

# Medical Documentation System “HALO”

---

IS 495: CAPSTONE

Dominic V. Triano

SUNY POLYTECHNIC INSTITUTE | 100 SEYMOUR AVE, UTICA, NY 13502

# Abstract

Current medical institutions make it hard to retrieve medical information. Between fees for data retrieval, and archaic data transmissions, it is not a fast or enjoyable process. This system proposes a modern way for medical data retrieval on an interface that is minimalistic and easy to use. When this project is finished, its expansion will continue until it is a fully functioning, secure medical information system. By using a Ktor server in conjunction with Freemarker Template files, we can make a system that is versatile and expansive. To understand how Ktor and Freemarker would work, we had to investigate both features and learn about their libraries and implementations. Freemarker creates the GUI while Ktor processes all the backend transactions. By the end of this project, this system was able to accomplish the main goal of giving users a more modernized way of viewing their files, by displaying them over a Web GUI and giving users the availability to download the files. This system could be a good base or addition to existing medical systems with a couple of security advancements.

# Table of Contents

|       |  |    |
|-------|--|----|
| 1     | Chapter 1: Introduction .....                            | 1  |
| 1.1   | Project Context .....                                    | 1  |
| 1.2   | Purpose .....  | 1  |
| 1.2.1 | Description: .....                                       | 1  |
| 1.3   | The Objective of the Project .....                       | 2  |
| 1.4   | Scope .....  | 2  |
| 1.4.1 | Limitations: .....                                       | 3  |
| 2     | Chapter 2: Review of Related Literature.....             | 4  |
| 2.1   | Kotlin Resources.....                                    | 4  |
| 2.2   | Freemarker and HTML Resources .....                      | 5  |
| 3     | Chapter 3: Methodology .....                             | 6  |
| 3.1   | Project Plan .....                                       | 6  |
| 3.2   | Data Gathering Procedures .....                          | 6  |
| 3.3   | Source of Data .....                                     | 8  |
| 3.4   | Software Model.....                                      | 8  |
| 4     | Chapter 4: Results and Discussion .....                  | 8  |
| 4.1   | Requirements Analysis.....                               | 8  |
| 4.2   | Requirements Documentation.....                          | 9  |
| 4.3   | Design of Software, Product, and Process .....           | 10 |
| 4.4   | Development and Testing .....                            | 11 |
| 4.4.1 | Database Creation .....                                  | 11 |
| 4.4.2 | Ktor Server Development: .....                           | 12 |
| 4.4.3 | Freemarker File Development:.....                        | 12 |
| 4.4.4 | Testing: .....   | 13 |
| 4.5   | Description of the Prototype.....                        | 14 |
| 4.5.1 | Prototypes Developed .....                               | 14 |
| 4.5.2 | Did they play a role in shaping the final system? .....  | 14 |
| 4.6   | Implementation Plan.....                                 | 14 |
| 4.7   | Implementation Results .....                             | 15 |
| 5     | Chapter 5: Summary, Conclusion, and Recommendations..... | 15 |
| 5.1   | Summary .....  | 15 |
| 5.2   | Conclusion and Recommendations.....                      | 15 |
| 6     | Appendices.....  | 16 |

|       |   |    |
|-------|---|----|
| 6.1   | Source Code .....                         | 16 |
| 6.1.1 | Application & Certificate Generator ..... | 16 |
| 6.1.2 | DAO .....                                 | 20 |
| 6.1.3 | Models.....                               | 31 |
| 6.1.4 | Freemarker Templates .....                | 32 |
| 6.2   | Sample Interfaces.....                    | 46 |
| 6.2.1 | Prototype Interface .....                 | 46 |
| 6.2.2 | Final Interface.....                      | 47 |
| 6.3   | User Guide.....                           | 53 |
| 6.4   | Information Flow.....                     | 53 |
| 6.5   | References.....                           | 54 |
| 6.6   | Poster .....                              | 55 |

## Table of Figures

|  |    |
|--|----|
| Figure 1. Method Example.....                          | 4  |
| Figure 2. Care Mount Medical Portal.....               | 7  |
| Figure 3. V Model .....                                | 8  |
| Figure 4. Database Configuration .....                 | 12 |
| Figure 5. Parameter Implementations.....               | 12 |
| Figure 6. Freemarker Keyword Example.....              | 13 |
| Figure 7. Prototype Login .....                        | 46 |
| Figure 8. Prototype Patient Portal Home .....          | 46 |
| Figure 9. Prototype Patient Portal Personal Info ..... | 47 |
| Figure 10. Home Page.....                              | 47 |
| Figure 11. Login .....                                 | 48 |
| Figure 12. Client Portal .....                         | 48 |
| Figure 13. Client Appointments Tab .....               | 49 |
| Figure 14. Client Return to Home Tab .....             | 49 |
| Figure 15. Staff Portal .....                          | 50 |
| Figure 16. Staff Client Select .....                   | 50 |
| Figure 17. Staff Appointments Tab .....                | 51 |
| Figure 18. Staff Return to Home Tab .....              | 51 |
| Figure 19. File Display and Download.....              | 52 |
| Figure 20. Halo Logo (unfinalized) .....               | 52 |
| Figure 21. Server Handshake (Created in DIA) .....     | 53 |
| Figure 22. File Download (Created in DIA).....         | 54 |
| Figure 23. HALO Poster.....                            | 56 |

# 1 CHAPTER 1: INTRODUCTION

---

## 1.1 PROJECT CONTEXT

Today we are in the information age and individuals can access their data from virtually anywhere. However, when it comes to an individual's medical data, this is not the case. Medical documents are not always available for download, most of the time you must physically go to a medical institution and request your files and obtain a physical copy of said files. On top of this, sometimes institutions charge a fee for you to go and retrieve your data. In the age of information, I believe that it is bizarre that individuals do not have proper access to their medical files.

In other cases, a medical institution may have an information system associated with it. However, many times individuals are intimidated by the interface for there is so much going on. When it comes to medical information, the utmost care should be put into making sure that individuals can access their data properly and without confusion.

## 1.2 PURPOSE

The purpose of this project is to create a system to allow individuals the proper access to their medical data. This includes viewing and downloading files remotely. This can also help with the transference of individual's information from institution to institution. Another purpose of this project is to create a system that is easy to use and understand. Like I mentioned in part 1.1, many existing systems are complex and hard for individuals to understand, so a more minimalistic and well-managed system will be the desired outcome. Another purpose of the project is to give users a system that is compatible with many devices, for some existing systems require certain software to function. And the final purpose is to create a base project to work on for years to come. The amount of time that should go into a system like this, is more than the time one must complete a capstone. There are so many outside variables that one needs to account for to create a perfect system.

### 1.2.1 Description:

With this project, individuals will be able to access the system for multiple purposes. Medical staff will be able to easily access to create and modify patient records. Many medical establishments have problems communicating between internal departments, and when it comes to communicating between medical establishments or communicating with medical establishments, sometimes it can take a long time to find the data you need. Standards for each medical establishment can be different and therefore some of their systems clash making it hard to communicate.

Each medical establishment will have its own login, but once on the system, you can look up data that you have been given the privilege to look at. There are different access levels that a medical establishment will be able to acquire. Preferable there will be an IT Admin at every site, and they would have the highest clearance, the Doctors and Nurses would have their own levels and so on. Special access can be given to others, not in the hospital but registered in the system. Once their level is updated, they will be able to see the data you have sent to them. This will not just be for medical establishments, but it can also be used by the government. Some jobs in the government require a medical check like for the military, so now instead of going back to all of your hospitals to get all of your medical data, you would be able to call up administrators of this system to create a new login for whoever is requesting your data and then all of your medical data will be in their hands to analyze appropriately.

Medical establishments have all their own forms and these forms change from establishment to establishment. In an ideal world, a board of doctors will get together to find the best form and standardize that form in the system. In this case, the form that is created will have all the necessary data relating to the patient. No more going back to a previous hospital to get more forms filled out, or having hospitals take more time communicating to try and request more data about a specific patient.

The base system, however, will be a Ktor server hosting Freemarker Template files. This will allow users to have a nice web GUI that allows them to communicate with the server. The server will be able to send a user the files they request. For the staff, they will be able to see the files of the patients they have been given access to. For the base system, the more detailed actions like creating new users, and uploading files will be through the terminal. Future updates to this system will create GUIs for these processes along with others.

### 1.3 THE OBJECTIVE OF THE PROJECT

The objective of this project is to create a system that is easier to use than current complex systems. Another objective of this project is to create a system to allow individuals to access their medical data to use as they see fit. This includes viewing and downloading files remotely along with transferring their files to other institutions. This could apply to activities like transferring data from hospital to hospital or even transferring data from the hospital to the military. The final objective is to learn about existing systems and to understand how to create new ones.

### 1.4 SCOPE

From starting from square zero there would be a lot to go into this project:

- Creating or joining a company
- Getting funding

- Hiring and training staff
- Buying Equipment
  - Servers
  - Terminals
  - Etc.
- Creation of a Kotlin Server
  - Creation of Multiple Databases
  - Creation of Tables
  - Creation of Individuals
- Creation of a Website
- Creation of a mobile application
- Creation of standardized forms
  - Surveying multiple medical professionals to create the standardized form
- Virtualizing Files
- Testing the system
- Fixing the bugs
- Acquiring a copyright
- Marketing the product
- Selling and distributing the product
- Maintaining the product after it has been distributed

#### 1.4.1 Limitations:

Although there is a lot that goes into the creation of this project, being a college student with not many channels to access everything needed, these are the limitations:

- Creating a company
- Getting funding
- Hiring and training staff
- Buying Equipment
- Creation of standardized forms
- Acquiring a copyright
- Marketing the product
- Selling and distributing the product
- Maintaining the product after it has been distributed

This leaves just the programming aspects of the project.

## 2 CHAPTER 2: REVIEW OF RELATED LITERATURE

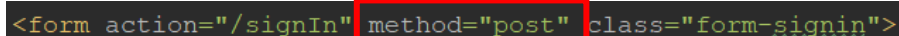
---

I would like to start by saying there are many classes in my college career that helped me in small ways to make this project come together. Classes like CITA 202: Computer User Support Conclusion and Skills helped me get into the mind of a user and think about some of the features that users may want, and classes like CITA 204: Systems Analysis and Design allowed me to think of different ways to evaluate existing systems and build a better system based off of that evaluation. Both of those classes were at my previous institution, however more recent classes like IS 469: Information Technology Project Management and COM 308: Analytical Research and Writing. IS 469 mainly helped when trying to figure out a way to plan and schedule this entire project, it has improved my thought process when working on large projects like this one, and COM 308 helped with my writing skills so a paper like this one could be created. There are many more that contributed to my knowledge, but these are the main ones that helped me with this project.

### 2.1 KOTLIN RESOURCES

Getting started with learning how to host websites with Ktor, I started with Ktors getting started page (JetBrains, 2018). Here it shows an example of a very simple server and how it communicates with Freemarker templates. This site is what got me to use Freemarker templates after seeing how it can send and receive data. From here I had to read up more on routes.

On their routing library page (JetBrains, 2018), there were many ways to implement routes. I learned the keywords and since there were not many examples, I just started testing different implementations to fully understand routes. The specifies route() is the extension on the URL beyond the home page. You can arrive at this page if you type in the specified path on the end of the base URL, or if you use an HTML file that then redirects you to this path. Under this path there is the get{} and post{} sections. The get{} is what responds when you first arrive at the path, and the post{} is what collects the data when you are outgoing. When using a form in HTML or FTL, you must specify the method as post for that path to receive the variables, shown in figure 1 below.



```
<form action="/signIn" method="post" class="form-signin">
```

*Figure 1. Method Example*

After learning about routing, I proceeded to learn about how to respond to files and other items. I was able to learn all about the different possible responses. With header responses, I could send structured data to mobile applications. This will be used later when I get to make a mobile application. However, I was mainly focused on the call.respond() and call.respondFile() functions. With call.respond() I can respond with a Freemarker template, wrapped in the



FreeMarkerContent() function to be able to display the Freemarker Templates as HTML files in a web browser. With call.respondFile() I will be able to select user's files and send them to be displayed in the browser. The user will be able to download this file if they wish, which is just what I needed (JetBrains, 2018).

Once I had learned about responses, requests were next on my list. Apart from the same call library, the requests library was easy to understand after learning about responses. The requests looked very similar to the responses. However, with responses, I was able to learn about parameters and queryParameters. These could be used to receive parameters from incoming files in get, and receive the parameters being sent to the post function. (JetBrains, 2018)

Another fantastic resource was the documentation for Exposed located on GitHub (JetBrains, 2020). Here I was able to learn about all the databases supported, the structuring for their SQL DSL, the structuring for DAO, and how to set up the base database. The examples provided made it a lot easier to understand just how it can be implemented. I learned that there is the possibility of adding a password to the database itself, this may be implemented in future builds. JetBrains showed examples of inserts, selects deletes, and updates. I picked it up fast thanks to my previous experience with SQL in CITA 215: Database Apps and Concepts at my previous institution.

The final resources was the information I remembered from CS 490: Kotlin Programming last semester along with my notes from the class. I came in with the knowledge of how to configure a Ktor server and define how it will be deployed and what module the main function is coming from. I also already knew about the install() feature. This feature makes it super easy to install things like Freemarker, Authentication, Routing, and many other features. I also came in with a good understanding of Kotlin as a language. Another class that helped me figure out a way to give the database some secure communication is NCS350: Wireless Systems and Security. This class helped me understand the importance of certificates and allowed me to understand how they work.

## 2.2 FREEMARKER AND HTML RESOURCES

Since I do not know much of HTML design, I turned to Bootstrap to learn about how to structure eye-pleasing formats using their libraries. I learned how to use JavaScript and got a better understanding of HTML

All of my research material for Freemarker was found on the Freemarker website (Apache, 2020). Here I was able to find the keywords that will be used to display the data from my models in Kotlin along with how to send load the Freemarker templates. Most of the documentation is in Java, (Apache, 2020) but after implementing the library the use of the

template loader was very similar to its Java implementation. This will be used paired with the Ktor responses to respond with Freemarker templates in the proper format.

I was able to find the keywords to be used in the Freemarker Templates on their data model page (Apache, 2020). Here they show a couple of examples of simple Freemarker templates. Inside them, they used keywords `<#list>` as a loop like a keyword that will perform the item inside of its body for each item in the list. This will be used for listing files and users. Other keywords I picked up from this was the use of `${}`. For this keyword, you would put the “modelName.variable” inside of the brackets. This will allow you to display and modify the data coming in with the specified model name from the Ktor route.

Finally, I was able to learn about `<#import>`. With this keyword, I could import libraries that I had created with specified `<#macros>` to allow me to display my data without repeating certain lines of code over and over (Apache, 2020).

Since I do not know much of HTML design, I turned to Bootstrap to learn about how to structure eye-pleasing formats using their libraries. I learned how to use JavaScript and got a better understanding of HTML

## 3 CHAPTER 3: METHODOLOGY

---

### 3.1 PROJECT PLAN

The plan for this project starts with researching other existing systems. I researched the ones that I had access to meaning, my current doctors office’s system, and a local hospital that has many of my medical files. From there I will ask a professional about these existing systems and ask if a system like the one I am proposing exists, and if so, what advice do they have for me. After I have gathered my data from testing existing systems and asking a professional, then I will start to create my system keeping in mind what I have seen and heard. Once the system is created, I will test that system to make sure it works as planned and fix any bugs. And after this project has been completed, I will add more features in the upcoming years to eventually create the system I can see this becoming.

### 3.2 DATA GATHERING PROCEDURES

The first case of data collection was at Danbury Hospital. Danbury Hospital is a hospital located near me that has a lot of medical data comprising of many test results, many x-rays, surgery descriptions, etc. This made it the perfect way for me to test current data retrieval methods. So, I called them up to ask for my files and when I got in touch with the correct department, they were able to find my files, however, I had to physically go to the hospital to pick the files up and there was a fee involved. Upon arrival, my files were given to me in the form of a disk, and in the physical form of papers. However, upon me inserting the disk into my

disk drive and trying to open the data, I was unable to read the disk. The disk was meant for a certain software and could not be opened on my computer. Upon taking this disk to my doctor's office, they had the same issue.

The second case of data collection was observing the system that is used by my doctor's office, Care Mount Medical. They did, however, have an existing system in which I could see my medical data. However, this system has a lot of different data and it was very intimidating to navigate. You can look at the interface below in figure 2. Although my data was somewhat virtualized, I still could not download the data, I still had to go to the office and pick up physical papers.

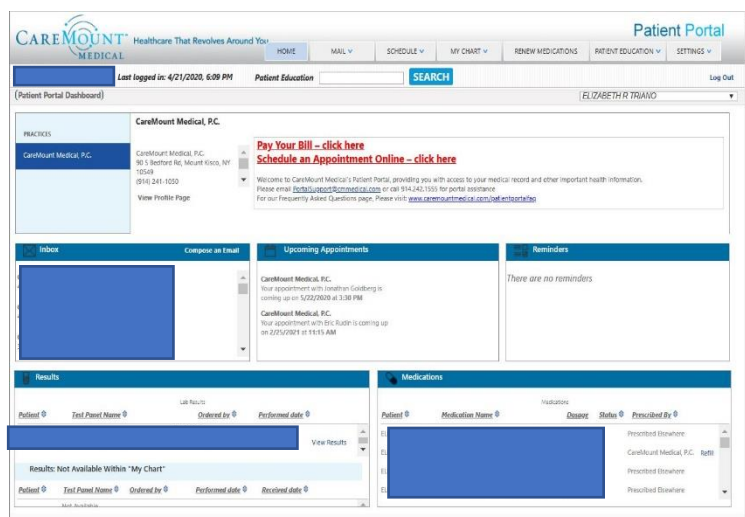


Figure 2. Care Mount Medical Portal

The third case was with the military. This case worked out by coincidence, for I am trying to become an officer in the Navy. When it came to the medical clearance, it took 4 weeks for my medical data to be sent from the hospital to my recruiter. On top of that, it took a week to go from the recruiter to the military doctors for evaluation.

After looking into other systems, I decided to contact a professional who works on systems like these. The individual who I was able to contact is Chris Apgar, CEO of Apgar and Associates, LLC. Chris travels around the country and evaluates systems like these and make sure that they are compliant to the HIPAA privacy and security policies. Upon asking him if systems like the one I am proposing already exist he says yes, and he believes that more should exist, however, they require a lot of security and it is hard to properly secure them.

When gathering data on what type of system should create, two methods came to mind: using PHP and MySQL or the use of Ktor and Freemarker Templates. I have had experience with PHP and MySQL in the past along with a small amount of exposure to Ktor. PHP and MySQL would be good for only a variety of actions, while Ktor could be a lot more versatile and better for future expansion.

From this research, I can take away that not many systems like the one I am proposing exist, and those who do have it, have complex interfaces that are hard to understand. Another takeaway is that many existing medical institutions may have their data virtualized, but the way it is transferred is

somewhat archaic, for example, Danbury hospital gave me a disk, yet most computers no longer have a disk drive, I was lucky enough to have a USB CD drive.

### 3.3 SOURCE OF DATA

The data from the cases in [section 3.2](#) came from the physical establishments like Danbury Hospital and physically meeting with Chris Apgar, while others came from online sources like the Care Mount Medical Portal located at <https://www.caremountmedical.com/>.

### 3.4 SOFTWARE MODEL

For this project, the use of a V model seemed most appropriate. The V model (displayed in figure 3 below) will be best for my plan for I will start by finalizing my project definition, then once it is finalized, it will be implemented, and after it is implemented I will test it. If I go back to verify that it abides by the initial project definition and it does not, I can just go through the V model again until it is perfected.

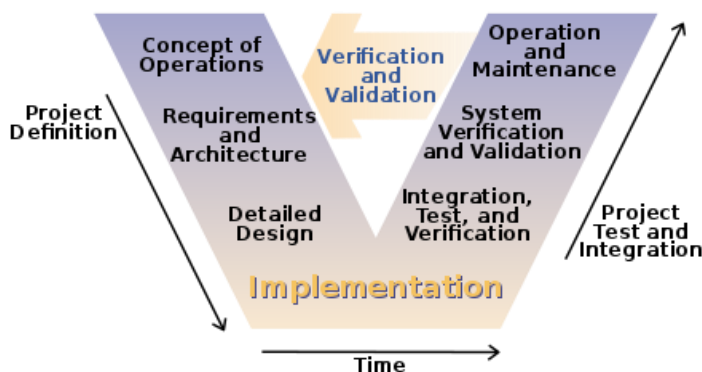


Figure 3. V Model

## 4 CHAPTER 4: RESULTS AND DISCUSSION

### 4.1 REQUIREMENTS ANALYSIS

The requirements of building this system after establishing the achievable scope are actually not that extensive. The requirements being

able to program the backend Ktor server is having a knowledge of Kotlin, or even Java since Kotlin and Java are so closely related. You will also want to know a little bit of HTML to create the GUI and to be able to install the IntelliJ J IDE. These requirements are listed on the JetBrains website as 1GB RAM Minimum, 2GB recommended, 300 MB Hard Disk Space + At Least 1 GB for caches, Microsoft Windows 8, or later, macOS 10.13 or later or any Linux that Supports Gnome, KDE or Unity DE.

You will also want to increase the speed of the created application for tasks like compile-time, or even just runtime response speed. The configuration I will be using will be an I5-9600K, 16GB of RAM, and 500GB of SSD storage. On a setup like this, running the server will be effortless for small test demonstrations, and compile-time will be minimal. The more storage you have means the more patient files you will be able to store, meaning larger institutions will need quite a lot of memory, but since individuals have different amounts of files and files are different sizes, this amount is unpredictable.

You will also want extra room to store data from plugins like Ktor, and to download libraries for Ktor, Exposed, H2database and Freemarker. These downloads will allow the Ktor server to properly function, to allow the addition and storage of an SQL database, and will allow the ability to display all the information coming from the database in a web GUI.

For future application development, you will also want to be able to run Android Studios Which has the following hardware recommendations: Windows 7 and up, 3GB RAM minimum, 8GB RAM Recommended with an additional GB if you want to run the emulator and 2GB Storage Minimum, 4GB recommended. Future requirements could also include the discussion with stakeholders about fine tunings of the system, like if the future stakeholders wanted certain features included in the system that they would like to add.

## 4.2 REQUIREMENTS DOCUMENTATION

This project should create an interface that is easy to understand and allows users to navigate with ease. The users should be able to carry out the functions of the application with ease. The base product should allow users to log on and download their data to archive for their use, or to distribute it to others who may require their medical records. In future builds, the user should be able to carry out functions like make appointments and if you are a doctor or nurse using the system, you should be able to upload files, and communicate between departments.

Stakeholders who decide to use this product as their system might have requests for other functions to be added to the system. If so, we will need to meet with the said stakeholders and figure out a project plan and figure out the scope of work that will be involved with adding their new function. If there is an already existing system that they wish to integrate our system into, then we will again have to sit with the stakeholders and see how compatible the systems are and figure out the scope of work involved with the integration.

Since this is the base product and there will be expansions in the future, and this system will be used in establishments of many different sizes, we have to make sure this system is created with good scalability. For the case of expanding the functions of the system, Ktor makes it easy to expand the system. With its functions like routes and install, new features can be added very quickly. And for the sense of creating systems of different sizes, all you need is good hardware to back it up, and the system will be able to support it.

The security of individuals' personal data is not something to be taken lightly. The current state of security on this project will be using self-signed certificates and SSL. This will be a good base to ensure the data does not get hijacked in transmission, however, it will need many improvements in the future until it can become the optimal system, I imagine it becoming.

### 4.3 DESIGN OF SOFTWARE, PRODUCT, AND PROCESS

The design of this system took a couple of failure prototypes before arriving at the system we designed for this project. The initial design of the project was a MySQL server with PHP and HTML documents to access the data from the database. It worked well for its purpose of displaying and keeping track of individuals data, however, the versatility of that system was not as great as Ktor. Not only that, but I could make much more complex transactions with a Ktor system. These were the main variables that switched my initial design from PHP and MySQL to Ktor. From there I had not had a lot of experience with Ktor, so I started a design based on what I knew and used Http and header libraries. However, after more research into these libraries, I could not use them to display and manipulate database data as I needed. This took me to the final path of using Ktor paired with Freemarker Templates to display and manipulate the database data as I needed.

With the configuration finalized, the design of the software continued. Starting with Ktor, we needed to design a well-structured backend. We can host this server by implementing Netty. Netty will allow the server to be hosted by listening on a specified port, this will be the entrance point configured for users. Once this host is created, we must configure the inner workings and the navigations of this server. This is where routing comes into play. Routes are a Ktor library that allows us to send specified data in certain directions. This is how websites work when you add a "/" at the end of the URL, this will take you to another page than the home page. Behind each of those paths are specified transactions that may respond in different ways depending on what is on that path. Sometimes it will take input variables from the page, or display different items, this is all established by the get and post methods located in the route. When you arrive at a path, the get method is first processed. Then when you leave that path, the post method comes into play and may take certain variables from the page or help direct you to your next path.

The routes can communicate with the request and respond libraries, both are self-explanatory. With the request method, you request specified items from the incoming data. This will allow us to get critical data from the user like their username and password when signing in to then authenticate the user and respond appropriately. Respond sends data back to the user, sometimes directing them to another path, or just sending them files.

The database itself needs models and objects to be fully functioning. The models will be the Kotlin representation of the Exposed SQL table objects. The objects will allow databases to be created with the specified variables, and the models will allow you to extract that data while keeping the correlation of the data intact. With these models and objects now created, all the database transactions now just have to be created. For this database to function as we need, we will create, select, and edit the data in the database.

Finally, we need to make a certificate generation system. This is the least complex piece of the system. Ktor already has a certificate generation library, making it easy to generate the certificate and then send the certificate to the users.

## 4.4 DEVELOPMENT AND TESTING

### 4.4.1 Database Creation

To create the database, we first need to create the models and objects that will allow the database to function. Creating these models are simple. The models are each a specified data class with just variables with specified types and names. These models will allow us to create the specified model to have structured data before sending it to the database, or when taking it out of the database. For this project, we will need to create a Doctor, File, and Client model. The Doctor model will have a variable to keep track of the username, password, name, and what medical institution they belong to. The File model will need to keep track of the username, name, file name, and the name of the medical institution it belongs to. Finally, for the Client, we will need to keep track of the username, password, name, and medical institution they belong to. These models can be referenced in [section 6.1.3](#).

After the models are created, we can create the Table objects. It is best to create these objects with relating names to their corresponding model. So, for these objects, we will need to keep track of Doctors, Files, and Clients, each table named accordingly. This is also where we can specify unique, primary key and not null just like in SQL. Starting with the Doctors Table, we will need to translate the variables from the model into an SQL table, so they will have the same variables, but they will now be assigned to varchar, Int, char or whatever SQL type they need to be converted to. Once you specify its SQL type, you can then add extensions like “.primaryKey()” for items you do not want duplicates of. We will be creating primary keys for the usernames of the Clients and the Doctors so that duplicate users cannot be created.

After forming the tables and models, we can now create the functions for the database itself. We first want to create declarations for each of our methods. We want to be able to create, read, update, and delete entries for each of these tables that we have just created. So, for each model, we created functions for Selecting certain variables from specified models, updating certain models’ variables, and deleting specified models (see [section 6.1.2.1](#) for the source code). After we have created ways to CRUD in our database, we have to create the initialization function. In this initialization function, we create the database and start a transaction in the database to populate the database. To start, we start up a database transaction to start editing the database. To create a table, we use SchemaUtils create function and send it the object model of the table we want to create. After that, we need to populate the tables. We will insert variables during the testing phase in [section 4.4.4](#).



#### 4.4.2 Ktor Server Development:

Starting the development of the Ktor server, I first created the application Kotlin file which will house the main method, this is where the server, and where it will be told to act in certain ways. Before we start the server, we need to configure how the database will be stored. This is done by connecting the database to a specified local URL and specifying what driver should be used. Since we want to keep the database open the entire time the machine is running, we will use the configuration “DB\_CLOSE\_DELAY=-1” to keep the database running for the entire time that the server is running. For this example, we will be using the H2database and drivers (shown in figure 4 below). To get the server started, we start the Netty server on port 8080. This will allow the user to connect by going to the server's IP using the specified port. Once we initialize Netty, we can start creating the routes.

```
val dao = DAOFacadeDatabase(Database.connect("jdbc:h2:mem:test;DB_CLOSE_DELAY=-1", driver = "org.h2.Driver"))
```

Figure 4. Database Configuration

We need to create a route for each type of transaction that will occur. So, we will need routes for a home page, sign in, each of the portals, and downloading files. This is easy to implement for you just use the routing libraries. You use route to specify the certain routes, and use get and post methods in that to decide how to use the incoming and outgoing data from each route. You can see examples of this in [section 6.1.1.1](#).

When responding with Freemarker Files, we use the Freemarker Ktor libraries to use functions like FreeMarkerContent to allow us to display the Freemarker files like HTML files on a user's screen. We can send data to the file by using the “mapOf()” function to map lists of certain models to names to be accessed in the Freemarker files. We also use the receive Parameters, and Parameters libraries to receive the values coming in from the Freemarker files and process them as needed. To do this we use “receiveParameters()” in the post method and “queryParameters[]” in the get method. In the square brackets, you will place the name of the variable in the Freemarker file that you want (shown in figure 5 below)

```
val postParameters = call.receiveParameters()
val action = postParameters["action"]
val action = call.request.queryParameters["action"]!!
```

Figure 5. Parameter Implementations

#### 4.4.3 Freemarker File Development:

Freemarker files are just HTML files with a different extension and some keywords to process the data. When displaying the data from the Freemarker files, we used Freemarker keywords like “<#list>”, “<#import>”, and “\${}”. When listing user files, we use the list keyword to catch specific models being passed into the Freemarker files. We used import to import macros from a created library and the “\${}” is used to display variables from the database. For



example, in figure 6 we see the list keyword being used to catch the human model and rename it hum to reference. Inside of the table body, the “\${}” is being used to display the first and last names of each human in the list. For the created button, when selected, it will send the username of the selected human to the /fileViewer route as the name parameter. You can see how it is received in the /fileViewer route in [section 6.1.1.1](#).

```
<#list human as human>
<tr>
  <td>${human.fname} ${human.lname}</td>
  <td>
    <a href="/fileViewer?name=${human.user}" class="btn btn-secondary float-right mr-2" role="button">See Files</a>
  </td>
</tr>
</#list>
```

Figure 6. Freemarker Keyword Example.

#### 4.4.4 Testing:

When testing this system, we populated the tables with some testable data for their corresponding model types. We do this by using the “batchInsert” method. After the database was populated, we then started the server using the configuration made in our application Kotlin file.

We then proceeded to chrome to try and display the system by going to “localhost:8080”. This was a success and it displayed the home page (shown in [section 6.2.2](#) figure 10). From the home page, we then proceeded to the login page and tested a sample login. We tested the fact that both fields were mandatory so no fields could be left blank, and then tested a valid user login. (shown in figure 11). We can see that it proceeded to the proper portal. Notice how we have a side navigation bar that allows you to navigate the pages. Here we will test that we can open and download a file in the created web environment. And as shown in [section 6.2.2](#) figures 12 – 14 the test of the client portal was a success.

We now go to attempt to log in as a doctor. We use a sample login, and it takes you to the doctor portal. Notice how the doctor portal displays a list of users they have access to, and we can view their files (shown in [section 6.2.2](#) figure 15).

After we had created the base product that we wanted, we decided to start to implement other features. In the side navigation, there is a tab on both the staff and client portals that allows them to create an appointment. This appointment is then put into a schedule table and linked to the users involved in the appointment (shown in [section 6.2.2](#) figures 13 and 17). After testing this function, we notice that the appointment is still outputted in a date-time format that it is stored in the database, and when creating an appointment, it doesn’t conform with the other user involved in the appointment, it just makes an appointment for both of the users.

## 4.5 DESCRIPTION OF THE PROTOTYPE

### 4.5.1 Prototypes Developed

The prototypes developed were those mentioned at the beginning of [section 4.3](#). It started with a system that communicated with a MySQL server by using PHP files paired with HTML files. The prototype database consisted of tables for different departments, users, and doctors. There were multiple databases created to simulate multiple medical institutions. The users and the doctors had similar data; the only difference is that the files were saved as blobs in the user table. Examples of this interface can be seen in [section 6.2.1](#). Being that this prototype was created in 2017 during my freshman year of college trying to figure out an idea for my capstone, I got drowned with other course work and revisited the idea my Junior year. When this came around, I had started using Kotlin and learned about Exposed, which is when I moved to the second prototype design.

The second prototype created was to use Ktor and Exposed in conjunction with Windows and Android applications to display the interface. I had only gotten the backend semi-functional before the idea of using a web interface hit me, for not everyone has android or Windows, so that would limit the number of people who can use the system. This code, however, is saved for when I implement a mobile application for the system.

The third and final prototype was similar to the second one. It used Ktor and Exposed in conjunction with a couple of Ktor HTTP libraries. After forming the backend and the routes, I tried to capture data from HTTP files using these given libraries, but I could not read the format in the way that I needed. Enter version 2.0 of the third prototype. This version used Freemarker Templates to extract and display data from the database in these HTML-like files. I took out all the HTTP implementations and added the Freemarker library and that is what created the final working system.

### 4.5.2 Did they play a role in shaping the final system?

Each prototype had a huge role in influencing the final system. The general concept of how the data should be stored stayed the same through each prototype, an SQL database with a way to access data from it. The only changes were how to communicate with the database. With each decision, the versatility and the ability to process complex transactions improved. I am glad that I ended with Ktor, for it is highly versatile for what I am trying to accomplish, and adding new features should be a breeze!

## 4.6 IMPLEMENTATION PLAN

This system will be tested on my local network, for a static IP costs \$500. In this implementation, we will be running the system on my desktop and make sure multiple users can connect to the database and make transactions at the same time with no errors and little to no speed dips. For this, we have a total of 6 computers that we will be connecting at the same

time to the server. We will each download a file from the server and navigate the websites to see how it works.

#### 4.7 IMPLEMENTATION RESULTS

After having multiple users connect to the database, we noticed that there was a limitation to the download speeds, but that was because of my hardware. It seems my network card has its speed limited to 300Mbps. Other than that, two instances of the user could not be logged in at the same time, and individuals could navigate around the system with ease. The system itself had no problem processing the transactions in the backend. Overall, the implementation was a success.

## 5 CHAPTER 5: SUMMARY, CONCLUSION, AND RECOMMENDATIONS

---

### 5.1 SUMMARY

After researching current medical systems, I believe them to be inadequate. Between data that is incompatible with certain systems and systems that are intimidating navigate, medical files are pretty hard to access. This should not be the case, for we are in the Information age, we can download all sorts of files from the internet, so we should have access to all of our personal data.

The development of this system took a little longer than I had expected. A lot more research went into the project than I thought was needed. But in the end, all that research paid off, for when I implement new features into the system, it will be a much easier process because of what I have learned. The use of Ktor and its extensive number of libraries made it very easy to make many database and server transactions after I fully understood the language. Not only that but since Kotlin is a newer language and so similar to Java, there are all sorts of new libraries being created. This means that in the future since there will be many more libraries, the possibilities of transactions possible for this Ktor server could be endless.

### 5.2 CONCLUSION AND RECOMMENDATIONS

In conclusion, a system like HALO would be a good initial or addition to a medical system. The only thing is that it will need increased security. Things like brute force attacks could try different passwords until it gets in, along with the fact that if attackers know the location of the server and knows where the files are located, they can find and view the client's personal medical files. However, libraries do exist for Ktor that will allow for a more secure system. So, with more time, a well-secured system could be created.

As for the future of this project, there are many different features that I will be adding in the future. Aside from increasing the security, I would like to integrate Google. If a user allows

the system access to their Google profile, that would mean the system would have access to the individual's calendar. This could allow the system to create a calendar event when an appointment is created. With these calendar events, it will be able to schedule different notifications through Google to remind the user of their upcoming appointment.

Another plan requires another part of the scope to come true. Back in [section 1.4](#) we mentioned meeting with a board of medical professionals and creating a set of standardized forms. Since there are so many medical institutions out there, many of them have a different structure to their forms and this can sometimes take time for “translation” when going between medical institutions. If a standardized form was created, then medical professionals will understand the structure of all files, no matter where they come from. This being said, a form parsing utility could also be created. This utility would know exactly where to look for things like age, height, medical conditions, current medications, and many other details about a client and output it in a summary for the doctor to read. For example, if you are traveling in another state, and you become injured and are taken to a hospital, the doctor will be able to look you up and see a summary of medical conditions you have and what medications you may be taking for a faster diagnosis.

Some simple additions to the system are items like an alphabetizer. This will allow for a neater, more organized system. Another simple addition is adding a chat system to communicate between departments. This will allow doctors to send requests to other departments instead of possibly using other systems like email or calling. For example, If I work in pediatrics and I want to have a client get an x-ray, I can send a request down to the radiology suite asking if I can get said client in for an x-ray. They can then either accept or deny this request and that appointment will then be pushed to your calendar.

## 6 APPENDICES

---

### 6.1 SOURCE CODE

#### 6.1.1 Application & Certificate Generator

##### 6.1.1.1 *Application.kt*

```
package main.kotlin

import freemarker.template.Configuration
import freemarker.cache.* // template loaders live in this package
import io.ktor.application.install
import io.ktor.application.call
import io.ktor.features.CallLogging
import io.ktor.features.ContentNegotiation
import io.ktor.freemarker.FreeMarker
import io.ktor.freemarker.FreeMarkerContent
import io.ktor.routing.routing
import io.ktor.routing.route
import io.ktor.routing.get
import io.ktor.routing.post
import io.ktor.request.receive
```

```

import io.ktor.request.receiveParameters
import io.ktor.http.content.resources
import io.ktor.http.content.static
import io.ktor.http.Parameters
import io.ktor.http.content.LocalFileContent
import io.ktor.response.respond
import io.ktor.response.respondFile
import io.ktor.server.engine.embeddedServer
import io.ktor.server.netty.Netty
import io.ktor.request.receiveMultipart
import main.kotlin.dao.DAOFacadeDatabase
import main.kotlin.model.Human
import main.kotlin.model.MedCenter
import org.jetbrains.exposed.sql.Database
import java.io.File
import io.ktor.routing.header

/* Author: Dominic Triano
 * Date: 2/15/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Processing of the data coming to and from the database, these are the main functions
 *
 */

val dao = DAOFacadeDatabase(Database.connect("jdbc:h2:mem:test;DB_CLOSE_DELAY=-1", driver = "org.h2.Driver"))
fun main() {
    embeddedServer(Netty, port = 8080) {
        //Configuration cfg = new Configuration(Configuration.VERSION_2_3_29)
        dao.init()
        install(FreeMarker){
            //allow the connection with the ftl files
            templateLoader = ClassTemplateLoader(
                this::class.java.classLoader, "templates")
            print("Test")
        }

        routing {
            route("/") {
                get {
                    //responds with said .ftl file
                    call.respond(FreeMarkerContent("Home.ftl", null))
                    //call.respondFile(File("resources", "name of file"))
                }
                post {
                }
            }

            route("/signIn") {
                get {
                    //responds with said .ftl file
                    call.respond(FreeMarkerContent("SignIn.ftl", null))
                }
                post {
                    //receives the parameters sent from the .ftl file associated with the path
                    val postParameters = call.receiveParameters()
                    val action = postParameters["action"]
                    //testing where data is going missing
                    print("testingg1")
                    print(postParameters["action"])
                    print(postParameters["inputUser"])
                    print(postParameters["inputPassword"])
                    when(action) {
                        "signIn" -> {
                            //print("testingg2")
                            val id = postParameters["inputUser"] ?: "empty"
                            val pass = postParameters["inputPassword"] ?: "empty"
                            if(dao.authentication(id, pass)) {
                                //print("testingg3")
                                //whatever access level they have says what they are
                                //1 - Client, 2 - Doctor, 3 - Developer, Failed to get it? Try again
                                print(dao.getHuman(id, pass)[0].user)
                                when(dao.getAccess(id)) {
                                    1 -> call.respond(FreeMarkerContent("index.ftl", mapOf("files" to

```

```

dao.getUserFiles(id) , "doctors" to dao.getDoctors(dao.getGroup(id)), "user" to dao.getHuman(id, pass),
"schedules" to dao.getClientSchedule(id)))
    3 -> call.respond(FreeMarkerContent("dev.ftl", mapOf("humans" to
dao.getAllHumans() , "staff" to dao.getHuman(id, pass)))
    else -> call.respond(FreeMarkerContent("SignIn.ftl", null))
    }
    }else if (dao.docAuthentication(id, pass)){
        call.respond(FreeMarkerContent("staff.ftl", mapOf("human" to
dao.getClients(dao.getDocGroup(id)), "doctor" to dao.getDoctor(id, pass), "files" to
dao.getMedFiles(dao.getDocGroup(id)), "schedules" to dao.getDocSchedule(id)))
    }else{
        call.respond(FreeMarkerContent("SignIn.ftl", null))
    }
    }
    //There should be no other possible action, but just in case, send them back
    else -> call.respond(FreeMarkerContent("SignIn.ftl", null))
    }
    }

route("/client"){
    get{
        // val user = call.request.queryParameters["user"] ?: "empty"
        // call.respond(FreeMarkerContent("index.ftl", mapOf("files" to dao.getUserFiles(user))))
    }
    post{
    }
}

route("/staff"){
    get{
        val user = call.request.queryParameters["user"] ?: "empty"
        call.respond(FreeMarkerContent("staff.ftl" , mapOf("humans" to dao.getClients(user))))
    }
    post{
    }
}

route("/dev"){
    get{
        call.respond(FreeMarkerContent("dev.ftl" , mapOf("humans" to dao.getAllHumans())))
    }
    post{
    }
}

route("/download"){
    get {
        //requesting specified variables from FTL
        val action = call.request.queryParameters["action"]!!
        val user = call.request.queryParameters["user"]!!
        val fileName = call.request.queryParameters["name"]!!
        // println("testing" + action + " " + user + " " + fileName)
        when(action){
            "download" -> call.respondFile(File("resources/userFiles/" + user, fileName))
            else -> call.respond(FreeMarkerContent("index.ftl", mapOf("files" to
dao.getUserFiles(user))))
        }
    }
    post{
    }
}

route("/fileViewer"){
    get{
        val id = call.request.queryParameters["name"] ?: "empty"
        call.respond(FreeMarkerContent("humans.ftl", mapOf("files" to dao.getUserFiles(id))))
    }
    post{
    }
}

```

```

    }

    route("/newHuman"){
        get{
            val user = call.request.queryParameters["username"]!!
            val type = call.request.queryParameters["type"]!!
            val pass = call.request.queryParameters["password"]!!
            val fname = call.request.queryParameters["firstName"]!!
            val lname = call.request.queryParameters["lastName"]!!
            val grpCode = call.request.queryParameters["grpCode"]!!
            val access = call.request.queryParameters["access"] ?: "0"
            when(type){
                "doc" -> dao.createDoctor(user, pass, (fname + " " +lname), grpCode)
                "human" -> dao.createHuman(user, pass, fname, lname, grpCode, access.toInt())
            }
        }
        post{
        }
    }
}

//
// route("/uploads"){
//     For later Implementation
//     get{
//         call.respond(FreeMarkerContent("upload.ftl" , null))
//     }
//     post{
//         val postParameters = call.receiveParameters()
//         val multipart = call.receiveMultipart()
//         multipart.forEachPart { part ->
//             // if part is a file (could be form item)
//             if(part is PartData.FileItem) {
//                 // retrieve file name of upload
//                 val name = part.originalFileName!!
//                 val file = File("/uploads/$name")
//
//                 // use InputStream from part to save file
//                 part.streamProvider().use { its ->
//                     // copy the stream to the file with buffering
//                     file.outputStream().buffered().use {
//                         // note that this is blocking
//                         its.copyTo(it)
//                     }
//                 }
//             }
//         }
//         // make sure to dispose of the part after use to prevent leaks
//         part.dispose()
//     }
// }
//
// }.start(wait=true)
//
}

```

### 6.1.1.2 CertificateGenerator.kt

```

package main.kotlin

import io.ktor.network.tls.certificates.generateCertificate
import java.io.File

/* Author: Dominic Triano
 * Date: 2/23/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 *
 * Generates a Certificate that Ktor requires on startup
 *
 */

```

```
object CertificateGenerator {
    @JvmStatic
    fun main(args: Array<String>) {
        val jksFile = File("build/temporary.jks").apply {
            parentFile.mkdirs() //creates the file path to store the keys (
            application.conf)
        }

        if (!jksFile.exists()) {
            generateCertificate(jksFile) // Generates the certificate
        }
    }
}
```

## 6.1.2 DAO

### 6.1.2.1 DAOFacadeDatabase.kt

```
package main.kotlin.dao

import io.ktor.html.each
import main.kotlin.model.*
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction
import org.joda.time.DateTime
import java.io.Closeable
import java.util.Date

/* Author: Dominic Triano
 * Date: 4/3/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * This file includes all of the database operations needed for this to function
 *
 */

interface DAOFacade: Closeable{
    fun init() //Initiation for the databases
    fun createHuman(userId : String, pwrld : String, fname: String, lname: String, grpCode : String, lvl : Int) //creates a human object with the passed in parameters
    fun changePwrld(userId : String, newPwrld: String)//changes of the password of a selected user
    fun deleteHuman(userId: String, grpCode: String)//deletes a human from the database
    fun createMedCenter(hidId : String, grpCode : String)//creates the entry for a medCenter
    fun deleteMedCenter(grpCode: String)//deletes a medCenter from the database
    fun getHuman(userId : String, pass : String): List<Human>//selects a human from the database
    fun getName(userId: String) : String
    fun getGroup(user: String) : String
    fun authentication(userId : String, pass : String): Boolean//authenticates the users user and password
    fun getAllHumans(): List<Human>//gets all humans
    fun getMedCenter(grpCode : String): MedCenter?//selects a medCenter from the database
    fun getMedHumans(grpCode: String): List<Human>//selects all humans belonging to a medCenter
    fun getAllMedCenters(): List<MedCenter>//selects all medCenters
    fun getAccess(user : String) : Int //returns a users access level
    fun getUserFiles(user : String) : List<File>//returns the list of files relating to a user
    fun getMedFiles(grpCode: String) : List<File>
    fun fileSearch(user: String, fileName : String) : List<File>
    fun getClients(groupId: String) : List<Human> //returns the clients of a doctor
    fun getSchedule(): List<Schedule> //returns the current schedule
    fun getClientSchedule(user: String): List<Schedule> //returns the clients current schedule
    fun getDocSchedule(doc: String): List<Schedule> //returns the doctors current schedule
    fun getDoctor(user: String, pass: String): List<Doctor> //selects a doctor
    fun docAuthentication(userId: String, pass: String) : Boolean //authenticates the doctor
    fun getDocGroup(user: String) : String //Checks their group
```



```

fun getDoctors(grpCode: String) : List<Doctor>
fun getDocName(user: String) : String
fun createDoctor(user: String, pass: String, name: String, groupId: String)
}

class DAOFacadeDatabase(val db: Database): DAOFacade{

    override fun init() = transaction(db){
        SchemaUtils.create(Humans)
        SchemaUtils.create(MedCenters)
        SchemaUtils.create(Files)
        SchemaUtils.create(Schedules)
        SchemaUtils.create(Doctors)

        val humans = listOf(Human("smith1", "1234", "John", "Smith", "GHP" ,1 ),
            Human("doe1", "1234", "Jane" , "Doe", "JAMC" , 3),
            Human("roll" , "1234" , "Rock" , "N' Roll" , "JAMC" , 1),
            Human("doe2", "1234", "John" , "Doe", "GHP" , 3))

        Humans.batchInsert(humans){ human ->
            this[Humans.user] = human.user
            this[Humans.pass] = human.pass
            this[Humans.f_name] = human.fname
            this[Humans.l_name] = human.lname
            this[Humans.groupId] = human.groupId
            this[Humans.access] = human.access
        }

        val medCenters = listOf(MedCenter("Good Hospital", "GHP"),
            MedCenter("Just an Alright Medical Center", "JAMC"))

        MedCenters.batchInsert(medCenters) { medCenter ->
            this[MedCenters.hid] = medCenter.hid
            this[MedCenters.groupid] = medCenter.groupId
        }

        val files = listOf(File("smith1" , "John Smith" , "HeartScan.jpg" , "GHP"),
            File("smith1" , "John Smith" , "ShoulderSurgeryPictures.pdf" , "GHP"),
            File("roll" , "Rock N' Roll" , "ankleSurgeryDescription.pdf" , "JAMC"),
            File("roll" , "Rock N' Roll" , "clavicleSurgeryBill.pdf" , "JAMC"))

        Files.batchInsert(files){ file ->
            this[Files.user] = file.user
            this[Files.fullName] = file.fullName
            this[Files.file] = file.fileName
            this[Files.hid] = file.hid
        }

        val docs = listOf(Doctor("Calla" , "1234" , "T.C. Callahan" , "GHP"),
            Doctor("Alexa" , "1234" , "Jordan Alexander" , "GHP"),
            Doctor("Stran" , "1234" , "Stephen Strange" , "JAMC"))

        Doctors.batchInsert(docs){ doctor ->
            this[Doctors.user] = doctor.user
            this[Doctors.pass] = doctor.pass
            this[Doctors.groupId] = doctor.groupId
            this[Doctors.name] = doctor.name
        }

        val schedules = listOf(Schedule("smith1" , "Calla" , DateTime(2020, 5, 3, 14,
0)))

```

```

        Schedules.batchInsert(schedules){ schedule ->
            this[Schedules.client] = schedule.client
            this[Schedules.doctor] = schedule.doctor
            this[Schedules.day] = schedule.day
        }

        Unit
    }

    override fun createHuman(userId: String, pwr: String, fname: String, lname:
String, grpCode: String, lvl: Int) = transaction(db){
        Humans.insert {
            it[Humans.user] = userId;
            it[Humans.pass] = pwr;
            it[Humans.f_name] = fname;
            it[Humans.l_name] = lname;
            it[Humans.groupId] = grpCode;
            it[Humans.access] = lvl;
        }

        Unit
    }

    override fun createDoctor(user: String, pass: String, name: String, groupId:
String)= transaction(db){
        Doctors.insert{
            it[Doctors.user] = user;
            it[Doctors.groupId] = groupId;
            it[Doctors.name] = name;
            it[Doctors.pass] = pass;
        }

        Unit
    }

    override fun changePwr(userId: String, newPwr: String) = transaction(db) {
        Humans.update({Humans.user eq userId}){
            it[Humans.pass] = newPwr;
        }

        Unit
    }

    override fun deleteHuman(userId: String, grpCode: String) = transaction(db) {
        Humans.deleteWhere {Humans.user eq userId}

        Unit
    }

    override fun getHuman(userId: String, pass : String): List<Human>{
        val humanList : ArrayList<Human> = arrayListOf()
        transaction(db) {
            Humans.select { Humans.user eq userId }.map {
                humanList.add(
                    Human(
                        user = it[Humans.user],
                        groupId = it[Humans.groupId],
                        pass = it[Humans.pass],
                        fname = it[Humans.f_name],
                        lname = it[Humans.l_name],
                        access = it[Humans.access]
                    )
                )
            }
        }
    }

```

```

        )
    }
}

return humanList
}

override fun authentication(userId: String, pass: String) : Boolean{

    val human : ArrayList<Human> = arrayListOf()
    transaction(db) {
        Humans.select { Humans.user eq userId }.map {
            Humans.select { Humans.pass eq pass }.map {
                human.add(
                    Human(
                        user = it[Humans.user],
                        groupId = it[Humans.groupId],
                        pass = it[Humans.pass],
                        fname = it[Humans.f_name],
                        lname = it[Humans.l_name],
                        access = it[Humans.access]
                    )
                )
            }
        }
    }

    return human.isNotEmpty()
}

override fun getAllHumans() = transaction(db) {
    Humans.selectAll().map{
        Human(
            it[Humans.user],
            it[Humans.pass],
            it[Humans.f_name],
            it[Humans.l_name],
            it[Humans.groupId],
            it[Humans.access]
        )
    }
}

override fun getMedHumans(grpCode: String) = transaction(db){
    Humans.select{Humans.groupId eq grpCode}.map{
        Human(
            it[Humans.user],
            it[Humans.groupId],
            it[Humans.pass],
            it[Humans.f_name],
            it[Humans.l_name],
            it[Humans.access]
        )
    }
}

override fun createMedCenter(hidId: String, grpCode: String) = transaction(db) {
    MedCenters.insert {
        it[MedCenters.hid] = hidId;
    }
}

```

```
        it[MedCenters.groupid] = grpCode;
    }

    Unit
}

override fun deleteMedCenter(grpCode: String) = transaction(db){
    MedCenters.deleteWhere{MedCenters.groupid eq grpCode}
    Unit
}

override fun getMedCenter(grpCode: String) = transaction(db){
    MedCenters.select{MedCenters.groupid eq grpCode}.map{
        MedCenter(
            it[MedCenters.hid],
            it[MedCenters.groupid]
        )
    }.singleOrNull()
}

override fun getAllMedCenters() = transaction(db){
    MedCenters.selectAll().map{
        MedCenter(
            it[MedCenters.hid],
            it[MedCenters.groupid]
        )
    }
}

override fun getAccess(user: String) : Int {

    val human : ArrayList<Human> = arrayListOf()
    transaction(db) {
        Humans.select { Humans.user eq user }.map {
            human.add(
                Human(
                    user = it[Humans.user],
                    groupId = it[Humans.groupId],
                    pass = it[Humans.pass],
                    fname = it[Humans.f_name],
                    lname = it[Humans.l_name],
                    access = it[Humans.access]
                )
            )
        }
    }

    return human[0].access
}

override fun getName(userId: String): String{
    val human : ArrayList<Human> = arrayListOf()
    transaction(db) {
        Humans.select { Humans.user eq userId }.map {
            human.add(
                Human(
                    user = it[Humans.user],
                    groupId = it[Humans.groupId],
                    pass = it[Humans.pass],
                    fname = it[Humans.f_name],

```

```
        lname = it[Humans.l_name],
        access = it[Humans.access]
    )
    }
}

return (human[0].fname + " " + human[0].lname)
}

override fun getUserFiles(user: String): List<File>{
    val fileList : ArrayList<File> = arrayListOf()
    transaction(db){
        Files.select { Files.user eq user }.map{
            fileList.add(
                File(
                    user = it[Files.user],
                    fullName = it[Files.fullName],
                    fileName = it[Files.file],
                    hid = it[Files.hid]
                )
            )
        }
    }

    return fileList
}

override fun getMedFiles(grpCode: String): List<File>{
    val fileList : ArrayList<File> = arrayListOf()
    transaction(db){
        Files.select { Files.hid eq grpCode }.map{
            fileList.add(
                File(
                    user = it[Files.user],
                    fullName = it[Files.fullName],
                    fileName = it[Files.file],
                    hid = it[Files.hid]
                )
            )
        }
    }

    return fileList
}

override fun fileSearch(user: String, fileName: String): List<File>{
    var fileList : ArrayList<File> = arrayListOf()
    transaction(db){
        Files.select { Files.user eq user }.map{
            Files.select{Files.file.substring(0, fileName.indexOf(".")) eq
fileName}.map {
                fileList.add(
                    File(
                        user = it[Files.user],
                        fullName = it[Files.fullName],
                        fileName = it[Files.file],
```

```

        hid = it[Files.hid]
    )
    }
}

return fileList
}

override fun getSchedule() = transaction(db){
    Schedules.selectAll().map {
        Schedule(
            it[Schedules.client],
            it[Schedules.doctor],
            it[Schedules.day]
        )
    }
}

override fun getClientSchedule(user: String): List<Schedule>{
    var schedule : ArrayList<Schedule> = arrayListOf()
    transaction(db){
        Schedules.select { Schedules.client eq user }.map{
            schedule.add(
                Schedule(
                    client = it[Schedules.client],
                    doctor = it[Schedules.doctor],
                    day = it[Schedules.day]
                )
            )
        }
    }

    for (schedule in schedule) {
        schedule.doctor = getDocName(schedule.doctor)
    }

    return schedule
}

override fun getDocSchedule(doc: String): List<Schedule>{
    var schedule : ArrayList<Schedule> = arrayListOf()
    transaction(db){
        Schedules.select { Schedules.doctor eq doc }.map{
            schedule.add(
                Schedule(
                    client = it[Schedules.client],
                    doctor = it[Schedules.doctor],
                    day = it[Schedules.day]
                )
            )
        }
    }

    for (schedule in schedule) {
        schedule.client = getName(schedule.client)
    }

    return schedule
}

```

```
}

override fun getDoctor(user: String, pass: String): List<Doctor> {
    val doc : ArrayList<Doctor> = arrayListOf()
    transaction(db) {
        Doctors.select { Doctors.user eq user }.map {
            doc.add(
                Doctor(
                    user = it[Doctors.user],
                    groupId = it[Doctors.groupId],
                    pass = it[Doctors.pass],
                    name = it[Doctors.name]
                )
            )
        }
    }
    return doc
}

override fun getDoctors(grpCode: String): List<Doctor>{
    val doc : ArrayList<Doctor> = arrayListOf()
    transaction(db) {
        Doctors.select { Doctors.groupId eq grpCode }.map {
            doc.add(
                Doctor(
                    user = it[Doctors.user],
                    groupId = it[Doctors.groupId],
                    pass = it[Doctors.pass],
                    name = it[Doctors.name]
                )
            )
        }
    }
    return doc
}

override fun docAuthentication(userId: String, pass: String): Boolean{
    val doc : ArrayList<Doctor> = arrayListOf()
    transaction(db) {
        Doctors.select { Doctors.user eq userId }.map {
            Doctors.select { Doctors.pass eq pass }.map {
                doc.add(
                    Doctor(
                        user = it[Doctors.user],
                        groupId = it[Doctors.groupId],
                        pass = it[Doctors.pass],
                        name = it[Doctors.name]
                    )
                )
            }
        }
    }
    return doc.isNotEmpty()
}
```

```
}

override fun getClients(groupId : String): List<Human>{
    val humanList : ArrayList<Human> = arrayListOf()
    transaction(db){
        Humans.select { Humans.groupId eq groupId }.map{
            humanList.add(
                Human(
                    user = it[Humans.user],
                    groupId = it[Humans.groupId],
                    pass = it[Humans.pass],
                    fname = it[Humans.f_name],
                    lname = it[Humans.l_name],
                    access = it[Humans.access]
                )
            )
        }
    }

    return humanList
}

override fun getDocGroup(user: String): String{
    val grp : ArrayList<Doctor> = arrayListOf()
    transaction(db) {
        Doctors.select { Doctors.user eq user }.map {
            grp.add(
                Doctor(
                    user = it[Doctors.user],
                    groupId = it[Doctors.groupId],
                    pass = it[Doctors.pass],
                    name = it[Doctors.name]
                )
            )
        }
    }

    return grp[0].groupId
}

override fun getGroup(user: String): String{
    val human : ArrayList<Human> = arrayListOf()
    transaction(db) {
        Humans.select { Humans.user eq user }.map {
            human.add(
                Human(
                    user = it[Humans.user],
                    groupId = it[Humans.groupId],
                    pass = it[Humans.pass],
                    fname = it[Humans.f_name],
                    lname = it[Humans.l_name],
                    access = it[Humans.access]
                )
            )
        }
    }

    return human[0].groupId
}

override fun getDocName(user: String): String{
```



```

        val doc : ArrayList<Doctor> = arrayListOf()
        transaction(db) {
            Doctors.select { Doctors.user eq user }.map {
                doc.add(
                    Doctor(
                        user = it[Doctors.user],
                        groupId = it[Doctors.groupId],
                        pass = it[Doctors.pass],
                        name = it[Doctors.name]
                    )
                )
            }
        }

        return doc[0].name
    }

    override fun close(){ }
}

```

#### 6.1.2.2 Doctors.kt

```

package main.kotlin.dao
import org.jetbrains.exposed.sql.Table

/* Author: Dominic Triano
 * Date: 2/15/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * The table for Doctors, this will keep track of all of the doctor's data when
implemented
 */
*/object Doctors : Table() {
    var user = varchar("user", 50).primaryKey()
    var pass = varchar("pass", 50)
    var name = varchar("fullName", 25)
    var groupId = varchar("groupId", 50)
}

```

#### 6.1.2.3 Files.kt

```

package main.kotlin.dao
import org.jetbrains.exposed.sql.Table

/* Author: Dominic Triano
 * Date: 4/17/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * The table for files, this is linked to the Human table by user
 */
*/object Files : Table() {
    var user = varchar("user", 25) //Instead of a username, this will be the full name
of the user
    var fullName = varchar("name" , 50)
    var file = varchar("file", 50) //all of the files on file for a user
    var hid = varchar("hid" ,10)
}

```

#### 6.1.2.4 Humans.kt

```
package main.kotlin.dao
import org.jetbrains.exposed.sql.Table

/* Author: Dominic Triano
 * Date: 2/15/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 *   DAO Object for A Human (Homo-sapiens), an individual who can be observed
medically. This object will keep all of
 * their data in an easily accessible format
 *
 */
object Humans : Table() {
    var user = varchar("user", 50).primaryKey() //The username for the human
    var pass = varchar("pass", 50) //The password for the user, it will be hashed
before getting here
    var f_name = varchar("fname", 25) //User's first name
    var l_name = varchar("lname", 25) //User's last name
    var groupId = varchar("groupId", 50) //The hospital id they belong to (possibly
linking it to other tables)
    var access = integer("accesslvl") // How much power does this user have? (1 is
client, 2 is Doc, 3 is Dev)
}
```

#### 6.1.2.5 MedCenters.kt

```
package main.kotlin.dao
import org.jetbrains.exposed.sql.Table

/* Author: Dominic Triano
 * Date: 4/3/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 *   DAO object for the medical Centers, this will help keep track of what hospitals are
involved
 *
 */
object MedCenters : Table() {
    var hid = varchar("hid", 50).primaryKey()
    var groupid = varchar("groupId", 50).primaryKey()
}
```

#### 6.1.2.6 Schedules.kt

```
package main.kotlin.dao
import org.jetbrains.exposed.sql.Table

/* Author: Dominic Triano
 * Date: 4/6/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 *   DAO object for schedules, this will allow a schedule to be saved on the website for
later viewing
 *
 */
//TODO -- integrate google clendar.
object Schedules : Table() {
```

```
var client = varchar("clientId", 25)
var doctor = varchar("docId" ,25)
var day = datetime("day")
}
```

### 6.1.3 Models

#### 6.1.3.1 Doctor.kt

```
package main.kotlin.model

/* Author: Dominic Triano
 * Date: 2/15/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Data Class for creating Doctors to enter into the database
 */
data class Doctor (val user: String, var pass: String,
                  var name: String, var groupId: String)
```

#### 6.1.3.2 File.kt

```
package main.kotlin.model

/* Author: Dominic Triano
 * Date: 4/17/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Data class for creating Files to enter into the database
 */
data class File(val user: String, var fullName : String, var fileName: String,
               var hid: String)
```

#### 6.1.3.3 Human.kt

```
package main.kotlin.model

/* Author: Dominic Triano
 * Date: 4/3/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Data class for creating Humans to enter into the database
 */
data class Human(val user: String, var pass: String,
                var fname: String, var lname: String,
                var groupId: String, val access: Int)
```

#### 6.1.3.4 MedCenter.kt

```
package main.kotlin.model

/* Author: Dominic Triano
 * Date: 4/3/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Data class for creating medCenters to enter into the database
```

```

*
*/

data class MedCenter(var hid: String, var groupId: String)

```

#### 6.1.3.5 Schedule.kt

```

package main.kotlin.model

/* Author: Dominic Triano
 * Date: 4/3/2020
 * Language: Kotlin
 * Project: Halo
 * Description:
 * Data class for creating medCenters to enter into the database
 */

data class MedCenter(var hid: String, var groupId: String)

```

### 6.1.4 Freemarker Templates

#### 6.1.4.1 Dev.ftl

```

<#import "template.ftl" as layout />

<body>
<style type="text/css">
h1 {
    font: bold italic 3em/2em "Times New Roman", "MS Serif", "New York", serif;
    margin: 0;
    padding: 0;
    color: #00ffff;
    border-top: solid #00ffff medium;
    border-bottom: dotted #00ffff thin;
    width: 1350px;
    text-align:center;
}

{box-sizing: border-box}
body {font-family: "Lato", sans-serif;}

/* Style the tab */
.tab {
    float: left;
    border: 1px solid #ccc;
    background-color: #f1f1f1;
    width: 30%;
    height: 300px;
}

/* Style the buttons inside the tab */
.tab button {
    display: block;
    background-color: inherit;
    color: black;
    padding: 22px 16px;
    width: 100%;
    border: none;
    outline: none;
    text-align: left;
    cursor: pointer;
}

```

```

    transition: 0.3s;
    font-size: 17px;
}

/* Change background color of buttons on hover */
.tab button:hover {
    background-color: #ddd;
}

/* Create an active/current "tab button" class */
.tab button.active {
    background-color: #ccc;
}

/* Style the tab content */
.tabcontent {
    float: left;
    padding: 0px 12px;
    border: 1px solid #ccc;
    width: 70%;
    border-left: none;
}
</style>

<@layout.mainLayout>
<#assign x = 0>
<#list staff as dev>
<#if x < 1>
<h1>Welcome ${dev.fname} ${dev.lname}</h1>
</#if>
<#assign x = 1>
</#list>

<div class="tab">
    <button class="tablinks" onclick="openTab(event, 'Clients', 'silver')"
id="defaultOpen">Clients</button>
    <button class="tablinks" onclick="openTab(event, 'newUser', 'silver')">New
User</button>
    <button class="tablinks" onclick="openTab(event, 'Home' , 'silver')">Back
Home</button>
</div>

<div id="Clients" class="tabcontent">
    <table class="table">
        <thead class="thead-dark">
            <tr>
                <th scope="col">Users</th>
                <th>
                    <input class="form-control" type="text" placeholder="Search" aria-
label="Search">
                </th>
            </tr>
        </thead>
        <tbody>
            <#list humans as human>
            <tr>
                <td>${human.fname} ${human.lname}</td>
                <td>
                    <a href="/editHuman?name=${human.user}" class="btn btn-secondary
float-right mr-2" role="button">Edit User</a>
                </td>
            </tr>
            </tbody>
        </table>
    </div>

```

```

        </tr>
      </#list>
    </tbody>
  </table>
</div>

<div id="newUser" class="tabcontent">
  <h3>New User</h3>
  <p>Please enter the details of the new user:</p>
  <form action="/newHuman" method="post" class="form-signin">
    <div class="form-group">
      <label for="type">Select The User Type:</label>
      <select id="type">
        <option value="doc">Doctor</option>
        <option value="human">Client</option>
      </select>
    </div>
    <div class="row">
      <div class="col-md-6 mb-3">
        <label for="firstName">First name</label>
        <input type="text" class="form-control" id="firstName" placeholder=""
value="" required>
        <div class="invalid-feedback">
          Valid first name is required.
        </div>
      </div>
      <div class="col-md-6 mb-3">
        <label for="lastName">Last name</label>
        <input type="text" class="form-control" id="lastName" placeholder=""
value="" required>
        <div class="invalid-feedback">
          Valid last name is required.
        </div>
      </div>
    </div>
    <div class="mb-3">
      <label for="username">Username</label>
      <div class="input-group">
        <input type="text" class="form-control" id="username"
placeholder="Username" required>
        <div class="invalid-feedback" style="width: 100%;">
          Your username is required.
        </div>
      </div>
    </div>
    <div class="mb-3">
      <label for="password">Password</label>
      <div class="input-group">
        <input type="password" class="form-control" id="password"
placeholder="Password" required>
        <div class="invalid-feedback" style="width: 100%;">
          Your username is required.
        </div>
      </div>
    </div>
    <!-- <div class="mb-3">-->
    <!-- <label for="email">Email <span class="text-
muted">(Optional)</span></label>-->

```

```

        <!--                <input type="email" class="form-control" id="email"
placeholder="you@example.com">-->
        <!--                </div>-->

        <div class="mb-3">
            <label for="grpCode">Medical Group Code</label>
            <input type="text" class="form-control" id="grpCode" placeholder="ex.)
GHP" required>
        </div>

        <div class="mb-3">
            <label for="access">Access Level</label>
            <input type="text" class="form-control" id="access" min="1" max="3"
required>
        </div>
        <hr class="mb-4">
        <button type="submit" class="btn btn-primary">Submit</button>
    </form>
</div>

<div id="Home" class="tabcontent text-center">
    <h3 style="text-align:center;">Are You Sure You Wish To Return Home?</h3>
    <a href = "/" class="btn btn-primary btn-lg" >Continue</a>
</div>

<script>
function openTab(evt, tabName) {
    var i, tabcontent, tablinks;
    tabcontent = document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }
    tablinks = document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
        tablinks[i].className = tablinks[i].className.replace(" active", "");
    }
    document.getElementById(tabName).style.display = "block";
    evt.currentTarget.className += " active";
}

// Get the element with id="defaultOpen" and click on it
document.getElementById("defaultOpen").click();
</script>
</body>
</@layout.mainLayout>

```

#### 6.1.4.2 Home.ftl

```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="With use of the bootstrap resources,
    this home page for the Halo project has been made">
    <meta name="author" content="Dominic Triano">
    <link rel="stylesheet" href="../static/bootstrap.css">
    <title>Halo Homepage</title>

```

```

    <!-- Bootstrap core CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet">

    <!-- Custom styles for this template -->
    <style>
    body {
        padding-top: 50px;
    }
    .starter-template {
        padding: 40px 15px;
        text-align: center;
    }
    </style>

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
    <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
</head>

<body>
<div class="jumbotron">
    <div class="container">
        <h1>Project Halo</h1>
        <p>
            Welcome to Project Halo! This project was dreamed up in 2018 and
            finished in April 2020 by Dominic Triano
            for his Capstone at SUNY Polytechnic Institute. The purpose of this
            project is to create a standardized
            system for medical centers to use to document patients and allow them
            access to their data in a fast and
            easy way. Look around to learn more about us, or go straight to sign-in
            below. </p>
            <p>
                <a href = "/signIn" class="btn btn-primary btn-lg">Sign In</a>
            </p>
        </div>
    <div class="container"></div>
</div>

<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <a class="navbar-brand" href="#">Halo</a>
        </div>
        <div class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li class="active"><a href="#">Home</a></li>
                <li><a href="#about">Tabs coming soon</a></li>
            </ul>
        </div>
    </div>
</div>

<div class="container">
    <!-- Example row of columns -->
    <div class="row">
        <div class="col-md-4">
            <h2>Fast</h2>
            <p>Instead of having to put in requests for your data and have to wait

```



```

days,
        just get a sign in from your doctor and have access to all your data
instantly! </p>
    </div>
    <div class="col-md-4">
        <h2>Easy</h2>
        <p>With Halo's interface, it is easy to understand and to find all of the
data you need! If you need any help, just contact our support! </p>
    </div>
    <div class="col-md-4">
        <h2>Accessible</h2>
        <p>Being a website, this is easily accessible from any platform at any
time. There is no Additional software needed. </p>
    </div>
</div>
</div>
<p> </p>

<div class="container">
    <h3>Client Features:</h3>
    <ul class="list-group list-group-flush">
        <li class="list-group-item">File Download</li>
        <li class="list-group-item">Schedule Viewing</li>
    </ul>
</div>
<p> </p>

<div class="container">
    <h3>Staff Features:</h3>
    <ul class="list-group list-group-flush">
        <li class="list-group-item">File Download</li>
        <li class="list-group-item">Schedule Viewing</li>
        <li class="list-group-item">Patient Creation</li>
    </ul>
</div>
<p> </p>

<div class="container">
    <h3>Upcoming Features:</h3>
    <ul class="list-group list-group-flush">
        <li class="list-group-item">Appointment Creation</li>
        <li class="list-group-item">File Upload</li>
        <li class="list-group-item">Mobile Application</li>
        <li class="list-group-item">Google Verification</li>
        <li class="list-group-item">Contact Directory</li>
    </ul>
</div>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</body>

```

```
</html>
```

### 6.1.4.3 index.ftl

```
<#import "template.ftl" as layout />

<body>
<style type="text/css">
h1 {
    font: bold italic 3em/2em "Times New Roman", "MS Serif", "New York", serif;
    margin: 0;
    padding: 0;
    color: #00ffff;
    border-top: solid #00ffff medium;
    border-bottom: dotted #00ffff thin;
    width: 1350px;
    text-align:center;
}

{box-sizing: border-box}
body {font-family: "Lato", sans-serif;}

/* Style the tab */
.tab {
    float: left;
    border: 1px solid #ccc;
    background-color: #f1f1f1;
    width: 30%;
    height: 300px;
}

/* Style the buttons inside the tab */
.tab button {
    display: block;
    background-color: inherit;
    color: black;
    padding: 22px 16px;
    width: 100%;
    border: none;
    outline: none;
    text-align: left;
    cursor: pointer;
    transition: 0.3s;
    font-size: 17px;
}

/* Change background color of buttons on hover */
.tab button:hover {
    background-color: #ddd;
}

/* Create an active/current "tab button" class */
.tab button.active {
    background-color: #ccc;
}

/* Style the tab content */
.tabcontent {
    float: left;
    padding: 0px 12px;
    border: 1px solid #ccc;
```

```

width: 70%;
border-left: none;
}
</style>

<@layout.mainLayout>
<#assign x = 0>
<#list user as use>
<#if x < 1>
<h1>Welcome ${use.fname} ${use.lname}</h1>
</#if>
<#assign x = 1>
</#list>

<div class="tab">
    <button class="tablinks" onclick="openTab(event, 'Files', 'silver')"
id="defaultOpen">Files</button>
    <button class="tablinks" onclick="openTab(event, 'Appointments',
'silver')">Appointments</button>
    <button class="tablinks" onclick="openTab(event, 'Home' , 'silver')">Back
Home</button>
</div>

<div id="Files" class="tabcontent">
    <table class="table">
        <thead class="thead-dark">
            <tr>
                <th scope="col">File Names</th>
                <th>
                    <input class="form-control" type="text" placeholder="Search" aria-
label="Search">
                </th>
            </tr>
        </thead>
        <tbody>
            <#list files as file>
            <tr>
                <td>${file.fileName}</td>
                <td>
                    <a
href="/download?action=download&name=${file.fileName}&user=${file.user}" class="btn
btn-secondary float-right mr-2" role="button">Download</a>
                </td>
            </tr>
            </#list>
        </tbody>
    </table>
</div>

<div id="Appointments" class="tabcontent">
    <h3>New Appointments</h3>
    <p>Please enter the details of the appointment you would like to make:</p>
    <form action="/appointment" method="post">
        <div class="form-group">
            <label for="time">Appointment (Date and Time):</label>
            <input type="datetime-local" class="form-control" id="time" name="time">
        </div>
        <div class="form-group">
            <label for="doctor">Doctor:</label>
            <select id="doctor">
                <#list doctors as doc>

```

```

        <option value="${doc.user}">${doc.name}</option>
    </#list>
</select>
</div>
<div class="form-group">
    <label for="desc">Description:</label>
    <input type="text" class="form-control" id="desc" name="desc"
placeholder="What is the purpose of this visit?">
</div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>

<p><br><br></p>
<table class="table">
    <thead class="thead-dark">
        <tr>
            <th scope="col">Current Appointments</th>
        </tr>
    </thead>
    <tbody>
        <#list schedules as schedule>
            <tr>
                <td>${schedule.doctor} ${schedule.day}</td>
            </tr>
        </#list>
    </tbody>
</table>
</div>

<div id="Home" class="tabcontent text-center">
    <h3 style="text-align:center;">Are You Sure You Wish To Return Home?</h3>
    <a href = "/" class="btn btn-primary btn-lg" >Continue</a>
</div>

<script>
function openTab(evt, tabName) {
    var i, tabcontent, tablinks;
    tabcontent = document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }
    tablinks = document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
        tablinks[i].className = tablinks[i].className.replace(" active", "");
    }
    document.getElementById(tabName).style.display = "block";
    evt.currentTarget.className += " active";
}

// Get the element with id="defaultOpen" and click on it
document.getElementById("defaultOpen").click();
</script>
</body>
</@layout.mainLayout>

```

#### 6.1.4.4 humans.ftl

```

<#import "template.ftl" as layout />

<body>
<style type="text/css">

```

```
h1 {
  font: bold italic 3em/2em "Times New Roman", "MS Serif", "New York", serif;
  margin: 0;
  padding: 0;
  color: #00ffff;
  border-top: solid #00ffff medium;
  border-bottom: dotted #00ffff thin;
  width: 1350px;
  text-align:center;
}

{box-sizing: border-box}
body {font-family: "Lato", sans-serif;}

/* Style the tab */
.tab {
  float: left;
  border: 1px solid #ccc;
  background-color: #f1f1f1;
  width: 30%;
  height: 300px;
}

/* Style the buttons inside the tab */
.tab button {
  display: block;
  background-color: inherit;
  color: black;
  padding: 22px 16px;
  width: 100%;
  border: none;
  outline: none;
  text-align: left;
  cursor: pointer;
  transition: 0.3s;
  font-size: 17px;
}

/* Change background color of buttons on hover */
.tab button:hover {
  background-color: #ddd;
}

/* Create an active/current "tab button" class */
.tab button.active {
  background-color: #ccc;
}

/* Style the tab content */
.tabcontent {
  float: left;
  padding: 0px 12px;
  border: 1px solid #ccc;
  width: 70%;
  border-left: none;
}
</style>

<@layout.mainLayout>
<#assign x = 0>
<#list files as file>
```

```

<#if x < 1>
<h1>${file.fullName}'s Files</h1>
</#if>
<#assign x = 1>
</#list>
    <table class="table">
        <thead class="thead-dark">
            <tr>
                <th scope="col">File Names</th>
                <th>
                    <input class="form-control" type="text" placeholder="Search" aria-
label="Search">
                </th>
            </tr>
        </thead>
        <tbody>
            <#list files as file>
            <tr>
                <td>${file.fileName}</td>
                <td>
                    <a
href="/download?action=download&name=${file.fileName}&user=${file.user}" class="btn
btn-secondary float-right mr-2" role="button">Download</a>
                </td>
            </tr>
            </#list>
        </tbody>
    </table>
</body>
</@layout.mainLayout>

```

#### 6.1.4.5 SignIn.ftl

```

<!doctype html>
<html lang="en">
<!--Css Styles-->
<style>
    .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
    }

    @media (min-width: 768px) {
        .bd-placeholder-img-lg {
            font-size: 3.5rem;
        }
    }

    body {
        height: 100%;
    }

    body {
        align-items: center;
        text-align: center;
        display: -ms-flexbox;

```

```

        display: flex;
        -ms-flex-align: center;
        padding-top: 40px;
        padding-bottom: 40px;
        background-color: #f5f5f5;
    }
    .form-signin {
        margin: 20px auto;
        width: 100%;
        max-width: 320px;
        padding: 15px;
    }
    .form-signin .checkbox {
        font-weight: 400;
    }
    .form-signin .form-control {
        position: relative;
        box-sizing: border-box;
        height: auto;
        padding: 10px;
        font-size: 16px;
    }
    .form-signin .form-control:focus {
        z-index: 2;
    }
    .form-signin input[type="email"] {
        margin-bottom: -1px;
        border-bottom-right-radius: 0;
        border-bottom-left-radius: 0;
    }
    .form-signin input[type="password"] {
        margin-bottom: 10px;
        border-top-left-radius: 0;
        border-top-right-radius: 0;
    }
</style>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <title>Sign-in</title>

    <!-- Bootstrap core CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet">

    <meta name="theme-color" content="#563d7c">
</head>
<body class="text-center">
    <form action="/signIn" method="post" class="form-signin">
        
        <h1 class="h3 mb-3 font-weight-normal">Please sign in</h1>
        <label for="inputUser" class="sr-only">Username</label>
        <input type="text" id="inputUser" name="inputUser" class="form-control"
placeholder="Username" required autofocus>
        <label for="inputPassword" class="sr-only">Password</label>
        <input type="password" id="inputPassword" name="inputPassword" class="form-

```

```

control" placeholder="Password" required>
  <button class="btn btn-lg btn-primary btn-block" type="submit" name="action"
value="signIn">Sign in</button>
</form>
<script>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
<script>
  <script src=js/bootstrap.min.js>
</script>
</body>
</html>

```

#### 6.1.4.6 Staff.ftl

```

<!doctype html>
<html lang="en">
<!--Css Styles-->
<style>
  .bd-placeholder-img {
    font-size: 1.125rem;
    text-align: middle;
    -webkit-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
  }

  @media (min-width: 768px) {
    .bd-placeholder-img-lg {
      font-size: 3.5rem;
    }
  }

  body {
    height: 100%;
  }

  body {
    align-items: center;
    text-align: center;
    display: -ms-flexbox;
    display: flex;
    -ms-flex-align: center;
    padding-top: 40px;
    padding-bottom: 40px;
    background-color: #f5f5f5;
  }

  .form-signin {
    margin: 20px auto;
    width: 100%;
    max-width: 320px;
    padding: 15px;
  }

  .form-signin .checkbox {
    font-weight: 400;
  }

  .form-signin .form-control {
    position: relative;
    box-sizing: border-box;

```



```

        height: auto;
        padding: 10px;
        font-size: 16px;
    }
    .form-signin .form-control:focus {
        z-index: 2;
    }
    .form-signin input[type="email"] {
        margin-bottom: -1px;
        border-bottom-right-radius: 0;
        border-bottom-left-radius: 0;
    }
    .form-signin input[type="password"] {
        margin-bottom: 10px;
        border-top-left-radius: 0;
        border-top-right-radius: 0;
    }
}
</style>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Sign-in</title>

    <!-- Bootstrap core CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet">

    <meta name="theme-color" content="#563d7c">

</head>
<body class="text-center">
    <form action="/signIn" method="post" class="form-signin">
        
        <h1 class="h3 mb-3 font-weight-normal">Please sign in</h1>
        <label for="inputUser" class="sr-only">Username</label>
        <input type="text" id="inputUser" name="inputUser" class="form-control"
placeholder="Username" required autofocus>
        <label for="inputPassword" class="sr-only">Password</label>
        <input type="password" id="inputPassword" name="inputPassword" class="form-
control" placeholder="Password" required>
        <button class="btn btn-lg btn-primary btn-block" type="submit" name="action"
value="signIn">Sign in</button>
    </form>
<script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
<script>
    <script src=js/bootstrap.min.js>
</script>
</body>
</html>

```

#### 6.1.4.7 Template.ftl

```

<#macro mainLayout title="Welcome to Halo Database">
<!doctype html>
<html lang="en">
    <head>
        <title>${title}</title>
    </head>

```

```

<!-- Required meta tags -->
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
</head>
<body>
<div class="container">
<div class="row m-1">
<#nested/>
</div>
</div>
</body>
</html>
</#macro>

```

## 6.2 SAMPLE INTERFACES

### 6.2.1 Prototype Interface



A prototype login page with a red background. It features a title "Patient Login Page" in bold black text. Below the title is a text input field containing "jsmith123". Underneath that is a password input field with black dots and a toggle icon on the right. At the bottom left is a "Login" button.

Figure 7. Prototype Login



A prototype patient portal home page. At the top left is the text "Patient Portal" in a large, stylized red font. At the top right is the DMC logo, which consists of a caduceus and the letters "DMC". Below the header is a navigation bar with four tabs: "Home", "Personal Info.", "Appointments", and "Medical Records". The "Home" tab is selected. Below the navigation bar is a red horizontal bar. Underneath that is a "Welcome!" section with the text "Hello John Smith!". Below this is a message: "Please use the tabs to navigate the portal. If you have any concerns, please contact our helpdesk at help@dmc.com. If you would like to return to the homepage, click here." At the bottom left of this section is a "Home" button.

Figure 8. Prototype Patient Portal Home

## Patient Portal



Hello John Smith!

|      |                |              |                 |  |
|------|----------------|--------------|-----------------|--|
| Home | Personal Info. | Appointments | Medical Records |  |
|------|----------------|--------------|-----------------|--|

**Personal Information:**

Age: 20

Weight: 164

Diagnosis: Car Accident

Treatment: Surgery on right ankle and stitch up mult. lac

If you think your information is incorrect, please contact your doctor.

Figure 9. Prototype Patient Portal Personal Info

### 6.2.2 Final Interface

|      |      |                  |
|------|------|------------------|
| Halo | Home | Tabs coming soon |
|------|------|------------------|

# Project Halo

Welcome to Project Halo! This project was dreamed up in 2018 and finished in April 2020 by Dominic Triano for his Capstone at SUNY Polytechnic Institute. The purpose of this project is to create a standardized system for medical centers to use to document patients and allow them access to their data in a fast and easy way. Look around to learn more about us, or go straight to sign-in below.

[Sign In](#)

### Fast

Instead of having to put in requests for your data and have to wait days, just get a sign in from your doctor and have access to all your data instantly!

### Easy

With Halo's interface, it is easy to understand and to find all of the data you need! If you need any help, just contact our support!

### Accessible

Being a website, this is easily accessible from any platform at any time. There is no Additional software needed.

**Client Features:**

|                  |
|------------------|
| File Download    |
| Schedule Viewing |

**Staff Features:**

|                  |
|------------------|
| File Download    |
| Schedule Viewing |
| Patient Creation |

**Upcoming Features:**

|                      |
|----------------------|
| Appointment Creation |
| File Upload          |
| Mobile Application   |
| Google Verification  |
| Contact Directory    |

Figure 10. Home Page



Please sign in

Username

Password

! Please fill out this field.

Sign in

Figure 11. Login

**Welcome John Smith**

|              |   |
|--------------|---|
| Files        | <b>File Names</b> <input type="text" value="Search"/> |
| Appointments | HeartScan.jpg <a href="#">Download</a>                |
| Back Home    | ShoulderSurgeryPictures.pdf <a href="#">Download</a>  |

Figure 12. Client Portal

---

## Welcome John Smith

|              |   |
|--------------|---|
| Files        | <h3>New Appointments</h3> <p>Please enter the details of the appointment you would like to make:</p> <p>Appointment (Date and Time):</p> <input type="text" value="mm/dd/yyyy --:-- --"/> |
| Appointments |   |
| Back Home    |   |

Doctor:

Description:

**Current Appointments**

T.C. Callahan 2020-05-03T14:00:00.000-04:00

Figure 13. Client Appointments Tab

---

## Welcome John Smith

|              |  |
|--------------|--|
| Files        | <h3>Are You Sure You Wish To Return Home?</h3> <input type="button" value="Continue"/> |
| Appointments |  |
| Back Home    |  |

Figure 14. Client Return to Home Tab

## Welcome T.C. Callahan

|  |                 |                                     |
|--|-----------------|-------------------------------------|
| Clients<br><br>Appointments<br><br>Back Home | <b>Patients</b> | <input type="text" value="Search"/> |
|  | John Smith      | <a href="#">See Files</a>           |
|  | John Doe        | <a href="#">See Files</a>           |

Figure 15. Staff Portal

## John Smith's Files

|                             |                                     |
|-----------------------------|-------------------------------------|
| <b>File Names</b>           | <input type="text" value="Search"/> |
| HeartScan.jpg               | <a href="#">Download</a>            |
| ShoulderSurgeryPictures.pdf | <a href="#">Download</a>            |

Figure 16. Staff Client Select

## Welcome T.C. Callahan

[Clients](#)[Appointments](#)[Back Home](#)

### New Appointments

Please enter the details of the appointment you would like to make:

Appointment (Date and Time):

Client:

Description:

### Current Appointments

John Smith 2020-05-03T14:00:00.000-04:00

Figure 17. Staff Appointments Tab

## Welcome T.C. Callahan

[Clients](#)[Appointments](#)[Back Home](#)

Are You Sure You Wish To Return Home?

Figure 18. Staff Return to Home Tab

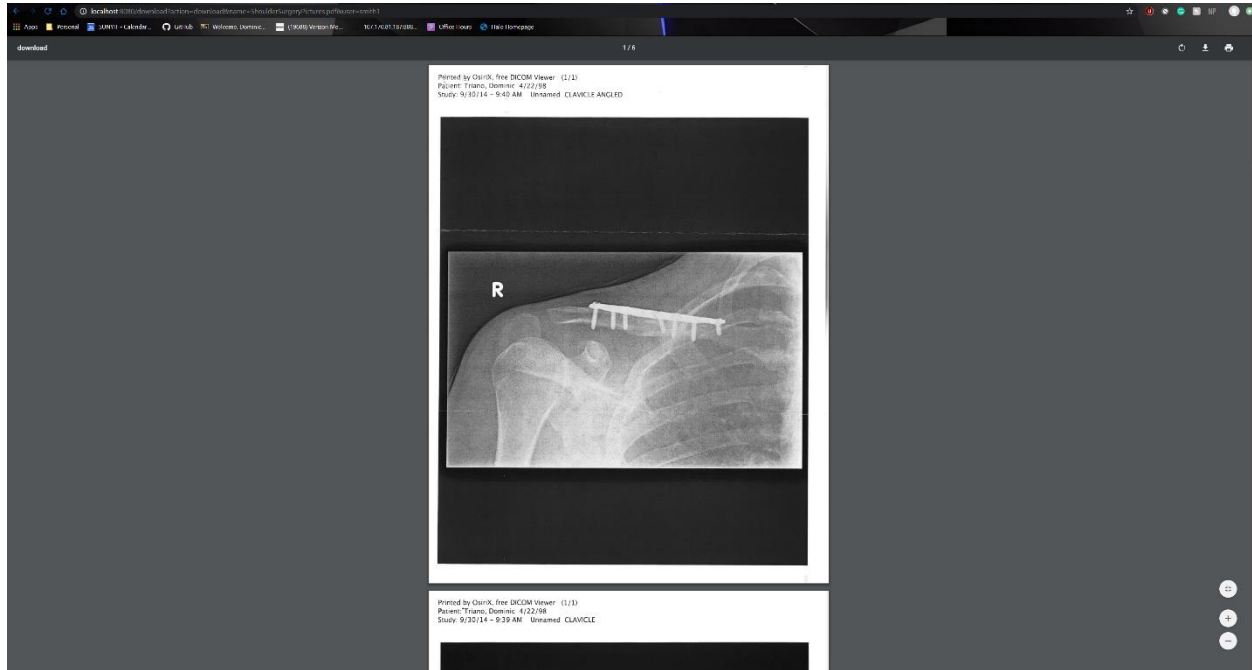


Figure 19. File Display and Download



Figure 20. Halo Logo (unfinalized)



### 6.3 USER GUIDE

Good Day! Thank you for choosing HALO! Our system is simple and easy. To get started, make sure your medical institution has created a username and password for you.

1. Go to the HALO home page in your web browser and click the login button
2. Enter the credentials you have been given and click sign in
3. Once you have landed on your portal, select the file you wish to download
  - a. The file should now be displayed in your browser
4. Click the download button in the top right-hand corner or right-click the image and click save as

And that's it! If you have any questions, please email the developer at [d.triano0100@gmail.com](mailto:d.triano0100@gmail.com).

Please stay with us! We will be adding new features in the future.

### 6.4 INFORMATION FLOW

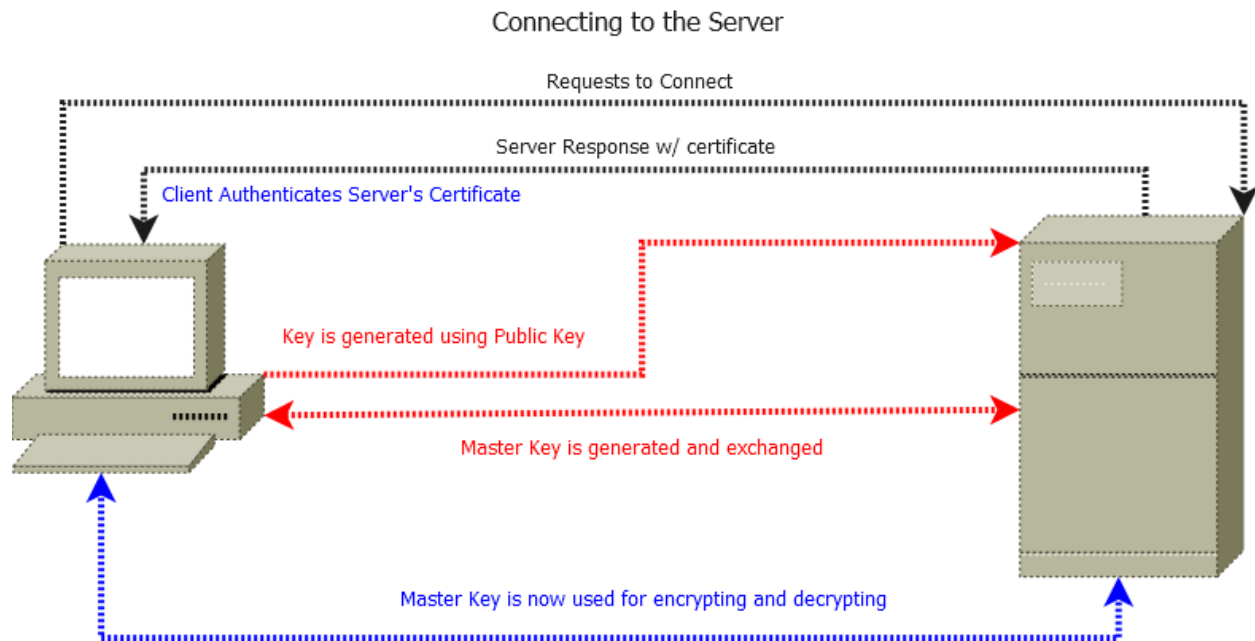


Figure 21. Server Handshake (Created in DIA)

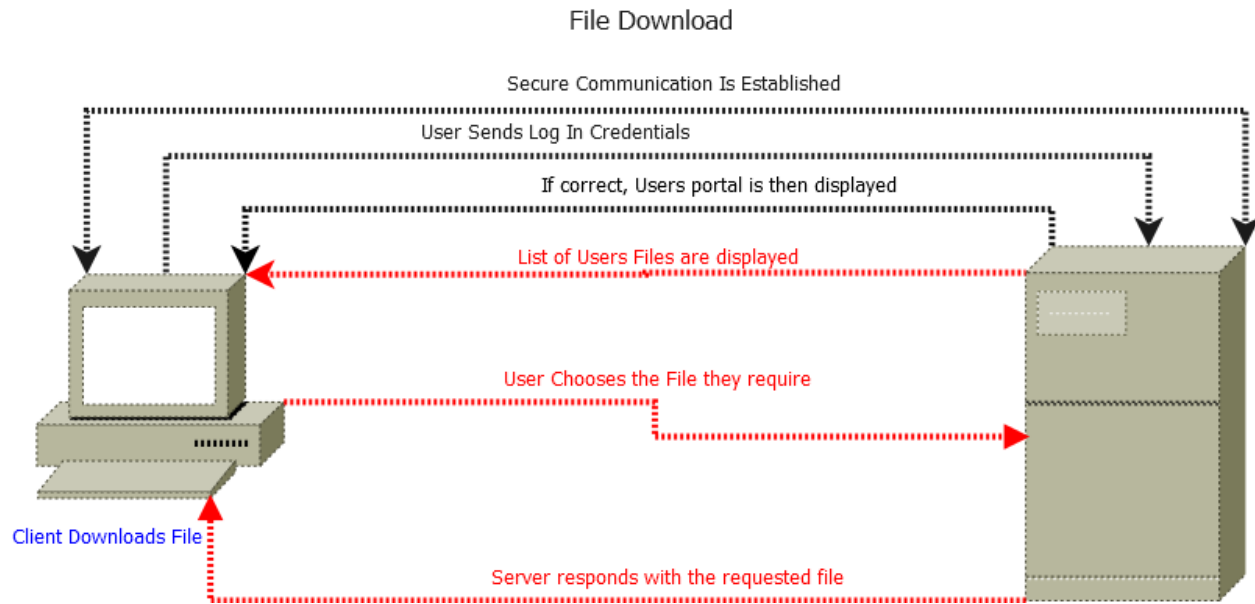


Figure 22. File Download (Created in DIA)

## 6.5 REFERENCES

Directives. (2020, March 6). Retrieved from

[https://freemarker.apache.org/docs/pgui\\_datamodel\\_directive.html](https://freemarker.apache.org/docs/pgui_datamodel_directive.html)

import. (2020, March 6). Retrieved from

[https://freemarker.apache.org/docs/ref\\_directive\\_import.html](https://freemarker.apache.org/docs/ref_directive_import.html)

JetBrains. (2018). Quick Start Website. Retrieved from <https://ktor.io/quickstart/guides/website.html>

JetBrains. (2018). Servers Routing. Retrieved from <https://ktor.io/servers/features/routing.html>

JetBrains. (2018). Servers Responses. Retrieved from <https://ktor.io/servers/calls/responses.html>

JetBrains. (2018). Servers Requests. Retrieved from <https://ktor.io/servers/calls/requests.html>

JetBrains. (2020, April 29). JetBrains/Exposed. Retrieved from

<https://github.com/JetBrains/Exposed>

Template loading. (2020, March 6). Retrieved from

[https://freemarker.apache.org/docs/pgui\\_config\\_templateloading.html](https://freemarker.apache.org/docs/pgui_config_templateloading.html)

What is Apache FreeMarker™? (2020, March 7). Retrieved from <https://freemarker.apache.org/>

## 6.6 POSTER



Figure 23. HALO Poster