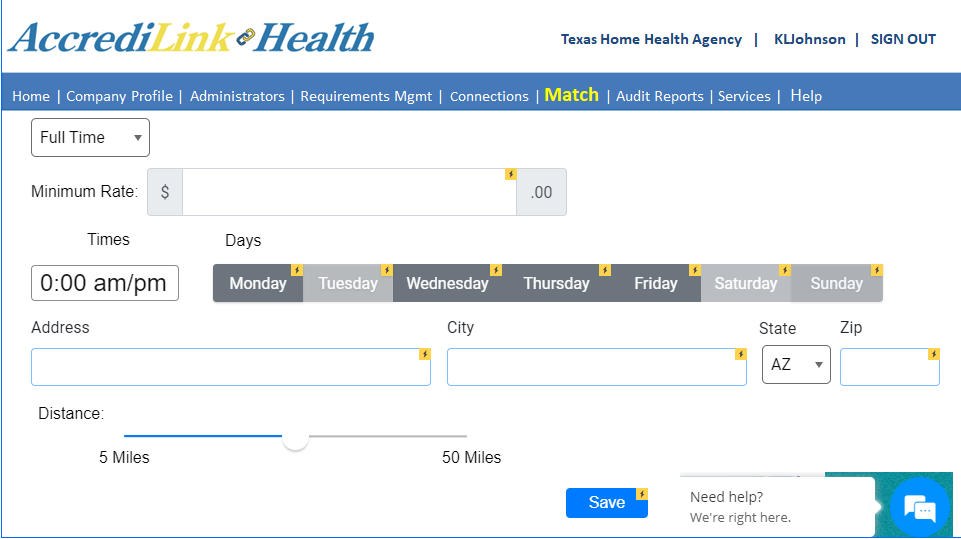
**Agency Job Posting Page**



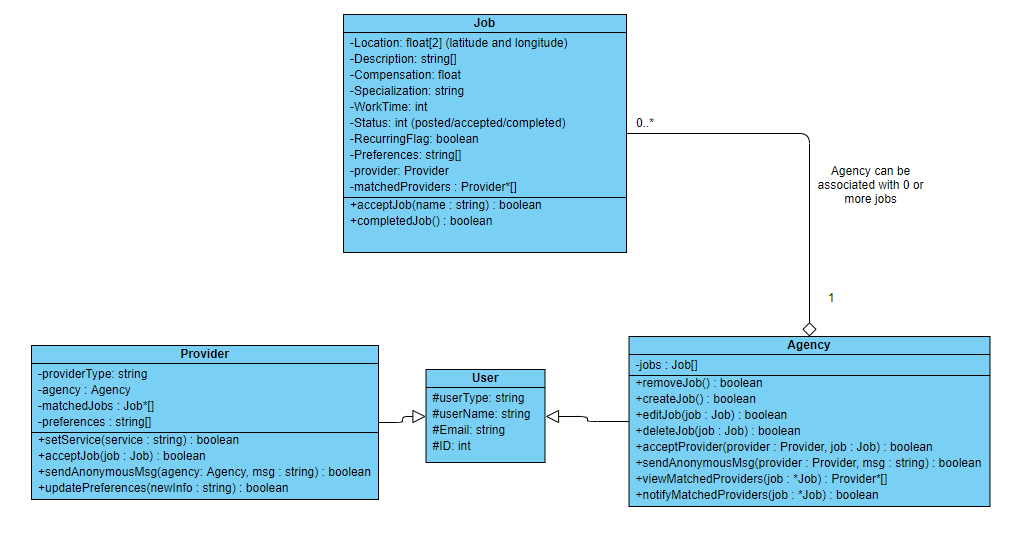
**Provider Landing Page**



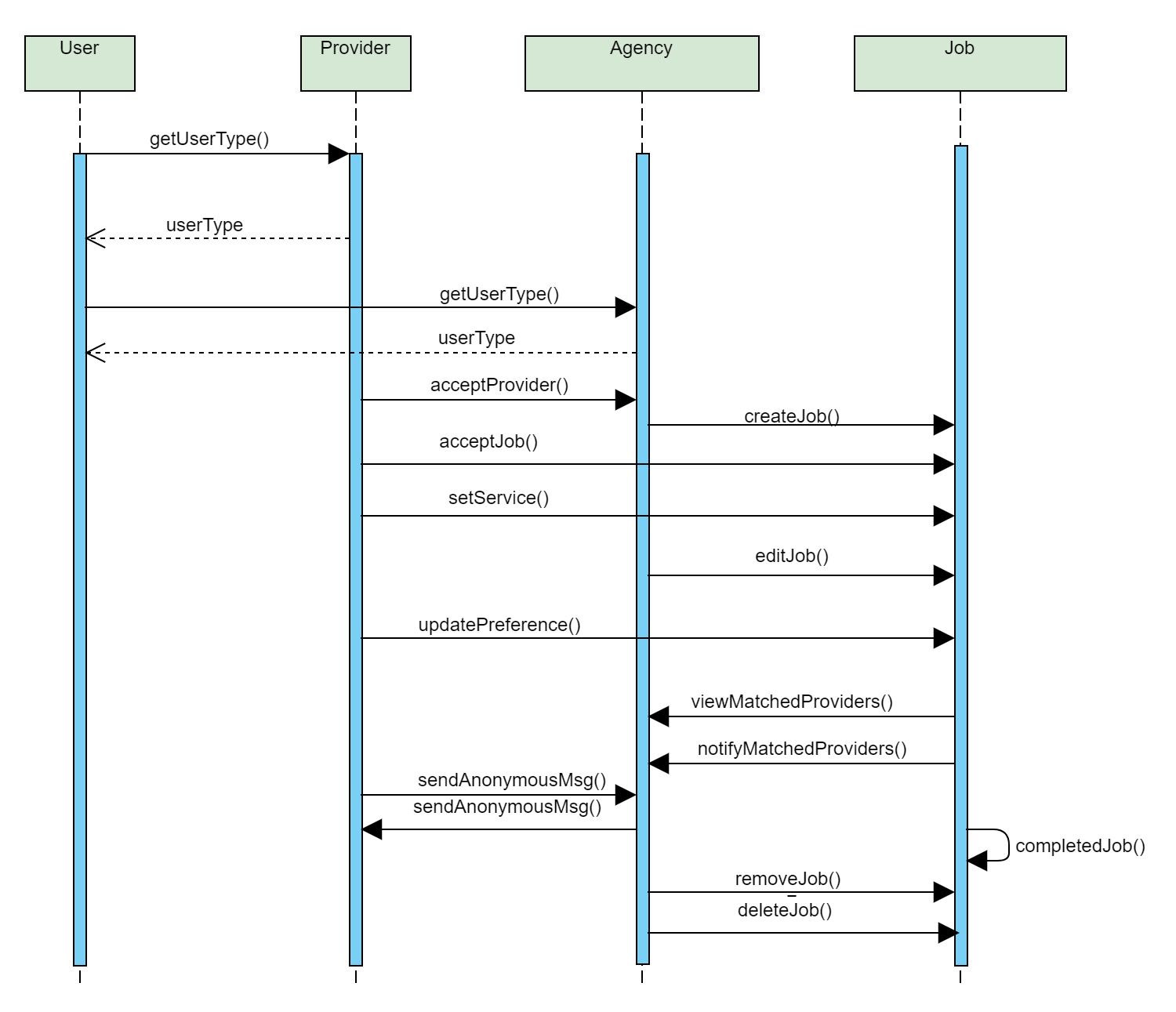
**Provider Preference Page**



**5.2 Static Model**

****

**5.3 Dynamic Model**

****

**5.4 Rationale for Detailed Design Model**

The static model is centered around the job object. This represents a job posted by an agency for a provider to fulfill. The job object contains relevant information describing the job, and has methods to change the job status. This match method is the core functionality our system performs. The other major class in the system is User, which has two subclasses: Provider and Agency. All users have a user type, name, and email. On top of this, agencies (who represent the people receiving care) can create and remove jobs, and providers can edit their information and accept jobs. Agencies can use the viewMatchedProviders() method to generate a list of matched providers. Once the Agency is ready, they can inform matched providers that they matched for the job, and it will be up to the providers to accept the job or not. This matching functionality is the core of our project. Once a job is matched, providers can accept the job. After a certain amount of time, the agency will be able to accept a provider. Once an agency accepts a provider, they will gain the ability to send anonymous messages to each other in order to work out details and establish a better connection.

The dynamic model shows the interaction between the object methods and the objects. The main actor role is taken by the User object. The additional actor roles consist of the Provider and the Agency. Similar to the static model, the diagram is centered around the focal point of the Job. Thus, the interactions between the Provider and Agency are accomplished through the Job object. Each method is shown by the use of an arrow between the involved actors. A straight arrow denotes an interaction between the sender to the receiver, while a dashed arrow denotes an interaction between the receiver to the sender. The events are portrayed in the order in which they are predicted to occur.

**5.4 Traceability from Requirements to Design Model**

The class diagram features methods that correspond to functional requirements. The below table shows functional requirements and their corresponding methods in the class diagram:

|  |  |
| --- | --- |
| **Functional Requirement** | **Class Diagram Method** |
| **R.1:** System shall allow agencies (on behalf of clients) to create a job posting online. | Agency.createJob() |
| **R.1.1:** System shall allow job posters to view anonymized profiles of providers with matching preferences. | Agency.viewMatchedProviders() |
| **R.1.2:** System shall allow providers to accept or decline job posting they are matched to. | Provider.acceptJob()  Note: This method is used to accept or decline job |
| **R.2:** System shall allow providers to set and edit preferences for matching purposes. | Provider.updatePreferences() |
| **R.2.1:** System shall maintain a DB entry of preferences for each provider | Provider.updatePreferences()  Note: when preferences are updated, database entry of provider is updated. Database entry is created when provider is created. |
| **R.3:** System shall allow providers and posters to connect anonymously for communication purposes after being matched. | Provider.sendAnonymousMsg() and Agency.sendAnonymousMsg() |
| **R.4:** System shall record accepted matches and unfilled jobs, and store in the database. | Agency.acceptProvider() adds job to database of accepted matches.  Agency.deleteJob() adds job to unfilled jobs database if Job.status == posted |

*Table 5.4.1: Requirements to Design Model*

**6. TEST PLAN**

**6.1 Requirements/Specifications-based System Level Test Cases**

Note - The use cases have been numbered as follows:

UC01 - Modify provider preferences

UC02 - Matched providers search

UC03 - Post job listing

UC04 - Anonymous communication after match

|  |  |  |
| --- | --- | --- |
| **Test Case Number** | **Description** | **Use Case Numbers** |
| TC01 - TC04 | Test updating preferences with valid inputs for each field. Numbering is as follows:  TC01 - location  TC02 - hours  TC03 - rate  TC04 - maxDistance | UC01 |
| TC05 - TC08 | Test updating preferences with empty input. One test case per field. Numbering as follows:  TC05 - location  TC06 - hours  TC07 - rate  TC08 - maxDistance | UC01 |
| TC09 - TC12 | Test updating preferences with invalid inputs. One test case per field. Numbering as follows:  TC09 - location  TC10 - hours  TC11 - rate  TC12 - maxDistance | UC01 |
| TC13 - TC16 | Test updating preferences with SQL injection inputs. Numbering as follows:  TC13 - location  TC14 - hours  TC15 - rate  TC16 - maxDistance | UC01 |
| TC17 | Verify updated preferences are reflected in database | UC01 |
| TC18 | Test trying to change someone else’s preferences to ensure it cannot be done | UC01 |
| TC19 | Test that text entered into fields is correctly formatted | UC01, UC04 |
| TC20 | Test that if preference update is cancelled, changes are not pushed through to database | UC01 |
| TC21 | Verify correct provider profile is pulled from database when request to set/update preferences is made | UC01 |
| TC22 | Verify search outputs all exact matches. | UC02 |
| TC23 | Test search when there are no providers in database | UC02 |
| TC24 | Verify match ordering works as expected | UC02 |
| TC25 | Test search with empty job input | UC02 |
| TC26 | Test search with SQL injection input | UC02 |
| TC27 | Test search with invalid input | UC02 |
| TC28 | Ensure posted jobs are put into appropriate database | UC03 |
| TC29 | Ensure newly added jobs show up in new job queries | UC03 |
| TC30 | Test adding job with valid inputs | UC03 |
| TC31 - TC36 | Test adding job with empty inputs. Numbering as follows:  TC31 - Location  TC32 - Name  TC33 - Description  TC34 - Discipline/specialty  TC35 - Rate  TC36 - Hours | UC03 |
| TC37 | Test job adding functionality without providing a job as input | UC03 |
| TC38 - TC43 | Test adding job with SQL injection in job fields. Numbering as follows:  TC38 - Hours  TC39 - Location  TC40 - Name  TC41 - Description  TC42 - Discipline/specialty  TC43 - Rate | UC03 |
| TC44 | Ensure all text in job fields is added to database | UC03 |
| TC45 | Ensure personal info of agency that posted job is not visible via the posted job | UC03 |
| TC46 | Test attempting to add several jobs at once | UC03 |
| TC47 | Test sending valid message | UC04 |
| TC48 | Test trying to send empty message | UC04 |
| TC49 | Ensure agency cannot obtain confidential information about provider | UC04 |
| TC50 | Ensure provider cannot obtain confidential information about agency | UC04 |
| TC51 | Ensure chat is persistent for participants | UC04 |
| TC52 | Ensure third parties are not able to read chat messages | UC04 |
| TC53 | Test to make sure chat is removed if job is declined or job offer is rescinded | UC04 |
| TC54 | Test to make sure messages that are deleted before being sent are not saved. | UC04 |
| TC55 | Test if user is notified if their message failed to send. | UC04 |
| TC56 | Ensure no sensitive/confidential information about providers is shown in search results | UC02 |

*Table 6.1.1: Test Cases*

**6.2 Traceability of Test Cases to Use Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Number** | **Use Case Description** | | **Test Case Numbers** |
| UC01 | Title: Modify Provider Preferences | |  |
| Basic flow | A provider that is already registered with Accredilink wishes to update their job preferences, which will affect jobs the provider matches for. The provider will navigate to the provider preference page and use the GUI there to input minimum rate, times, days, max distance, and an address for location purposes. When the provider has inputted their selection, they hit the save button, which takes the inputted data and feeds it to the API, which makes the appropriate database changes. | TC1, TC2, TC3, TC4, TC5, TC6, TC7, TC8, TC9, TC10, TC11, TC12, TC13, TC14, TC15, TC16, TC17, TC18, TC19, TC20, TC21 |
| UC02 | Title: Matched Providers Search | |  |
| Basic flow | Once a job has been posted by an agency, the system will search through the providers in the database that are currently seeking jobs. Matched providers will be made available to the agency on the My Matches screen. At the top of the matches list will be any providers that meet all the stated criteria for the job posting, if there are any. Below that, providers that meet most of the criteria for a job will be ranked from most relevant/closest to the job description to least relevant/closest to the job description according to an algorithm executed by the backend API. | TC22, TC23, TC24, TC25, TC26, TC27, TC56 |
| UC03 | Title: Post Job Listing | |  |
| Basic flow | An agency navigates to the job posting page, where they use the GUI to input relevant information about the job, including a textual description, pay, work time and hours, location, and specialty required. Once the information is entered, the agency presses the Post button, and the information is fed to the backend API that creates the job, stores it in the database, and links the job to the posting agency. | TC28, TC29, TC30, TC31, TC32, TC33, TC34, TC35, TC36, TC37, TC38, TC39, TC41, TC41, TC42, TC43, TC44, TC45, TC46 |
| UC04 | Title: Anonymous Communication After Match | |  |
| Basic flow | After the agency selects a provider as a final candidate for the job, the agency and provider are given an anonymized form that they can use to communicate. Both parties are given the ability to see certain details about the other (not personal information however) and send messages back and forth to further determine if the match would be a good fit. After the match is finalized or rescinded, the chat link between the two parties is removed. | TC19, TC47, TC48, TC49, TC50, TC51, TC52, TC53, TC54, TC55 |



**6.3 Techniques for Test Generation**

Note: We will be using Mocha, a test automation framework for Node.js, to write and perform our automated tests.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Number** | **Techniques Used for Test Generation** | **Blackbox/Whitebox** | **Criteria Used for Measuring Test Quality** |
| TC01 | Create automated test case with a valid location for API call that updates test provider | Whitebox | API response correspond to expected values and database entry for job has changes reflected in it |
| TC02 | Create automated test case with valid hours for API call that updates test provider | Whitebox | API response correspond to expected values and database entry for job has changes reflected in it |
| TC03 | Create automated test case with a valid rate for API call that updates test provider | Whitebox | API response correspond to expected values and database entry for job has changes reflected in it |
| TC04 | Create automated test case with a valid maxDistance for API call that updates test provider | Whitebox | API response correspond to expected values and database entry for job has changes reflected in it |
| TC05 | Create automated test case with an empty input for location in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC06 | Create automated test case with an empty input for hours in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC07 | Create automated test case with an empty input for rate in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC08 | Create automated test case with an empty input for maxDistance in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC09 | Create automated test case with an invalid input, such as an address on a street that does not exist, for location in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC10 | Create automated test case with an invalid input, such as a negative number, for hours in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC11 | Create automated test case with an invalid input, such as a negative number, for rate in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC12 | Create automated test case with an invalid input, such as a negative number, for maxDistance in the API call that updates a test provider | Blackbox | Error message should be given to the user, and changes should not go through |
| TC13 | Create automated test case with an input for location that attempts to use SQL injection to maliciously change the test database | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC14 | Create automated test case with an input for hours that attempts to use SQL injection to maliciously change the test database | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC15 | Create automated test case with an input for rate that attempts to use SQL injection to maliciously change the test database | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC16 | Create automated test case with an input for maxDistance that attempts to use SQL injection to maliciously change the test database | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC17 | After valid preferences are submitted as changes to a sample provider, use database queries to ensure the database has changed to reflect the new preferences | Whitebox | The database should be examined to ensure the desired changes went into effect. |
| TC18 | While a test provider is logged in, it should send an API request to change a different test provider’s preferences | Whitebox | The changes should not go into effect and an error message should be displayed |
| TC19 | Create automated test case that enters text into a field and then prints out what the system reads into different backend fields from that | Whitebox | The system should read in all valid characters and store the appropriate string in the data |
| TC20 | Create automated test case that opens page to edit preferences, enters values, but then closes page without saving. | Whitebox | The system should disregard the information that was entered and make no changes to the database |
| TC21 | Create automated test case that makes fixed, trivial update to preferences of predetermined provider, and prints the provider ID of the provider that was altered | Whitebox | The system should have made the change to the correct provider |
| TC22 | Create automated test case that runs the match function on a test job against a test database of providers and prints out the IDs of all test providers that have an exact match | Blackbox | The system should correctly output every provider that completely matches job requirements |
| TC23 | Create automated test case that runs the search when there are zero providers in the database | Blackbox | The search should return and output a message saying there were no matches found |
| TC24 | Populate a test database of providers with a handful of providers that can be manually ranked with the matching algorithm and create an automated test case that creates a job and returns a list of provider IDs it matched against in order | Blackbox | The system should correctly order the providers that partially match the job requirements |
| TC25 | An automated test case should search for matching providers with a null job | Blackbox | An error message should be displayed to the user |
| TC26 | An automated test case should create a job that has all fields containing SQL injections and run the search algorithm with it, and then manual database queries should be used to verify database integrity | Whitebox | The databases should not be impacted by the SQL code in the job fields |
| TC27 | An automated test case should try to run the search algorithm with a job that has an invalid ID, like a negative number ID | Blackbox | An error message should be displayed to the user |
| TC28 | An automated test case should create a test agency that creates a test job, which is posted, after which manual database queries can be used to determine if the job went to the correct database | Whitebox | The database of posted jobs should have the test job added, and no other databases should be impacted |
| TC29 | An automated test case should create and post a test job and then create a test provider and immediately create a search for jobs that the test job should be returned in | Blackbox | The test job should be among the list of jobs returned |
| TC30 | An automated test case should try to create a job with valid fields and then create a database query to confirm the job was created and added to the database successfully | Whitebox | The job should be returned as part of the database query’s return |
| TC31 | An automated test case should create a job with all valid fields except for location, which should be left empty. | Blackbox | An error message should appear, prompting the user to fill in the missing field |
| TC32 | An automated test case should create a job with all valid fields except for name, which should be left empty. | Blackbox | An error message should appear, prompting the user to fill in the missing field |
| TC33 | An automated test case should create a job with all valid fields except for description, which should be left empty, and then create a database query to confirm the job was created and added to the database successfully | Whitebox | The job should be returned as part of the database query’s return |
| TC34 | An automated test case should create a job with all valid fields except for discipline, which should be left empty. | Blackbox | An error message should appear, prompting the user to fill in the missing field |
| TC35 | An automated test case should create a job with all valid fields except for rate, which should be left empty. | Blackbox | An error message should appear, prompting the user to fill in the missing field |
| TC36 | An automated test case should create a job with all valid fields except for hours, which should be left empty. | Blackbox | An error message should appear, prompting the user to fill in the missing field |
| TC37 | An automated test case should attempt to add a job to the database without actually providing a job to be added | Whitebox | The database should return an error |
| TC38 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the hours field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC39 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the location field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC40 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the name field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC41 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the description field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC42 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the discipline field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC43 | An automated test case should attempt to add a job to the database with an attempted SQL injection in the rate field, and then use database queries to ensure the database was not altered by the SQL injection. | Whitebox | The database should be examined to ensure no unintended changes were made to it |
| TC44 | Input should be given in the job creation page, and then database queries should be done to ensure all input text is captured and stored within the system | Whitebox | The database should have captured all of the |
| TC45 | An automated test case should create a test agency and have it post a test job. Then, a manual search that the test job qualifies for should be done, and the results should be inspected to ensure no private info about the agency was revealed. | Blackbox | No personal/unwanted information should be visible in the search results |
| TC46 | An automated test case should attempt to add several jobs at the same time, to make sure the system can handle it | Blackbox | The system should continue to run smoothly throughout the job adding process |
| TC47 | An automated test case should try to send an anonymous message to a test entity | Blackbox | The message should arrive and be readable to the test entity |
| TC48 | An automated test case should try to send an empty message to a test entity | Blackbox | The system should not allow the message to be sent if it is empty |
| TC49 | An automated test case should create a provider, send a message to a test agency, and then the message should be inspected manually from the test agency’s point-of-view to ensure no private info about the provider is visible | Blackbox | No private/personal info about the provider should be visible to the recipient through the message |
| TC50 | An automated test case should create an agency, send a message to a test provider, and then the message should be inspected manually from the test provider’s point-of-view to ensure no private info about the sender is visible | Blackbox | No private/personal info about the agency should be visible to the recipient through the message |
| TC51 | An automated test case should create an agency and a provider, have them each send a message to each other, close the messaging page, and reopen the messaging page to ensure the messages are still visible | Blackbox | The messages should still be readable to both agency and provider |
| TC52 | An automated test case should create two providers (named A and B) and two agencies (named C and D). A and C should exchange a few messages, and then B and D should each open the message page to see if they can read A and C’s messages | Blackbox | A and C’s messages should not appear on B and D’s pages |
| TC53 | An automated test case should create and agency and provider. The created agency should create a job that the created provider is a match for. The match should be made, and the created agency and providers should exchange test messages. The match should then be cancelled, after which the created agency and provider should try to send more messages | Blackbox | After the match is cancelled, the provider and agency should no longer have the ability to directly chat back and forth |
| TC54 | A message should be entered, but not sent. Then, the messaging page should be closed and reopened. There should be no evidence of the message that was not sent. | Blackbox | The message should not remain in the textbox that is used to type messages and should not have been sent |
| TC55 | A message should be sent that contains a special test code, flagging the backend system to not forward it through the recipient. The sender should receive a notification that the message was not sent. | Blackbox | The message should not be sent, and the sender should be notified of this fact. |
| TC56 | An automated test case should create several valid test providers. A manual search should then be done and visually inspected to ensure no private information about any test provider is revealed in search results | Blackbox | No private/confidential information should be visible to the searcher |

*Table 6.4.1: Test Generation Techniques*

**6.4 Assessment of the Goodness of Test Suite**

Because the automated tests that were needed to run were on APIs, Mocha was able to meet the needs of the tests. None of the automated tests needed to be particularly complex, and Mocha was able to work just fine.

Another consequence of the tests being mainly API based is that it is relatively simple to apply test generation techniques to get a well defined set of tests. No amount of testing can prove that a system is completely bug free, but the test suite should be adequate to catch most major defects.

**ACKNOWLEDGEMENT**

**REFERENCES**