Constructing Server For Real Time ECG Monitoring Application Using Java Spring Boot

by
Zhilun Liu

M.Eng, University of Victoria, 2020

A Project Report Submitted in Partial fulfillment of the

Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

SUPERVISORY COMMITTEE

Constructing server for ECG monitoring application

Using Java Spring Boot

by

Zhilun Liu

M.Eng, University of Victoria, 2020

Supervisory Committee

Dr. XiaoDai Dong, Department of Electrical and Computer Engineering, University of Victoria (Departmental Member)

Dr. Issa Traore, Department of Electrical and Computer Engineering, University of Victoria (Departmental Member)

ABSTRACT

In this report, we describe several key elements to construct a safe and efficient server for remote electrocardiogram monitoring. In particular, we will focus on the RESTful API designing rules, database table constructing rules, and the network security of the server. By using Java Spring Boot and its sub-frameworks Spring data JPA, which helps to simplify the interaction between server application and database, and Spring security, which constructs a basic security structure including authentication and authorization layers, we provide the implementation of APIs, the detailed security requirements and risk management. The details of database structures are presented. This report also explains the procedure of ECG data animation in the front-end application.

Table of Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Xiao-dai Dong for her continuous support and her invaluable suggestions without which I could not have completed this project. Her patience and encouragement have backed me up the whole time, and she has always been open and honest to me, I would have never completed my degree without her supervision.

In addition, I would like to thank my parents and my friends for their motivation and support through my studies.

Abbreviations

ECG	electrocardiogram
JWT	Json Web Token
RESTful	Resource Representational State Transfer
API	Application Programming Interface
DAL	Data access Layer
DAO	Data access object
DB	database
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
RFC	Request for Comments
3NF	Third Normal Form
PK	primary key
NN	non-nullable
AI	auto-increment
JPA	Java Persistence API
JSON	JavaScript Object Notation
CSRF	Cross-site request forgery
XSS	Cross-site scripting
URL	Uniform Resource Locator
WinForm	Windows Forms
FPS	frame per second
JSON	JavaScript Object Notation

Chapter 1 Introduction

With the development of medical knowledge, there is an increasing demand for more convenient and efficient ways to do the medical testing, like the ECG testing. People used to go to the clinic and take half of the day to get one ECG test, which is inconvenient and inefficient. Therefore, a mobile ECG monitoring application could solve this problem so that the patient does not even need to go to the clinic. Patients can make appointments with the clinic and pick up the ECG sensor device and the mobile phone with the phone application at a specific location. Then the patient can make a video call to the clinic to start hook up including that the nurse guides the patient how to attach the sensor, check the connection and the dynamic ECG graph. After that, patients can wear the device and continue their day with work or sleep. Finally the doctor can analyze the ECG graph through the application and generate a report and send it by mail or email to the patient.

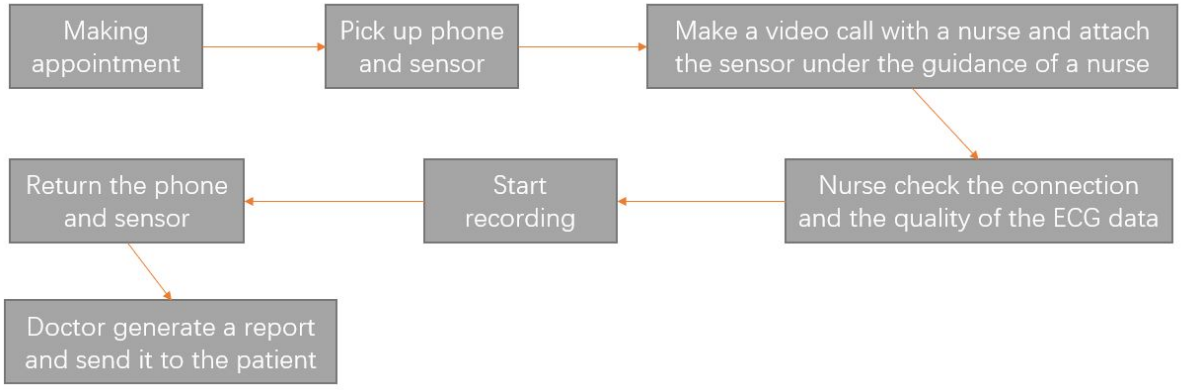

Fig. 1 The process of a complete ECG test

There are several modules in order to finish a series of operations above, which are sensor, phone application, server, and windows application. Fig. 2 shows the process of the whole ECG monitoring system. Sensor is a small device that attaches to the patient and transfers the ECG data to the phone application via bluetooth. Then the phone application

stores the data temporarily until sent to the server. Server handles and stores data sent from different patients and sends it to the windows application whenever needed. There are several other operations like nurse login and registration, creating or searching patients, creating, updating or searching the appointments, etc. We need an efficient and safe back-end to implement all those functions above, including a safe database architecture, a complete risk management documentation and a well designed API library. We will introduce the details of how we build an efficient server of our ECG monitoring project in the rest parts of this report.

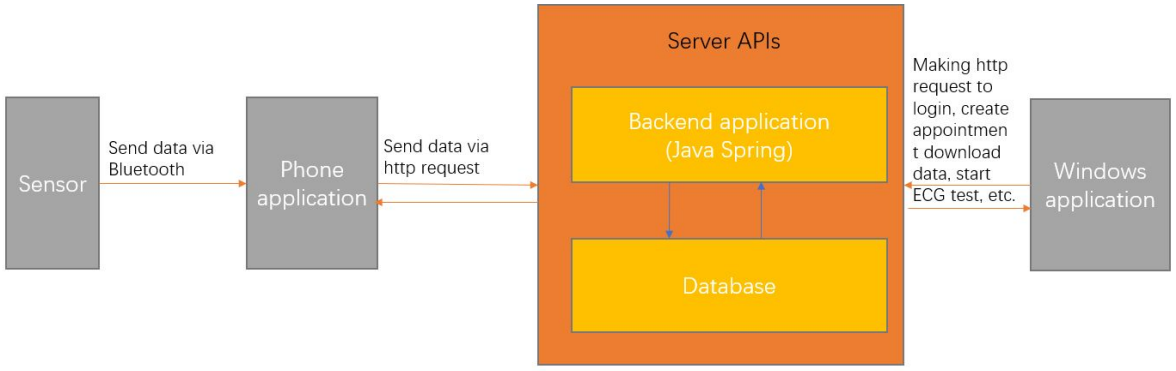

Fig. 2 The process of whole ECG monitoring system

1.1 Introduction to Backend Architecture

The back-end is all the components required to process the request from a client and send the response to the client, and it usually includes four major parts [15]:

- The server, which is the computer (machine) that receives requests
- The APIs, which is the defined methods of communication between server and client

- The back-end application, which is the application running on the server which handles all the requests including retrieving information from the database and generating the response.
- The database, which is used to store and organize the data.

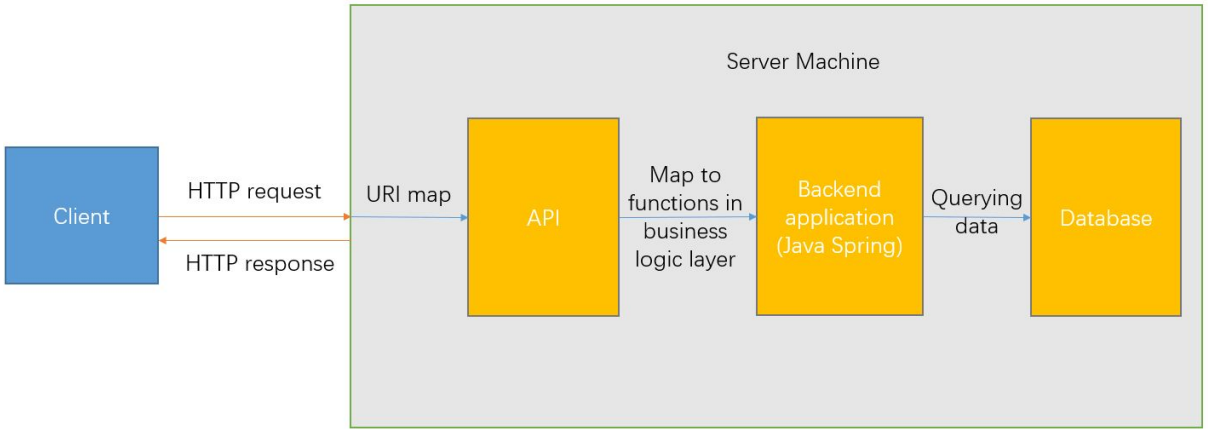

Fig. 3 Backend architecture

Firstly, we build a server by choosing a suitable platform and operating system to support the deployment of our back-end application. As the server interacts with external requests from phones and computers, there is a chance that hackers with ulterior motives put backdoors or Trojan horse programs on our servers [18]. Therefore, many small application developers choose to deploy their application on the server agent company like Amazon because it can provide stable and secure services. We currently deploy the server on Computer Canada's machine with linux operating system because it can provide the basic security features and it is not a private machine so people have to be granted permission before doing any modification to the system.

Database is also a significant part of the back-end. It provides an interface to save data in a persistent way in memory, which reduces the load on the main memory of the server and prevents data loss when the server crashes or loses power. Most of the requests sent to the

server involve database query, which means the client requests information that is stored in the database or submits data with their request and uploads into the database [15]. Moreover we build the database according to the third normal form to reduce data redundancy, increase data quality, and capture complete business requirements.

After that, we design the APIs to meet user requirements and transfer data efficiency at the same time. API defines the kinds of calls or requests that can be made, how to make them, the data formats of the input of requests that should be used, the conventions to follow, etc [16]. API like a menu which provides a list of dishes (operations) one can order, along with a description of each dish (required input information). When one specifies what menu items he/she wants, the kitchen (back-end application) does the work and provides one with some finished dishes (outputs). In this case the user does not know exactly how the kitchen (back-end application) prepares that food, which is more efficient and safer [17]. For this project, we design our APIs according to the RESTful (representational state transfer) API designing rules, which is an ideal choice because of its stateless property, which makes it more convenient and safe. The details of API implementation according to the RESTful designing rules are given in Chapter 5 .

The back-end application contains logic about how to respond to the requests of different APIs. Developers usually use existing frameworks to simplify the logic of routing (matching a HTTP verb (method) and a URI (Uniform Resource Identifier) based on a request). We choose Spring Boot as the basic framework to simplify the work since it can configure and route the back-end application automatically, and save us plenty of time on writing query statements to interact with the database when getting or creating information by using its sub-framework like Spring Data JPA.

Since we use Spring Boot as our basic framework to develop the back end application, the application will be formed by a 3-tier architecture, which means dividing the application into three parts: Model, Repository, and Resource, representing Entity class, DAL (data access layer) and Business logic layer respectively. The Entity class is an object wrapper for a database table, which means that the columns in the database table will be mapped to variables of an entity class. Data access layer accesses the data in the database by interacting with the entity class to implement the reading and saving operations of data in the database. The business logic layer will implement the actual logic of every APIs, like reading the specific data in the database or creating and saving new data into the database. The details of the structure of Spring Java are given in Chapter 3.

1.2 Security

We used Spring Security to implement most of our security functions. As we know, the security of the application mainly contains two parts, authentication and authorization . This is also the main target area of Spring Security. To implement authentication, the server gives the accessor a valid credentials (usually means a user or an equipment need to provide it to perform other operations in the application). Authorization is to establish a rule to determine whether a credential has the permission to perform a certain operation in the application. Spring security can easily handle the traditional way of authentication by username and password, and also supports various third party authentication methods such as HTTP BASIC authentication headers (an IETF RFC-based standard) [13]. We use Json Web Token as the third party authentication method because it is safer and more efficient. It does not store the authentication token in the database and is simpler to use. The security aspects of the server design are presented in Chapter 4.

1.3 ECG animation

I also participated in part of the front-end development, for example, the ECG animation part, which would send the request to download actual ECG data in Base64 form and convert it to binary values and finally draw it in real time in the front end application. In order to achieve the effect of real-time transmission, we have to carefully consider and deal with the delay caused by the drawing process and the delay caused by data transmission between the server and front-end application. For example, the actual time that it takes to draw one minute of ECG data would be longer than one minute because the front end application also need time to finish the drawing. Therefore, we need to calculate the time difference between them, and draw the corresponding additional ECG data. The details of how the real time ECG animation is implemented is presented in Chapter 6.

Finally, the conclusion and future work is presented in Chapter 7.

Chapter 2 Database

Database is the warehouse that stores the data and there are many kinds of warehouses, such as where one can throw things in casually, or one can store them in a structured order. The most popular type of database can be classified into two major types, relational database and non-relational database [21]. Relational databases include SQL Server, MySQL, SQLite, Oracle, Sybase and DB2, which are a database created on the basis of a relational model, with the help of mathematical concepts and methods such as set algebra to process the data in the database [20]. Non-relational databases include Couchbase, MongoDB, Redis, BigTable and RavenDB, which are databases that do not follow the relational model provided by traditional relational database management systems [22]. Relational database is a more suitable choice for our project since there are many querying and retrieving processes that involve many different tables together. For example, the user often wants information about the nurse and its clinic at the same time. Fortunately, relational databases support JOINs, which combines columns from one or more tables in a relational database. So we do not need to store data in multiple locations, which avoids integrity issues associated with maintaining separate databases [21]. In this project, we use MySQL to build a relational database.

2.1 Architecture

Our database is composed of 9 tables, which are clinics, phones, devices, appointments, ecg test, ecg raw data, nurses, patient information and verification code.

- Clinic table stores information about the clinic including address, phone number, clinic name, email, etc.

- Phone table stores the information of every phone that clinic distributes to every patient, including phone mac address, which is used to identify the phone, and clinic id, which is used to identify which clinic the phone belongs to.
- Device table stores information of the sensor that clinic distributes to every patient, including device mac address, which is to identify the sensor, phone id and clinic id, which indicates the relationship between sensors and phones, and sensor and clinics. Moreover, the table also stores the location of the sensor device, and the usage status of the sensor device.
- The appointment table stores information of appointments by storing the nurse id, patient id, clinic id, device id, and ecg test id. The appointment table also stores the start time, end time, device pick up time and device return time of the appointment.
- The ecg test table stores information needed during the real ECG test period, including the actual start time, actual end time, scheduled end time (usually is setted to be the 24 hours after started recording), and the status of the test (hooking up, recording or termination).
- The ECG raw data table stores the actual ECG data and its information of every ECG test, including the received time, start time and end time of the actual ECG data. It also stores the ecg test id, which is to indicate which ecg test it belongs to. Moreover, it stores the phone status of every piece of ecg raw data, which is used to represent the network connections of phone application, including bluetooth connection between phone application and sensor device, and the internet connection between phone and server.
- Nurses are the major users of the application, the nurse table stores their names, clinic id, email address, which is used as the username when they log in, and password.

- Patient information table stores the basic information of every patient, including their name, birthday, phn, address, clinic Id.
- The Verification code table stores the information when new users try to register their email, including the expired time, email address, and the actual verification code.

2.2 Table Relationship

In order to store all those information safe and efficiently, we have to design the table structure according to the Third Normal Form (3NF), which is the regular form used by database normalization, which requires that all non-primary key attributes are only relevant to candidate keys, which means that non-primary key attributes should be independent and unrelated. It can reduce the duplication of data, avoid data anomalies, ensure referential integrity, and simplify data management.

Firstly, There is a primary key for every table. For example, in the table of clinics, there is a primary key called clinic_id (PK). The rest of the attributes are candidate attributes which describe this clinic. We set it to non-nullable (NN) and auto-increment (AI) since it is automatically generated when each entity of data is created. After we set the primary key to auto-increment, it becomes unique and we can use it to identify every entity (clinics) in the other tables. In addition to that, we add a boolean attribute called deleted to maintain the database better, because everytime when a user making request to delete an entity of data, we first set the deleted attribute to true before we decide whether to delete it permanently or restore it later.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
clinic_id	BIGINT(10)	☑	☑	☐	☐	☐	☐	☑	☐	
clinic_name	VARCHAR(25)	☐	☑	☐	☐	☐	☐	☐	☐	
address	VARCHAR(65)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
phone	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
email	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
fax	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
mail_template	VARCHAR(5000)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
deleted	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Fig. 4 Table of clinic

In the table of nurses, the primary key is nurse_id (PK). There is a foreign key in the table that is called clinic_id because every nurse has a clinic it belongs to. This foreign key represents the relationship between this table and the table of clinics. According to the third normal form, we should not write any information about the clinic in this table to avoid duplication of data. Every nurse must belong to a clinic, therefore the clinic_id should also be set to non-nullable (NN). With the primary key and foreign key, we can easily combine two tables together to form a more informative table with both nurse information and its clinic information.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
nurse_id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
nurse_last_name	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
nurse_mid_name	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
nurse_first_name	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
nurse_phone_number	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
password	VARCHAR(256)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
nurse_email	VARCHAR(51)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
deleted	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
clinic_id	BIGINT(10)	☐	☑	☐	☐	☐	☐	☐	☐	

Fig. 5 Table of nurse

Chapter 3 Backend Application

3.1 Introduction

Spring Boot is a Java-based framework used to create a micro service, which is easy to deploy and minimize the configuration. It provides comprehensive and extensive basic support for the development of Java applications, which helps developers solve basic problems in development, so that developers can focus on the application development. The Spring Boot framework itself is carefully built, which allows developers to integrate the Spring framework in the development environment without worrying about wiring and configuring the basic structure of the server application. There are three main layers: entity class layer, data access layer and business logic layer. Spring Boot automatically configures your application based on the dependencies you have added to the project by using @EnableAutoConfiguration annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, Spring Boot auto-configures an in-memory database, which is created/initialized when the application is started; and destroyed when the application is closed. Moreover, Spring Boot application scans all the beans and packages declarations when the application initializes [14].

3.2 Entity Class Layer

The first part of the server is entity class. Spring uses annotation @Entity to mark all the entity classes, which convert the data in raw data format of a database to Java objects of the server. Then the server can access and modify data of the database through the corresponding Java objects more conveniently. Figs. 6 shows the code of the entity class of the clinic, and Fig. 7 shows the table structure of clinic in the database. We use annotation

@Table to declare that this class maps to the table in the database, through which we can specify the name of the table (name), catalog (catalog) for the entity. The given input of name and catalog can help us to locate the table called clinics and belongs to a catalog called ecg. And in the Clinics class, we use annotation @Id to identify the primary key which is the clinic_id. @Generated annotation specifies the primary key generation strategy. Since we set the primary key to auto-increment in the database, we set the generation strategy to IDENTITY, which allows the database to automatically assign the next available value for the id of the object when its row is inserted. Then we use annotation @Column to declare the mapping relationship between the variables of the entity class and the attributes in the database. For example, we map the clinic_id attribute to the Integer variable clinicId. It's worth mentioning that, according to the Spring variables naming regulation, if there is an underline in the name of the attribute in the database, then in the name of the variable of the entity class, the first letter after the underline has to be capital letter (like clinic_id map to clinicId). This is because we will use the variable name to build the MySQL query function later in the repository (Data access layer).

```java
@Entity
@Table(name = "clinics", catalog = "ecg")
public class Clinics {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "clinic_id", nullable = false)
    private Integer clinicId;
    @Column(name = "address", length = 65)
    private String address;
    @Column(name = "phone", length = 15)
    private String phone;
    @Column(name = "email", length = 15)
    private String email;
    @Column(name = "fax", length = 15)
    private String fax;
    @Column(name = "clinic_name", nullable = false, length = 25)
    private String clinicName;
    @Column(name = "deleted")
    private Boolean deleted;
    @Column(name = "mail_template")
    private String mailTemplate;
```

Fig. 6 Entity class of clinics

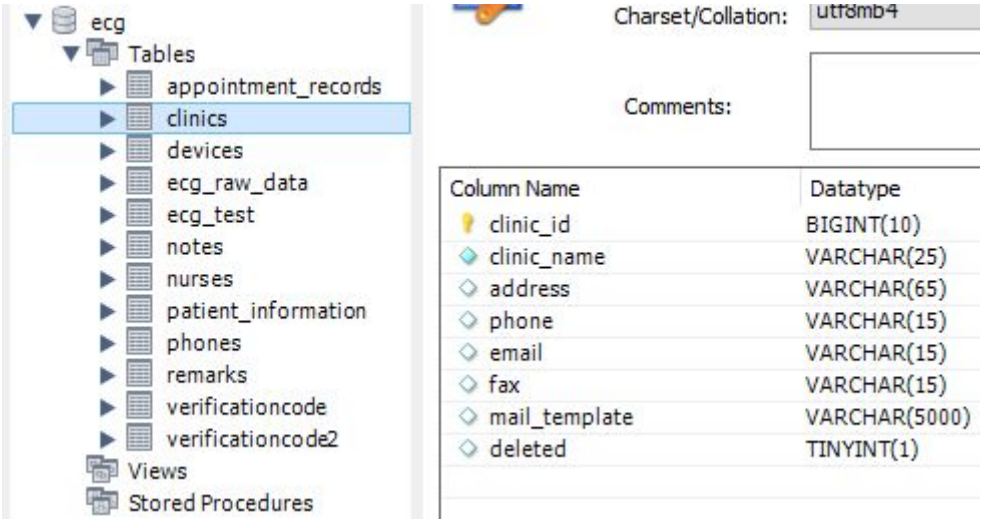

Fig. 7 Clinics table in the ecg schema

3.3 Data Access Layer

In the past, developers spent a huge amount of time on writing redundant code for the DAL (data access layer). For a simple query to search an entity of data by its id, the programmer has to write at least three lines of code. Not to mention we often have to write much more complex queries. Besides, developers have to spend more time on safety concerns like preventing SQL injection. However, the Spring Boot gives us a safer and more efficient framework, Spring data JPA, to help us develop the data access layer. We can build our data access layer by extending the data access layer framework provided by Spring Data JPA. The JPA Repository framework provides many methods that are commonly used. For example, save () can update or create a new attribute into the table; count () can output the total number of entities in the table; findAll () can output all entities; getOne (ID id) can return the entity with a specific primary key. It also provides a more complex method for developers to customize. In the Fig. 8, we can see that there is a method called findByClinicName (). This method is a customized method provided by JPA Repository which takes one string as its input and returns one Clinics object. When the framework parses this method, it first removes findBy, and then parses the remaining attributes. The detailed rules are as follows: First, it will determine whether the ClinicName is a variable of Clinics object. Then, from right to left, intercept the string starting with the first capital letter (here is Name), and then check if the remaining string is an attribute of Clinics. If it is, it means to query according to the attribute “Name”. Therefore, to simplify the query method, we will avoid the case that one variable’s name is contained in other variables. When querying, it is usually necessary to query based on multiple attributes at the same time, and the query conditions are also in various formats (greater than a certain value, in a certain range, etc.), Spring Data JPA

provides some keywords for expressing conditional query. For example, many statements in MySQL can be used the same way in Spring Data JPA like “And”, “Or”, “Between”, “Like”, etc. The Spring Data also provides methods like “Before” and “After” to query the time type variables like Date or Calendar (a data type to store time) in Java. In the Fig. 9, we can see that the method takes two Calendar times as input (startTime and endTime) and returns a List of Appointment object which start time is after the given input startTime and end time is before the given input endTime. Using this method can give us a simple time filter query. Finally, the Spring Data JPA provides a chance for developers to write their own customized query by using @Query annotation. The method will take the string after “ =: ” in the customize query as the parameter of the input of the method. In the Fig. 8, we can write our customized query after the @Query annotation, and take the template and name as the input parameters which are specified in the @Param annotation. In addition to that, @Modifying annotation has to be added when the query is implementing delete or update operation.

```
public interface ClinicRepository extends JpaRepository<Clinics, Integer> {
    Clinics findByClinicName(String cName);

    Clinics findByClinicId(Integer clinicId);

    @Modifying(clearAutomatically = true)
    @Query("update Clinics clinic set clinic.mailTemplate =:template where clinic.clinicName =:name")
    void updateMailTemplate(@Param("template") String template, @Param("name") String clinicName);
}
```

Fig. 8 Repository of clinics (clincis?)

```
List<AppointmentRecord> findByAppointmentStartTimeAfterAndAndAppointmentEndTimeBefore(
        Calendar startTime, Calendar endTime);
```

Fig. 9 Method of repository of appointment

3.4 Business Logic Layer

The business logic layer is implemented by Controller, which is called resource in the Spring Data JPA framework. It contains logic about how to respond to the requests of different APIs. It can perfectly match the RESTful API designing rules. The annotation @RestController is a combination of annotation @ResponseBody and @Controller. It can mark the class with the annotations as a controller class and also import the ResponseBody package, which then Spring would auto configure as the business logic layer of the application. The ResponseBody package can return the output of every http request as a JSON (JavaScript Object Notation) String of an object, which can be easily handled by the front-end. Then the annotation @RequestMapping describes the basic url for all the methods in this class. For example, in Fig. 10 below, the basic url for all the methods in the ClinicResource would be the "domain name (localhost:8080 if it's running locally)/v1/clinic name/". Then there are different urls mapped to different methods. For example, the url for the getMailTemplate () method would be the Get request of the "localhost:8080/v1/clinicName/ mail-template". And the url for the createMailTemplate () method would be the Patch request of the "localhost:8080/v1/clinicName/mail-template".

```
@RestController
@RequestMapping("/v1/{clinic}")
public class ClinicResource {
    @Autowired
    ClinicRepository clinicRepo;
    @Autowired
    private JavaMailSender mailSender;

    private ErrorInfo errorInfo;
    private HttpStatus http;

    @GetMapping("/mail-template")
    public ResponseEntity<RestModel> getMailTemplate(@PathVariable("clinic") String clinicName) {
```

```
@PatchMapping("mail-template")
@Transactional
public ResponseEntity<RestModel> createMailTemplate(
        @PathVariable("clinic") String clinicName, @RequestBody Clinics input) {
```

Fig. 10 Resource of clinic

Moreover, the Spring data JPA has three ways to take input for every request, they are annotation @PathVariable, @RequestBody, and @RequestParam. @PathVariable is pretty straightforward. It takes part of the url as a parameter of input. For example, in the Fig. 11, we can see the url for this method is "/ecg-test/{ecg-test-id}/start-record/{device-id}".

The annotation @PathVariable would take the value of {ecg-test-id} and {device-id} as the parameters of input. If the url is "/ecg-test/1/start-record/1", it will assign the value of ecgTestId and deviceId to 1. @RequestBody annotation is commonly used to handle content encoded by content/type that is application/json type. It will convert the JSON string input to an Java object and assign the value in the JSON string to the corresponding attribute in the Java object. For example, if the input of content is "{"ecgTestId" = 1}", it will be converted to an EcgTest object with ecgTestId value equal to 1.

```
@PutMapping("/ecg-test/{ecg-test-id}/start-record/{device-id}")
public ResponseEntity<RestModel> startRecording(
        @PathVariable("clinic") String clinicName, @PathVariable("ecg-test-id") int ecgTestId,
        @PathVariable("device-id") int deviceId, @RequestBody EcgTest ecgTest) {
```

Fig. 11 StartRecording method

The third way of input that is commonly used is @RequestParam. This annotation would accept the input from the parameter in the url. For example, in Fig. 12, the getEcgTestList () method would require 2 parameters in the url, which are period-start-time and

period-end-time. Therefore, the url of this method must contain parameters period-start-time and period-end-time in the url. For example, the url could be "localhost:8080/v1/test/patient/ecg-test?period-start-time=2020-09-27T00:00:00-0000&period-end-time=2020-10-27T00:00:00-0000".

```
@GetMapping("/ecg-test")
public ResponseEntity<RestModel> getEcgTestList(
        @PathVariable("clinic") String clinicName, @RequestParam(value = "period-start-time") String start,
        @RequestParam(value="period-end-time") String end) {
```

Fig. 12 GetEcgTestList method

Chapter 4 Security

4.1 Introduction

Patient's information is always sensitive data that needs to be carefully stored and protected. Since our project is designed for a medical device, the server has to be secured by following the standards and regulatory practices mandated by the government. The international standard IEC 62304 – medical device software – software life cycle processes is a standard which specifies life cycle requirements for the development of medical software and software within medical devices [4]. By following this standard, we need a software and system security requirement documentation to decompose system/product requirements into software requirements within the context of the system architecture design (identifying the system functions that are to be implemented in software). We also need risk management documentation to control the risks, for example, how to prevent various types of malicious cyber attack.

4.2 Software and System Requirements

IEC 62304 standard mandates developers to analyze the security requirements of the software and system requirements. Software and system requirements can be divided into the following parts:

- Physical characteristics: choose the suitable coding languages and platform
- Computing environments : choose the hardware memory size, time zone and operating system.
- Inputs and outputs : determine the inputs and outputs of software system
- Security requirements: determine the security feature of the software.

For the physical characteristics of our project, we use Java language to develop the server since it is a most commonly used language in the industry and the framework supports Java. This helps us to speed up the development process. Moreover, Spring Security is a framework supported by Java language, which helps us construct the basic architecture of security features.

For the computing environments, Linux operating systems would be an ideal choice in the industry for deploying the server, because Linux and Unix-based operating systems have less known and exploitable security flaws in the information security field. Compared to the Windows operating system, hackers have less interest in attacking the machine with Linux operating systems because only less than 12 percent of computer users choose to use Linux. Therefore, hackers would spend more time on finding bugs of the windows application system rather than the linux operating system [27]. Since our project is expected to provide service to the clinics and patients located at various time zones, therefore, the server is expected to store all the time data into a database after converting them to UTC format (Coordinated Universal Time).

For the inputs and outputs, our project is expected to transfer and serialize data between the server and front end application (windows-end or phone-end) efficiently since there is a large number of data transmissions between them. Therefore, JSON will be used as the serializing language for transferring data between the server and front-end application since JSON can be recognized and accepted by most languages and platforms.

```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```


Fig. 13 JSON example [28]

Moreover, HTTPS should be applied during data transmission. This protocol protects data transmission by using a asymmetric public key structure, which uses two different keys to encrypt the transmission between two parties:

- Private key: private key is stored at the web server, which is used to decrypt the information encrypted by the public key.
- Public key: public key can be accessed by everyone who wants to interact with the server safely, and it is used to encrypt data before transmitting to the server.

Finally, we need to analyze the security features needed for our project. Since we are developing a server for a clinic application, the server is expected to handle and store information from different clinics. Therefore, we need to design a safe and efficient authorization and authentication layer to prevent various types of malicious cyber attack. Since we are using Java Spring frameworks, the built-in Spring security framework acts as the basic security layer. In order to handle the network requests from a large number of users

and save the memory of the server, we will use the JSON Web Token to implement the authentication layer, which is detailed in the next section.

4.3 Risk Management

Risk management is a significant step since we are developing a clinical use software for performing remote ECG holter tests. Therefore we must design a safe and efficient security layer to protect the sensitive information. For authentication, we use the Json Web Token (JWT) to replace the original Java Spring security authentication layer, because it is more efficient than the original layer, which authenticates the network request by cookies. By using Json web token, we can prevent duplicate submissions and Cross Site Request Forgery, because the server will only authorize the request with a token, rather than just giving the authorization to the user's browser. All tokens are only valid for a limited time. If the authorization were given to the user's browser, a malicious hacker could induce the user to click the malicious URL/link to hack the sensitive information. Fig. 14 gives a clear example, where the attacker embeds the malicious request into a hyperlink and sends it to the victim. Then if the victim clicks the hyperlink, it would lead the victim to send the malicious request with the victim's browser, which contains cookies that are used to authenticate the users. Then the attacker can use the cookie to fake identity and get what they want, like fund

transfer or get the sensitive information.

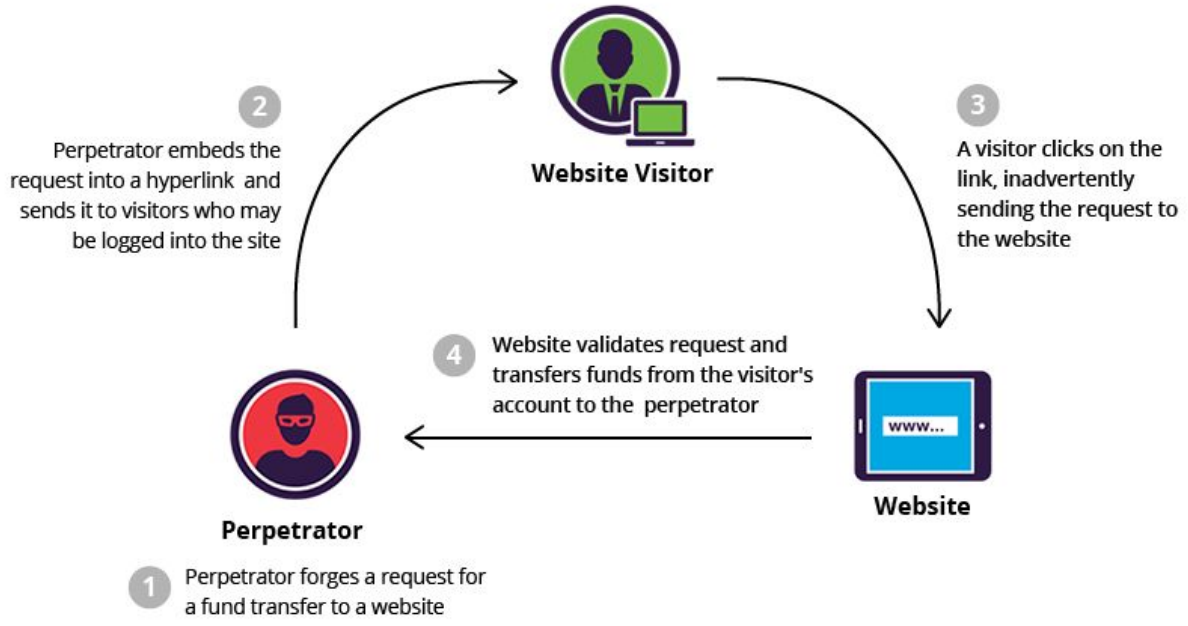

Fig. 14 CSRF attack [1]

Json web token can also prevent the XSS attack. Fig. 15 below gives an example of how hackers launch an XSS attack. Hackers usually inject the website with a malicious script to induce users to visit their website. Since session cookies are usually stored in the browser, the malicious script can steal the session cookie once the user visits the website with the malicious script. By using a token instead of a cookie, the token is stateless, which means that the browser does not store the token (it stores in local storage rather than session storage). Then it can prevent the hacker using XSS attack to hijack the cookie, which causes the sensitive information disclosure. However, a good hacker could write a script to steal the JWT that is stored in the local storage. Therefore we need to add another layer to prevent the XSS in the future.

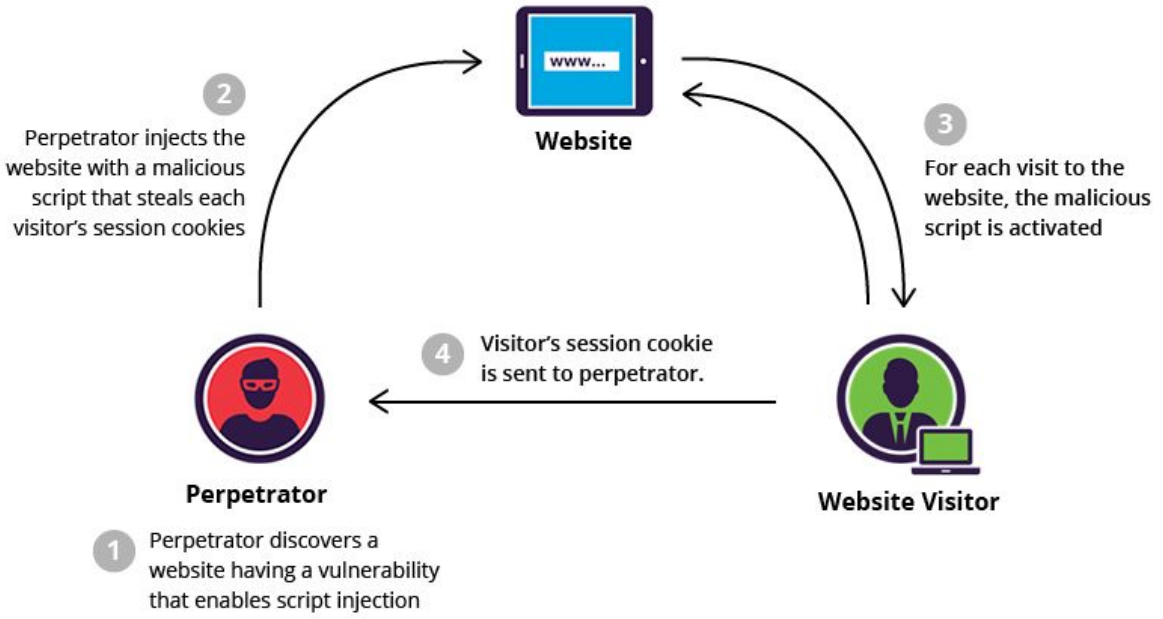

Fig. 15 XSS attack [2]

Authorization also plays an important role when we implement the security features. The most common authorization attack is URL modification, where a hacker uses the token of clinic A to get the sensitive data of another clinic by modifying the url. Therefore, we add a layer in the authorization filter to verify the URL, to make sure the data that the request is accessing belongs to the clinic of the URL. In Fig. 16, the if statement will determine whether the clinic Id of the user is equal to the clinic Id of the clinic Name in the url.

```
if (clinicRepo.findByClinicName(clinic) == null || userRepository.findByNurseEmail(user.getUsername()).getClinicId()
        != clinicRepo.findByClinicName(clinic).getClinicId()) {
    result.setMessage("Login failed because of this user is not belong to your clinic");
    String responseStr = this.gson.toJson(result);
```

Fig. 16 Code for authorization

SQL injection is another common method to hack the sensitive data. The attacker can create input content that acts as malicious payload and is the key part of the attack. After the

attacker sends this content, malicious SQL commands are executed in the database [3]. The common method to prevent this attack is input validation and parameterized queries including prepared statements. However, since our project is a lightweight application, we do not have many complex interactions. Therefore, most of the sql queries can be implemented by the Spring data JPA framework, which is more safe and efficient. We only need to do input validation on several native queries.

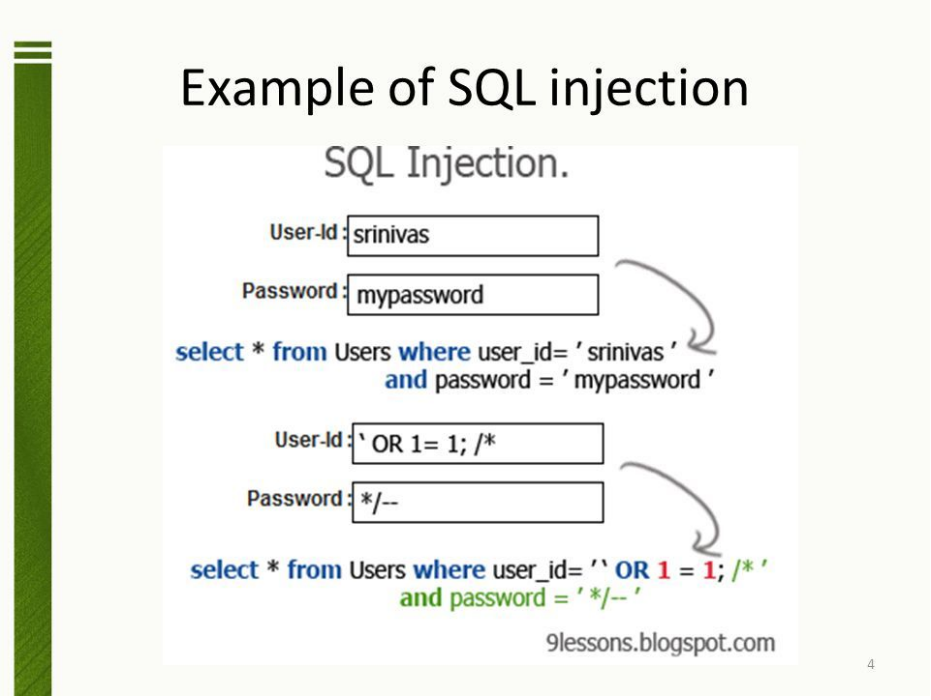

Fig. 17 SQL injection example [24]

The next issue would be interface injection. Hackers usually would transfer the task of creating the object to someone else and directly use the dependency. If developer use the same API (or URI) for two different tasks, hacker could easily use this API to fake their request. For example, Hacker would try to create a new admin account by using the update account interface. Therefore, we should apply RESTful API designs,which are used to divide all interfaces of different functions. For example, the interfaces for update accounts will only

accept limited inputs of the request required to update an account, and use it only to change the existing account in the database.

Last but not least, some coding logic bugs may cause serious issues. For example, a hacker can easily know the language and platform that server used by analyzing the exceptions that returned. Therefore, instead of returning the exception directly, we return the custom error information, which only returns the limited information that is not sensitive, for every network request.

4.4 Spring Security

4.4.1 Basic Architecture

The implementation of all our security features are based on the Spring security features. The basic architecture of the spring security authentication is given by the flowchart. We will pass all Http requests to the authenticationFilter first. Then it will generate a UsernamePaaswordAuthenticationToken and pass it to the AuthenticationProvider (which is the implementation of the AuthenticationManager). The AuthenticationProvider will ask UserDetailsService, which is a class to interact with the database, to query the real password of the user in the database using the username in the UsernamePaaswordAuthenticationToken. It will also generate a UsernamePaaswordAuthenticationToken after querying. Finally, the Authentication Provider will compare two UsernamePasswordAuthenticationToken and send the result back to the AuthenticationFilter to generate the SecurityContext with the authentication result.

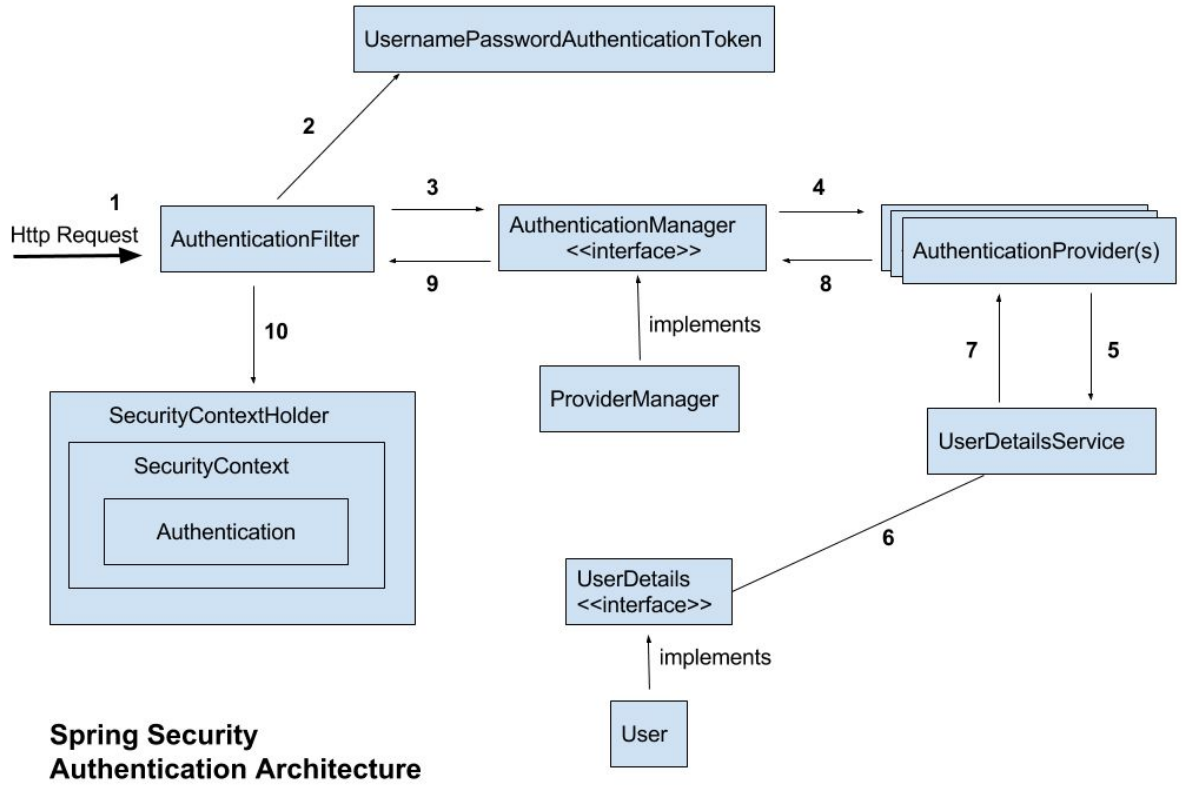

Fig. 18 Spring security authentication [24]

4.4.2 Json Web Token

4.4.2.1Why Use Json Web Token

The biggest advantage of using tokens instead of cookies is that the token is statelessness. The backend does not need to keep records of tokens. Each token is independent, contains all the data to check its validity, and conveys user information through declarations. The work on the server side only needs to generate a token after the login is successful, or verify whether the incoming token is valid.

Secondly, cookies can handle single domains and subdomains very well, but encountering cross-domain problems becomes difficult to handle. Our project involves both windows application end and phone application end, and JWT can handle the cross-domain issues very well. Because every time you send a request to the backend, you only need to check the JWT, as long as they are verified and can process the request.

Moreover, Modern APIs not only interact with browsers, but can support browsers as well as native mobile platforms, such as IOS or Android. The native mobile platform is not necessarily compatible with cookies, and there are some restrictions in use. On the other hand, tokens are easier to implement on IOS and Android, without the concept of cookie storage.

4.4.2.2 Introduction to JWT

As we mentioned above, we are using Json Web Token as the third party authentication filter to replace the original way of using cookies to do authentication. JSON Web Token is a JSON object defined in RFC 7519 as a secure way to represent a set of information between two parties. JWT consists of header, payload and signature. Generally speaking, JWT is a string with the following format: *header.payload.signature*. The user first logs in to the authentication server using the login system of the authentication server (for example, username and password,). Then, the authentication server creates a JWT and sends it to the user. When a user makes an API call to an application, the user passes the JWT and API call. In this example, the application server will be able to verify whether the incoming JWT was created by the authentication server or not (the verification process will be explained in more detail later). When a user makes an API call using an attached JWT, the application can use JWT to verify that the API call came from an authenticated user.

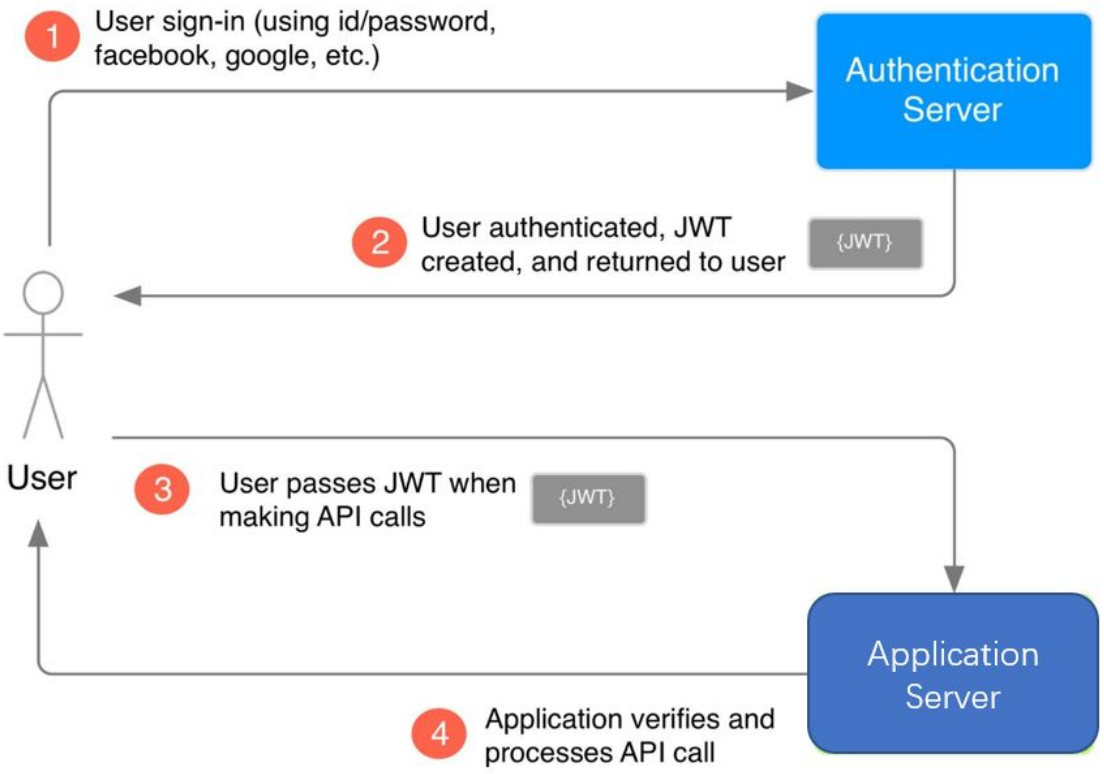

Fig. 19 JWT flowchart [5]

The header of the JWT contains two parts, the value of the “typ” specify the object, and the value of the “alg” specify the algorithm that is used to generate the signature. For example, the HMAC-SHA256 algorithm is used to generate signatures in the example below.

```
{
    "typ": "JWT",
    "alg": "HS256"
}
```

Fig. 20 JWT Header example [5]

Then, the payload is responsible for storing the data in JWT. For example, Issuer (iss), expiration time (exp), subject (sub) and audience (aud). Developers also can store customized fields into the JWT, but it will affect the total size of it. Then a signature will be generated by the algorithm containing the header and payload. The signature is calculated using the following pseudocode:

```
// signature algorithm
data = base64urlEncode( header ) + "." + base64urlEncode( payload )
hashedData = hash( data, secret )
signature = base64urlEncode( hashedData )
```

Fig. 21 Pseudocode to generate signature [5]

It first uses base64url to encode the header and payload and concatenate them with ".". Then use the hash algorithm with a private key (secret) to hash the string of the encoded header and payload. After that , use base64url to encode the hasedData again to get the signature. Finally, concatenate the encoded header, encoded payload and encoded signature together to generate the JWT.

The authorization part of JWT is not complex. In our example, we are using a JWT signed by the HS256 algorithm, where only the server knows the private key. Since the server knows the key, when a user makes an API call with JWT attached to the server, the server can use the header and payload on the JWT to execute the same signature algorithm again to get a new signature. The server can then verify whether the signature obtained from its own hash operation matches the signature on the JWT or not . If the signatures match, the JWT is valid and the API call comes from a trusted source. Otherwise, if the signatures do not match, the received JWT is invalid, which may be an indication of a potential attack on the application.

Therefore, by verifying the JWT, the application adds a layer of trust between itself and the user.

4.4.2.3 Implementation

We customize two classes (JwtAuthenticationFilter, JwtAuthorizationFilter) to replace (extend) the original authentication filters and authorization filters. In the JwtAuthenticationFilter

```
public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
```

Fig. 22 JwtAuthentication class name

(Fig. 23), we use annotation @Override to override the attemptAuthentication method because we are using RESTful API designing rules, therefore, the username and password are transferred to the server in the form of Json String. After reading the value of username and password from the Json String, we store it as a UsernamePasswordAuthenticationToken and pass it to the AuthenticationProvider.

```java
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
                                            HttpServletResponse response) throws AuthenticationException {

    String[] urlParmeter = request.getRequestURI().split( regex: "/");
    clinic = urlParmeter[2];

    try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(request.getInputStream()));
        String str = null;
        StringBuilder sb = new StringBuilder();
        while ((str = reader.readLine()) != null) {
            sb.append(str);
        }
        JSONObject json = JSONObject.parseObject(sb.toString());
        String param = json.toJSONString();
        username = json.getString( key: "username");
        password = json.getString( key: "password");

    } catch (IOException e) {
        e.printStackTrace();
        response.setStatus(400);
        //result.setMessage("Invalid Username or password, Please check again");

    }
    var authenticationToken = new UsernamePasswordAuthenticationToken(username, password);

    return authenticationManager.authenticate(authenticationToken);
}
```

Fig. 23 AttemptAuthentication

Then, in the AuthenticationProvider, we will query the real user information from the database using the userName in the UsernamePasswordAuthenticationToken and generate another UsernamePasswordAuthenticationToken for comparing.

```
@Override
public Authentication authenticate(Authentication auth) throws AuthenticationException {
    String userName = auth.getName();
    String password = auth.getCredentials().toString();

    UserDetails user = userDetailsService.loadUserByUsername(userName);
    if(bCryptPasswordEncoder.matches(password, user.getPassword())){
        password = user.getPassword();
    }else{
        password = null;
    }
    return new UsernamePasswordAuthenticationToken(userName,
            password, AUTHORITIES);
}
```

Fig. 24 Authentication provider authenticate method

After that, if two UsernamePasswordAuthenticationToken match, the JwtAuthenticationFilter will sign a Json Web Token and send it back to the front-end application. It first gets the username (Principal) in the UsernamePasswordAuthentication Token. Then use the private key (JWT_SECRET) and all the information (TOKEN_TYPE, TOKEN_ISSUER, TOKEN_AUDIENCE) to generate the JWT. Note that all the JWT information including private key, issues and hash algorithm are stored in the SecurityConstant class. We will not show the details of the class in order to protect the security of the JWT.

```
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                        FilterChain filterChain, Authentication authentication) throws IOException {
    var user = ((User) authentication.getPrincipal());
    var roles = user.getAuthorities() Collection<GrantedAuthority>
                .stream() Stream<GrantedAuthority>
                .map(GrantedAuthority::getAuthority) Stream<String>
                .collect(Collectors.toList());

        var signingKey = SecurityConstants.JWT_SECRET.getBytes();

        var token = Jwts.builder()
                .signWith(Keys.hmacShaKeyFor(signingKey), SignatureAlgorithm.HS256)
                .setHeaderParam( s: "typ", SecurityConstants.TOKEN_TYPE)
                .setIssuer(SecurityConstants.TOKEN_ISSUER)
                .setAudience(SecurityConstants.TOKEN_AUDIENCE)
                .setSubject(((User) user).getUsername())
                .setExpiration(new Date(System.currentTimeMillis() + 5000000))
                .claim( s: "rol", roles)
                .compact();
        PrintWriter out = response.getWriter();
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        //Add the token to the header of response
        response.addHeader(SecurityConstants.TOKEN_HEADER,  s1: SecurityConstants.TOKEN_PREFIX + token);
        out.flush();
    }
}
```

Fig. 25 SuccessfulAuthentication method in JwtAuthetication Filter

4.5 Storing Encoded Password

Even though we are using HTTPS to transfer the data, it is still not safe at the server end if we store the password in plaintext in the database. Developers used to use MD5 to encode the passwords, however, it is not safe enough for a medical device server because the MD5 value generated by any same file and string is the same. With the computing power nowadays and the rainbow table (is a precomputed table for caching the output of cryptographic hash functions, usually for cracking password hashes), the hacker can easily decode the password encoded by MD5. Therefore, we choose Bcrypt, which is a more powerful encoding algorithm, to encode our passwords. “Bcrypt was designed for password hashing hence it is a slow algorithm. This is good for password hashing as it reduces the number of passwords that

an attacker could hash per second when crafting a dictionary attack. ”[6] Since it is a slow algorithm, hackers would spend much more time on cracking password hashes or using rainbow tables. An article indicates that to encode a password using bcrypt usually takes 0.3 second, and MD5 only needs 1ms. However, A password that takes 40 second to exhaustively decode by MD5 to get the plaintext would need 12 years if it’s encoded by Bcrypt [7].

Moreover, Bcrypt is a one way hash algorithm, which means that even if the attacker knows the encoded password in the database, they will not be able to convert it to plaintext.

The Bcrypt has 4 parameters. SaltRounds, which is an Integer, represents the round of hashing. It would be safer and take more time if it is set to a higher value. MyPassword is the plaintext of the password. Salt is a 128 bits (22 chars) random String. MyHash is a string of 138bits (31 chars) which is generated by the plaintext password and the salt. Finally, the encoded password would be made by the version number of Bcrypt, the round number, the salt, and the Hash.

Fig. 26 Bcrypt password structure[8]

Chapter 5 API and Functions

5.1 Introduction

The ECG monitoring application is a cross-platform software. It involves both windows application and phone application. RESTful API designing rules would be a perfect choice for our project. The RESTful API designing principles are: When naming an interface, you should use nouns, not verbs, because it is a resource that operates through the interface. Secondly, developers should add a version number to the url, which is more intuitive for version iteration management. Moreover, the type of operation of the resource should be represented by the http verb, which means using Get to get resources, using Post to create resources, using Put to update the resource, and using delete to delete resources.

For example, registering a new nurse user would use Post http request with url

ecg.uvic.ca:8080/v1/test/ nurses

In this example, v1 would be the version number, and test/nurses would be the resource name. Finally, the data format requested and returned by the server should use JSON as much as possible, avoiding using XML. Since the RESTful API designing rules are self-explanatory, it can separate the front-end and back-end. The front-end is irrelevant, the back-end is only responsible for data processing, and the front-end representation can be any front-end language (android, ios, html5, C#, etc.). Front-end and back-end developers can pay more attention to their own development, and interaction details between the front-end and back-end can be completed by interface documents without too much mutual understanding.

5.2 Implementation

Fig. 27 shows the basic procedure of one real time ECG monitoring test. Each box represents a resource, which also means it represents a table in the database. For example, Nurse login or register mainly interacts (get entities or create new entities) with the Nurse table in the database.

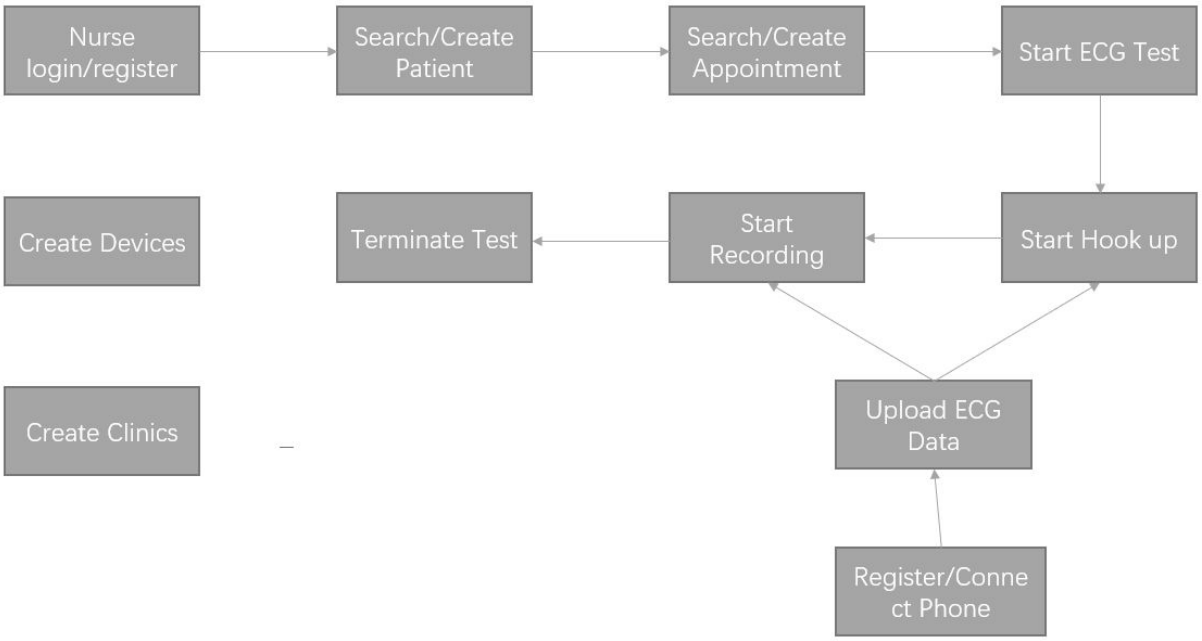

Fig. 27 Flowchart of application process

The developers will create and insert device information and clinic information to the database at the very beginning. And the clinic information including clinic name are stored in the local storage of the windows application. Therefore, whenever the windows application wants to make a request, it can send their clinic name as the authorization parameter in the url. The details of every API are recorded in the ecg rest API design documentation [9]. We will only talk about the API used during the monitoring period, because they are more complex and important.

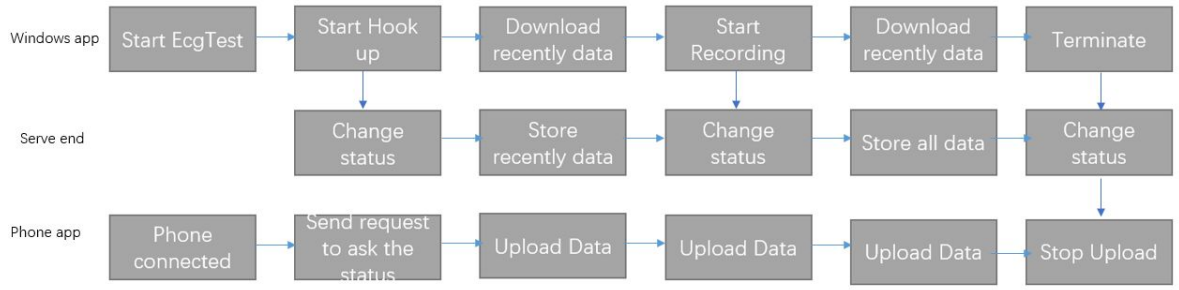

Fig. 28 Monitoring Procedure

5.2.1 EcgTest

EcgTest is an object that stores all the information of one specific Ecg Test of one patient, including actual start time, scheduled end time, actual end time, etc.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ecg_test_id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
actual_start_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
scheduled_end_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
deleted	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
patient_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
nurse_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
phone_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
device_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
clinic_id	BIGINT(10)	☐	☑	☐	☐	☐	☐	☐	☐	
comment	VARCHAR(100)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
hookup_status	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
recording_status	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
actual_end_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
terstatus	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Fig. 29 The EcgTest Table structure

Whenever an appointment is created, the server will create the corresponding EcgTest and store it into the database as well. Then, if the nurse sends the request to start hookup or recording for a specific EcgTest, the hook up status or recording status would be set to true, which means that the EcgTest is ready to receive datas from the phone application. Once a nurse sends a request to start recording, the server would set the actual start time and

calculate the scheduled end time, which is the start time plus the Ecg Test period (24 hours). Moreover, the server would check the calculated scheduled end time and the appointment end time, and take the one which is earlier.

```
//When the nurse start the recording for the first Time, set the startTime and endTime
if(test.getStartTime()==null){
    test.setStartTime(nowTime);
    nowTime.add(Calendar.HOUR, amount: 24);
    AppointmentRecord app = appointmentRecordRepository.findByEcgTestId(test.getEcgTestId());
    if(nowTime.after(app.getAppointmentEndTime())){
        test.setScheduledEndTime(app.getAppointmentEndTime());
    }else{
        test.setScheduledEndTime(nowTime);
    }
}
ecgTestRepo.save(test);
errorInfo = ErrorInfo.OK;
httpcode = HttpStatus.OK;
```

Fig. 30 Part of the code of start recording API

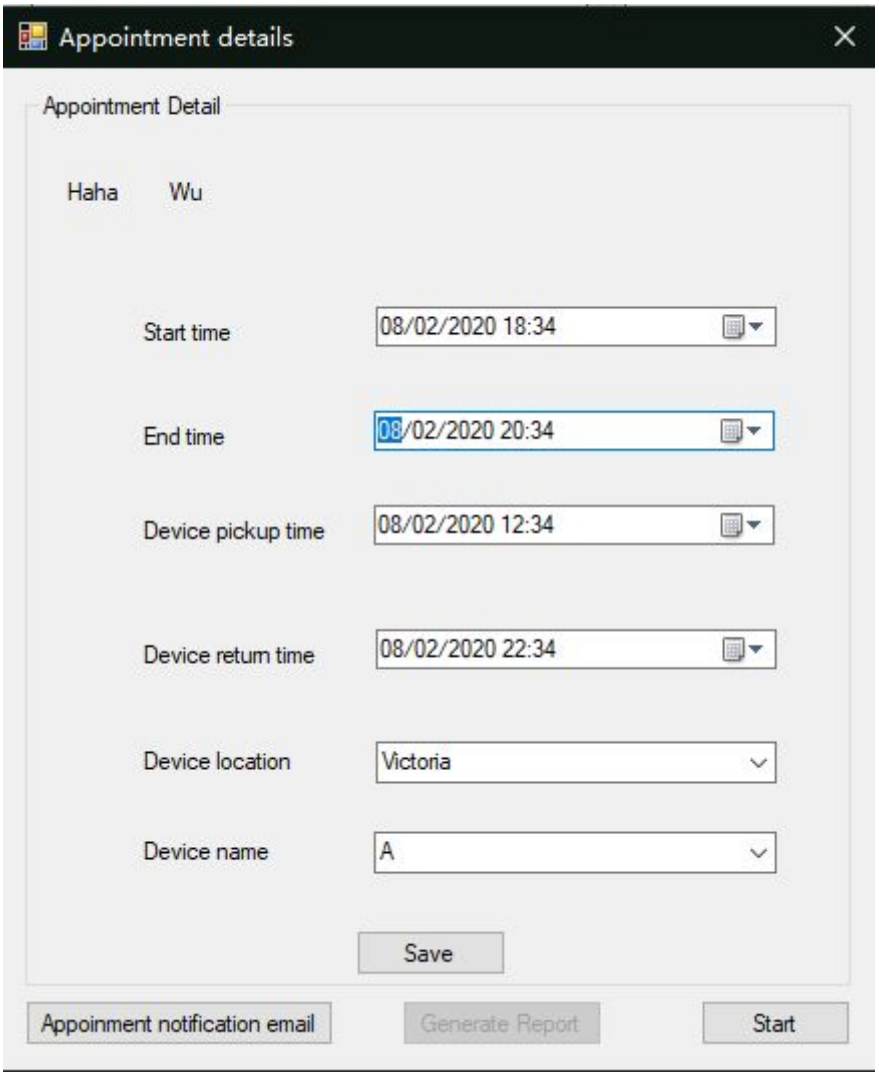

Fig. 31 Screenshot of Appointment Detail

5.2.2 Phone

Phone connection is also an important part of the monitoring. Once the patient connects their phone application and the device, their phone application would receive the mac address of the device, which is already stored and defined in the server. Phone can use the device mac address as the identity to connect with the server. After the phone sends a request to connect with the server, if it is the first time that the phone tries to connect with the server, the server will create an entity of the phone which stores the information of the phone. Then the server creates and returns a verification code with 30 mins validation time to the phone for further

operations. This verification code also records the relationship between the phone and the device.

vcid	content	expired_time	device_id	phone_id
1	vZRsnR	2019-08-03 12:23:23	1	5
2	2CNC4q	2019-08-03 12:24:14	1	6
3	eTf9BI	2019-08-03 12:30:44	1	7

Fig. 32 Table of verification code

Once the phone receives the verification code, it will send a request to the server to ask for the EcgTest status. It is designed to send requests every five seconds at the very beginning. The server will return two parameters to the phone: frequency and containData. Frequency is used to decide the frequency of the next request sent by phone. ContainData is used to decide whether to send requests with the actual ECG data to the server or not. The values of these two parameters depend on the ECG status. If the ECG status is “hooked up”, the server will return a frequency of 5 seconds and containeData equal to true to phone. The phone application will send requests with the Ecg Data (5 seconds period size) and with a frequency of 5 seconds. If the status is “recorded”, the server will return a frequency of one minute and containeData equal to true to phone. The phone application will send a request with ECG Data (one minute period size) in a frequency of one minute, since the nurse does not need real-time monitoring during the recording period. If the status is “terminated”, the server will return a frequency of five minutes and containeData equal to false to phone. Then the phone application will enter the sleeping mode which is sending requests without ECG Data and with a frequency of 5 minutes. The coding details are shown in Fig. 34.

5.2.3 Ecg Raw Data

EcgRawData is the object that stores the real data of the EcgTest, including received time, start time, end time, ecg test id, status flag, phone status and size. The received time records the actual time that the data is received by the server and stored into that database. The status_flag is to determine what type of the data is (hookup data or recording data). These attributes will be used to decide whether to store the data permanently or temporarily. Phone status is used to record the status of connection of phone, including connection between the sensor and phone, and the connection between phone and the server. This is how we determine the phone status: If the bluetooth of the phone is fine, but the phone cannot receive any data, that means the sensor is broken. Then the phone will send a request to change the phone status to 1. If the bluetooth of the phone is disconnected, the phone application will send a request to change the phone status to 2. If the server is not receiving any request from the phone for more than 5 minutes during the hookup period or recording period, the server will change the status to 3. These status can be used for warning the user (nurse) in the front end application.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ecg_raw_data_id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☐	☐	
ecg_test_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
received_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
ecg_raw_data	LONGTEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
deleted	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
start_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
end_time	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
clinic_id	BIGINT(10)	☐	☑	☐	☐	☐	☐	☐	☐	
status_flag	TINYINT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
phone_status	INT(11)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
size	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
		☐	☐	☐	☐	☐	☐	☐	☐	

Fig. 33 Table structure of EcgRawData

When the phone application sends Ecg Data to the server, it first needs to send its verification code to authenticate itself. As mentioned above, the verification code table contains both the phone id and the device id, which means that we can use the device id to find the corresponding Ecg Test that the ECG Raw Data belongs to, because every device will only serve one Ecg Test at a time. Then we can use this to specify the ecg test id of the ECGRawData object. The phone application will also send the start time and end time of an ECGRawData on request. The start time of the ECGRawData is used to identify the duplicate data (with the same start time). After the server receives the ECGRawData, it will calculate the size of the Data (in binary), then convert the data to base64 string and store it in the database.

```
EcgRawData JsnewData = new ObjectMapper().readValue(newData, EcgRawData.class);
JsnewData.setClinicId(clinicRepo.findByClinicName(clinicName).getClinicId());
JsnewData.setEcgTestId(ecgTestRepo.findByDeviceIdAndStartTimeBeforeAndScheduledEndTimeAfterAndClinicId(
        vc2Repo.findByContent(VerificationCode).getDeviceId(), nowTime, nowTime,clinicId).getEcgTestId());
JsnewData.setDeleted(false);

JsnewData.setReceivedTime(nowTime);
```

```
//Find the data is hookup or record data .
if (ecgTestRepo.findByDeviceIdAndStartTimeBeforeAndScheduledEndTimeAfterAndClinicId(
        vc2Repo.findByContent(VerificationCode).getDeviceId(), nowTime, nowTime,clinicId).isHookupStatus()) {
    result.setcontainData(true);
    result.setFrequency(hookupFreq);
    JsnewData.setStatusFlag(false);
    entity.setModel(result);
    EcgRawData ecgRawData = dbFilesStorageService.storeFile(file, JsnewData);
} else if (ecgTestRepo.findByDeviceIdAndStartTimeBeforeAndScheduledEndTimeAfterAndClinicId(
        vc2Repo.findByContent(VerificationCode).getDeviceId(), nowTime, nowTime,clinicId).isRecordingStatus()) {
    result.setcontainData(true);
    result.setFrequency(reocrdFreq);
    JsnewData.setStatusFlag(true);
    entity.setModel(result);
    EcgRawData ecgRawData = dbFilesStorageService.storeFile(file, JsnewData);
```

Fig. 34 Part of the code of upload ECGRawData

Moreover, there will be cases where the phone application only sends part of the data because of the internet issues, and it would resend the data again in a few seconds. Therefore, the server also needs to update the duplicate data (with the same start time).

```
//If the data is already in the database, then updateECGTest the data.
List<EcgRawData> rawdatas = ecgRawRepo.findByStartTime(JsnewData.getStartTime());
if (!rawdatas.isEmpty()) {
    for (EcgRawData rawData : rawdatas) {
        String fileInString = Base64.encodeBase64String(file.getBytes());
        ecgRawRepo.updateRawData(fileInString, rawData.getEcgRawDataId());
    }
    errorInfo = ErrorInfo.DublicateData;
    httpcode = HttpStatus.OK;
    break;
}
```

Fig.35 Part of the code of updating duplicate data

5.2.4 Timer

Another important function of the server is that it needs to check itself regularly for some functions, such as delete expired hookup data, check the connections between server and phone applications and stop the ECG tests when it is the scheduled end time. Good thing is that Java Spring also provides very convenient annotations @EnableScheduling to help us to configure the timer. Developers only need to write the methods that need to be executed and add annotation @Scheduled in the beginning. Then use cron expression to state the frequency of the method that needs to be executed with.

The Cron expression is a string. The string is separated by 5 or 6 spaces and is divided into 6 or 7 fields. Each field represents a meaning. Cron has the following two syntax formats: *Seconds Minutes Hours DayofMonth Month DayofWeek Year* Or *Seconds Minutes Hours*

DayofMonth Month DayofWeek. For example, 0 0 12 ? * WED means executing every Wednesday at 12:00 PM. The characters that can appear in each field are as follows. For the field of Seconds, Minutes and Hours, there can be " , - * /" four symbols and integers between 0-59. Symbol "," is used to separate multiple integers. Symbol "*" is used to represent the entire time period. Symbol "-" is used to specify a time range. Symbol "/" is used to specify a time value increase. n/m means start from n and increase m every time. For the field of DayofMonth, there can be ", - * / L W" six symbols and integers between 1-31. Symbol "L" means the last day of month. Symbol "W" means the week days of the month. For the field of Month, there can be ", - * /" four symbols and integers between 1-12 or strings of JAN-DEC. For the field of DayofWeek, there can be five symbols: ", - * / L " and integers between 1-7 or strings of SUN-SAT [10].

According to the design, the server should check itself for every 2 minutes. Thus, we need to set the cron expression to be " 0 0/2 * * * *". To stop the ECG tests after the scheduled end time, we need to first search for the active Ecg tests using the Ecg test status. We can filter all the ECG tests whose terminated status is false and hookup status or recording status is true to get the active Ecg tests. After that, we can set the terminated status to true, so the phone application will stop sending data to the server. We should also set the actual end time after we stop the Ecg Test.

```
@Scheduled(cron = "0 0/2 * * * *")
public void timeTest(){
    System.out.println("stoping test");
    TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
    Calendar nowTime = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
    //find the ecgTests that is hooking up or recording
    try{
        do{
            List<EcgTest> tests = ecgTestRepository.findByTerstatusAndHookupStatusOrTerstatusAndRecordingStatus(
            if(tests==null||tests.isEmpty()){
                System.out.println("There is no ecgTest that is hooking up or recording right now");
                break;
            }
            for(EcgTest ecgTest : tests){
                if(nowTime.after(ecgTest.getScheduledEndTime())){
                    ecgTest.setTerstatus(true);
                    ecgTest.setRecordingStatus(false);
                    ecgTest.setHookupStatus(false);
                    ecgTest.setActualEndTime(nowTime);
                    ecgTestRepository.save(ecgTest);
                }
            }
```

Fig. 36 Code of timer to stop ECG test

Moreover, we also need to inform the nurse when the connection between the server and phone application is disconnected by email. We will check the data status to confirm the connection between server and phone application. If the ECG test is in hookup status or recording status but the server is not receiving any data for more than five minutes, the server will email the nurse with the information (including patient's first name, mid name, last name, and its phone number) of the patient of this ECG test for the operation exception.

```
List<EcgTest> tests = ecgTestRepository.findByTerstatusAndHookupStatusOrTerstatusAndRecordingStatus( terStatus: false, hookStatus: true,
if(tests==null||tests.isEmpty()){
    System.out.println("There is no ecgTest that is hooking up or recording right now");
    break;
}
for(EcgTest ecgTest : tests) {
    //send email to nurse if internet between server and phone is disconnected
    //find the data received in findDataPeriod(5) mins
    List<EcgRawData> recentData = ecgRawDataRepository.findByReceivedTimeAfterAndEcgTestId(fiveMinsBefore,ecgTest.getEcgTestId());
    if(recentData.isEmpty()){
        if(nurseRepository.findByNurseId(ecgTest.getNurseId())== null){
            System.out.println("Unable to find nurse with nurseId: "+ecgTest.getNurseId());
            break;
        }
        String email = nurseRepository.findByNurseId(ecgTest.getNurseId()).getNurseEmail();
        PatientInfo patient = patientInfoRepository.findByPatientIdAndClinicId(ecgTest.getPatientId(),ecgTest.getClinicId());
        String firstNameOfPatient = patient.getPatientFirstName();
        String midNameOfPatient = patient.getPatientMidName();
        String lastNameOfPatient = patient.getPatientLastName();
        String phoneNumber = patient.getPhoneNumber();
        //send email if no data is received within findDataPeriod(5) mins
        message.setFrom("ArbutusHolsterEcg");
        message.setTo(email);
        message.setSubject("Connection notification");
        message.setText("The connection between server and the phone is disconnected, the patient name is"
                + firstNameOfPatient +" "+midNameOfPatient+" "+ lastNameOfPatient + " The phone Number of this " +
                "patient is " +phoneNumber);
        mailSender.send(message);
```

Fig. 37 Code of timer to email nurse when disconnected

Chapter 6 ECG Animation

6.1 Introduction

In the project, I also contribute to developing the implementation of the ECG animation part of the windows end application, which will download the ECG data from the server and show it in real time in the windows application. The whole windows application is being developed by a teammate, and my work is confined to the real time ECG animation part. We use the Windows Forms (WinForms) framework to develop the windows application, which is a graphical (GUI) class library included as a part of Microsoft .NET Framework or Mono Framework, providing a platform to write rich client applications for desktop, laptop, and tablet PCs [11]. We use the WinForms because WinForms is fit for building a lightweight application and the learning curve is a lot less steep than it is for WPF or UWP. The UI editor in Visual Studio is much better and there is no need to struggle with the often complex syntax of XAML. Developer only needs to drag and drop the controls (like button or textbox) and wire it up to the code [12]. Moreover, considering the version of the computer system of the machine in some clinics would be quite old, WinForms would be a better choice because of its compatibility.

The basic idea of drawing the ECG animation is using a Timer to update (redraw) the whole picture, including the grid, label and ECG data curves in a certain frequency. According to the standard of the ECG, we let the five grids be one second. Each grid contains 50 points, and each point (one value) will be represented by a single pixel in the picture.

Fig. 38 Screenshot of one ECG test

6.2 Implementation

After selecting a specific ECG test and press start button, the windows application would enter the main interface (form) showing in Fig. 39. The user (nurse) can press the hookup (or recording) button to change the test status, which tells the phone application to start sending data to the server.

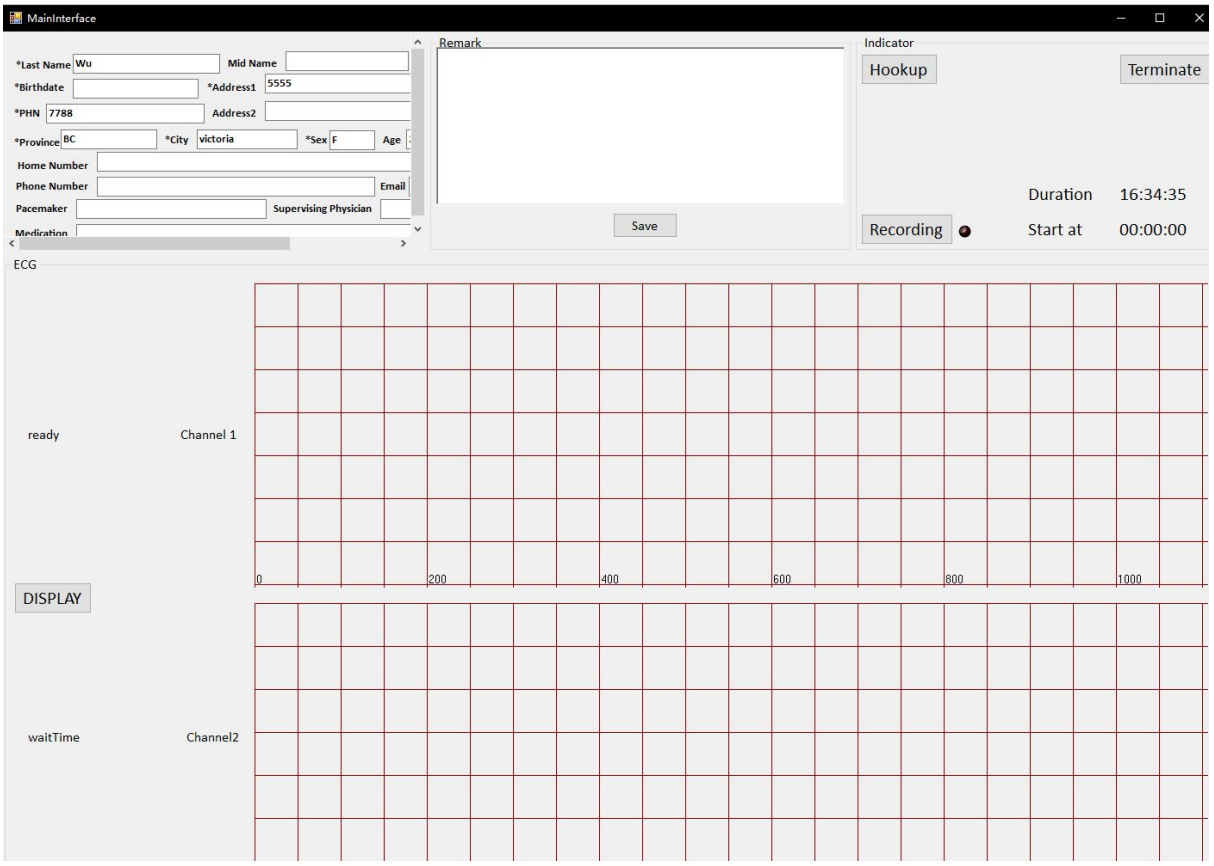

Fig. 39 Screenshot of the main form of ECG animation

The user (nurse) can press the display button to start the real time ECG animation displaying after a certain time waiting (10 seconds if it's a hookup period, and 70 seconds if it's a recording period). Because it has to give the server a certain time to wait for the arrival of data. After clicking the display button, the system will start a timer to update the picture and make the request to download ECG data. The frequency of the timer to make the request is 5 seconds when it is the hookup period or 50 seconds when it is the recording period. This is also the reason why we have to wait 10 seconds or 70 seconds before displaying the ECG data. Because in the worst case, the phone would start sending data after 10 seconds (or 70 seconds when recording) after the user clicks the display button.

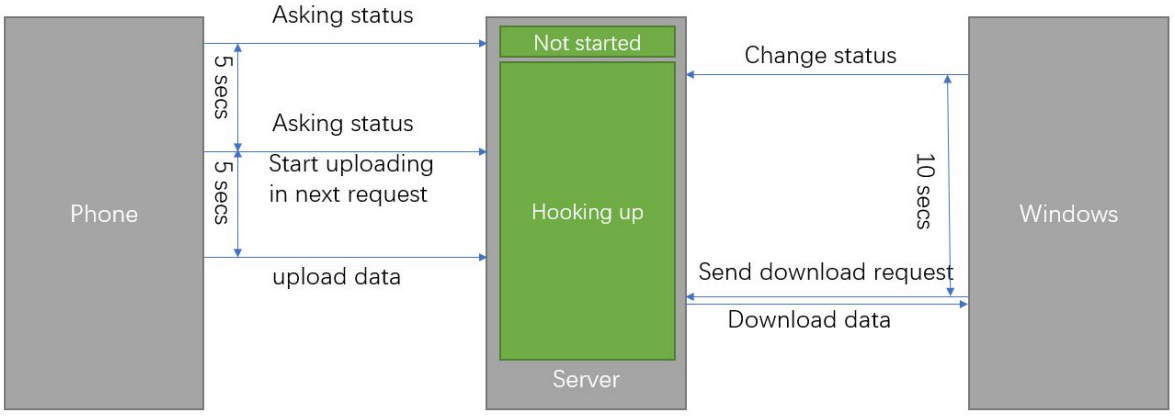

Fig. 40 Flowchart of downloading data when hookup

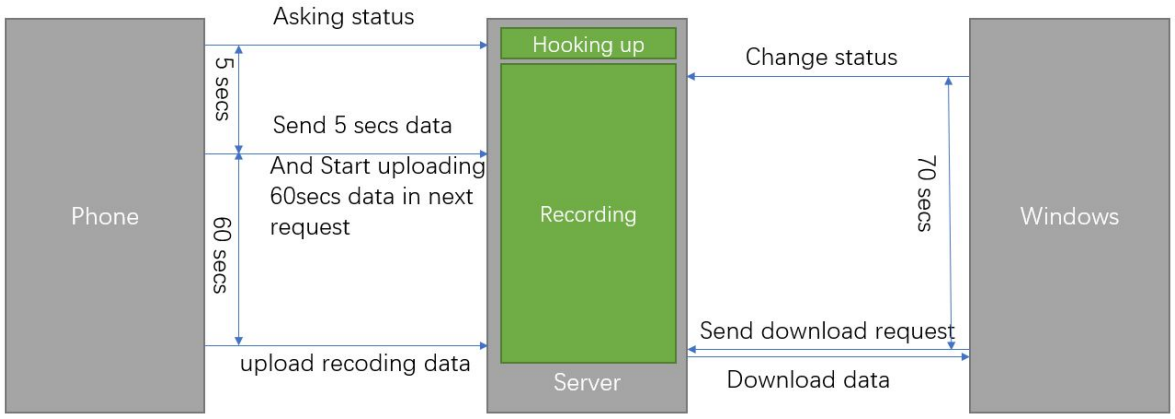

Fig. 41 Flowchart of downloading data when recording

Then we will convert the ECG data from base64 string to an array of binary values. The binary array actually contains values of two channels. Every 5 bytes represent two values in two channels. For every 5 bytes, the first byte is to identify, the second byte is the high byte value of the first channel, and the third byte is the low byte value of the first channel. The fourth byte is the high byte value of the second channel, the fifth byte is the low byte value of the second channel. And we need to concatenate the high byte value and low byte value together to form the real value of the ECG data. To concatenate them together, we first do an

AND operation between the high byte and 0b00111111 because only the last six bits are of real value. Then we shift the real high byte value 8 bits to the left and do an OR operation between the real high byte value and the low byte value to get the final value made by 16 bits.

```
private int Concate(byte highbits, byte lowbits)
{
    byte convertionBits = Convert.ToByte(0b_0011_1111);
    highbits = (byte)(highbits & convertionBits);
    return ((highbits << 8) | lowbits);
}
```

Fig. 42 Code to concatenate high and low byte

After processing the data, we store them in a buffer which is waiting to be drawn later. Since we want to draw the baseline of the ECG data at the middle of the picture (grid) and the values after processing are not exactly zero, we need to make an adjustment to make the baseline of the ECG data located at the middle of the picture (grid). The adjustment is -11.469945 because the ECG data is -11.469945 when the sensor is unattached to the patient (when the ECG data is a straight line). Moreover, in order to make the picture of the ECG data look more clearly, we amplify the values by a factor, which makes the data span a larger range in the Y-axis.

```
public void UpdateValue(float[] newValues)
{
    if (newValues != null)
    {
        foreach (float dataIndouble in newValues)
        {
            if (dataIndouble == 0)
            {
                //in front of every data, there will be some zeros
                values.Add(viewsizeY / 2);
            }
            else
            {
                //data is around 11.469945 when no input, for more user friendly, we set it to zero
                double minusToZero = 11.469945;
                values.Add((dataIndouble - (float)minusToZero) * yAxisFactor + (viewsizeY / 2));
            }
        }
    }
}
```

Fig. 43 Code of adding data to the buffer

Next we need a timer to draw and update the picture. This timer is designed to be executed at a frequency of 25 times per second because the minimum number of FPS that the human eye can accept without jamming is 25 FPS. Since there are 240 values (points) per second, therefore, we have to draw 9.6 values (points) per frame. However, since we are using the basic timer framework of C#, which are running on the same thread, there will be a delay for every timer. The delayed time depends on the executing time of the drawing function. That means we have to draw more than 6 values (points) per frame to make up the delay. Therefore, we have to calculate the time difference between the scheduled executing time and the real executing time of the timer. For example, the timer is expected to draw 6 points in 40 ms, but it takes longer because of drawing the picture. So if it takes 45 ms to finish the drawing, we will calculate the points needed to be drawn in 45 ms, which is 6.75 points. However, since we cannot draw 0.75 points, we will store the extra value of points in a buffer (storedPoints). When this buffer (storedPoints) exceeds one, we will take out one point from the buffer and add one more point in the next drawing timer task.

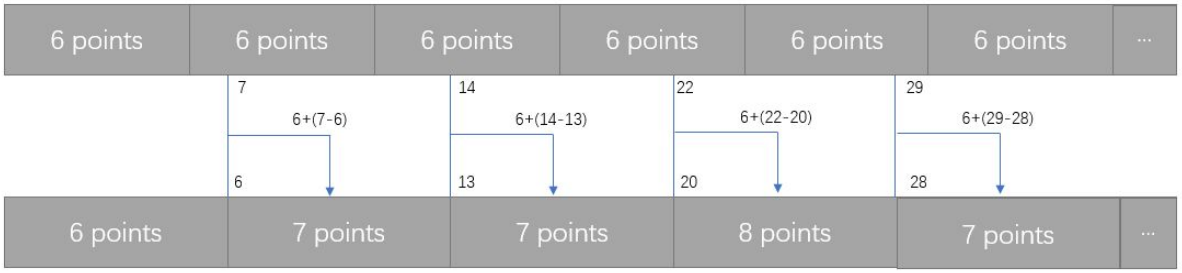

Fig. 44 Points make up process

```
private void DrawTimer_Tick(object sender, EventArgs e)
{
    if (drawTimerCnt == 0)
    {
        stopwatch = Stopwatch.StartNew();
        lastTime = stopwatch.ElapsedMilliseconds;
    }
    //record the actual running time since begining and divide by the expect running time to get the timeDifferenceFactor
    thisTime = stopwatch.ElapsedMilliseconds;
    this.SpendTime = thisTime - lastTime;
    lastTime = thisTime;
    //reset the number of points to be added every tick
    //here we assusme the result is Integer, if not should do floor and calculating
    numberOfpoints = (int)(pointsPerSec * DrawTimer.Interval * 0.001);
    //do points make up if there is time difference
    if (this.SpendTime != 0)
    {
        storedPoints += (pointsPerSec * SpendTime * 0.001) - Math.Floor(pointsPerSec * SpendTime * 0.001);
        numberOfpoints = (int)Math.Floor(pointsPerSec * SpendTime * 0.001);
    }
    if (storedPoints >= 1)
    {
        numberOfpoints++;
        storedPoints--;
    }
    pointsToLine = new float[numberOfpoints];
```

Fig. 45 Code of calculating the points to be drawn

After calculating the points to be drawn in the next timer, we would extract the corresponding number of points from the buffer (values[]) which stores the datas, and storing them into another buffer (points[]) which stores the points to be drawn in the next timer. This buffer is also an array of double type numbers. When we transfer the data from values buffer to points buffer, we first determine whether the points buffer is full or not, because the screen can only show a limit number (pixels) of points, which depends on the size of the screen. If the points buffer is full, we delete the oldest data in the array, and add the newest data at the end of the

array. In the meantime, we also calculate the changed value that the picture shifted to the left by recording how many points (pixels) we have added into the buffer (points[]). We use this changed value to update the position of the grid and the labels. Note that the change value is restored to zero if it is more than the distance between two labels. Finally, the timer calls the paint function to draw or update the picture of ECG data. It first calculates the original position of the grid and labels by dividing the size of the picture by the size of the grid and label (we initialize the size of grid to be 50*50 pixels, and the distance between labels to be 200 pixels). Then draw the grid and label using its original position added by the change value we calculated. After that, the function draws the points in the buffer (points[]).

Chapter 7 Conclusion

In conclusion, we have implemented and tested a remote ECG server application that provides data services to a phone client and a windows client. Major functions of this application include uploading data from the phone application to the server, downloading data from the server to the windows application, showing the ECG data in the windows application in real time and some general functions like create/search patient, create/search appointments. More importantly, we have achieved some basic defense against cyber attack by using the Json Web Token to improve the security and efficiency of authentication between the server and windows application, using BCrypt to encrypt the password before storing them in the database, and using Spring Data JPA to prevent SQL injection. However, there are still many other security issues that need to be fixed, like the authentication between the server and phone application. We currently just use the device mac address to authenticate phone applications when it sends http requests to the server. However, the mac address could be stolen and faked by malicious attackers and they can send fake data to the server. Therefore, we need to design a new system to authenticate the phone application in the future. In addition to that, we just store the Json Web Token in local storage rather than in the browser web storage to prevent the XSS attack. However, there could be a possibility that the token is stolen through a memory leak or a Man-in-the-middle attack, or the user just simply lost the device. Therefore, we should be able to revoke all the token for the user (like log them out), or mitigate it by detecting irregularities in the client requests. For example, if a client makes requests from different IP addresses or sends conflicting user-agents, we might want to revoke the JWT and require re-authorization [29]. In the future, we will adopt a

holistic and systematic approach to carefully examine the needed security measures of the overall system.

Moreover, there are still more features that need to be completed like helping doctors to analyze the ECG data using third party analyze software. Once we have implemented all the features including the front-end application, we will apply a HTTPS certification, to prevent attackers from getting sensitive information during the transmission of data. Sensitive data, like passwords, arrived at the server application side, are encrypted using the Bcrypt algorithm and stored in the database. In the future, we will also design the process of server maintenance, such as viewing the logs collected by the system, which record information about hardware, software, and system problems in the system. It also monitors events that occur in the system. Developers can use it to check the cause of the error or look for traces left by the attacker should security breach occur.

Reference

[1] Threats, A. (2020, July 07). What is CSRF: Cross Site Request Forgery Example: Imperva. Retrieved July 15, 2020, from https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

[2] Threats, A. (2019, December 29). What is XSS: Stored Cross Site Scripting Example: Imperva. Retrieved July 15, 2020, from https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/

[3] What is SQL Injection (SQLi) and How to Prevent It. (2020, April 01). Retrieved July 15, 2020, from https://www.acunetix.com/websitesecurity/sql-injection/

[4] IEC 62304. (2019, October 05). Retrieved July 15, 2020, from https://en.wikipedia.org/wiki/IEC_62304

[5] Cpselvis. (n.d.). Understanding JSON Web Tokens (JWT) in five simple steps. Retrieved July 15, 2020, from https://zhuanlan.zhihu.com/p/46229969

[6] Draveness. (2019, November 21). Why MD5 cannot be used to encrypt password. Retrieved July 15, 2020, from https://draveness.me/whys-the-design-password-with-md5/

[7] Chen, H. (2011, December 21). How to prevent password attack. Retrieved July 15, 2020, from https://coolshell.cn/articles/2078.html

[8] Martin6699. (n.d.). New opinion about Bcrypt encryption. Retrieved July 15, 2020, from https://www.jianshu.com/p/2b131bfc2f10

[9] Liu, Z. (n.d.). Ecg rest API design documentation. Retrieved July 15, 2020, from https://docs.google.com/document/d/16-3CN6fgMRWe2QYqLH8UDMqwppyUldZZbGJWLXUJPeo/edit?usp=sharing

[10] Wiki. (2020, July 08). Cron. Retrieved July 15, 2020, from https://en.wikipedia.org/wiki/Cron

[11] Wiki. (2020, May 09). Windows Forms. Retrieved July 15, 2020, from https://en.wikipedia.org/wiki/Windows_Forms

[12] Peter, M. (2019, September 14). The Death of WinForms Has Been Greatly Exaggerated - SubMain Blog. Retrieved July 15, 2020, from https://blog.submain.com/death-winforms-greatly-exaggerated/

[13] Taylor, L., Winch, R., & Alex, B. (n.d.). Spring Security Reference. Retrieved July 15, 2020, from https://docs.spring.io/spring-security/site/docs/4.0.1.RELEASE/reference/htmlsingle/

[14] Spring Boot - Introduction. (n.d.). Retrieved July 15, 2020, from https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

[15] C. (n.d.). Back-End Web Architecture. Retrieved July 17, 2020, from https://www.codecademy.com/articles/back-end-architecture

[16] Fisher, Sharon (1989). "OS/2 EE to Get 3270 Interface Early". Google Books.

[17] C. (n.d.). Back-End Web Architecture. Retrieved July 17, 2020, from https://www.codecademy.com/articles/back-end-architecture

[18] B. (n.d.). How to choose server. Retrieved July 18, 2020, from http://c.biancheng.net/view/6878.html

[19] Rouse, M. (2020, April 07). What is a RESTful API (REST API) and How Does it Work? Retrieved July 18, 2020, from https://searchapparchitecture.techtarget.com/definition/RESTful-API

[20] Relational database. (2019, December 31). Retrieved July 18, 2020, from https://zh.wikipedia.org/wiki/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93

[21] Zhongyue, S. (n.d.). MySQL database development. Retrieved July 18, 2020, from https://www.jianshu.com/p/28f59950a310

[22] M. (n.d.). What Is A Non Relational Database. Retrieved July 18, 2020, from https://www.mongodb.com/scale/what-is-a-non-relational-database

[23] Homan, J. (2020, January 30). Relational vs non-relational databases. Retrieved July 19, 2020, from

https://www.pluralsight.com/blog/software-development/relational-non-relational-databases

[24] Tennakoon, C., Says:, Y., Says:, F., Says:, N., Says:, T., Says:, R., & Says:, H. (2017, August 30). Spring Security : Authentication Architecture. Retrieved July 20, 2020, from https://springbootdev.com/2017/08/23/spring-security-authentication-architecture/

[25] Spring Security structure. (2019, November 02). Retrieved July 20, 2020, from https://semlinker.com/spring-security-arch/

[26] SQL INJECTION COUNTERMEASURES & - ppt video online download. (n.d.). Retrieved July 22, 2020, from https://slideplayer.com/slide/6836897/

[27] Linux China. (n.d.). Why Linux is safer than windows and OS. Retrieved July 24, 2020, from https://zhuanlan.zhihu.com/p/38316760

[28] How to Read JSON Data and Insert it into a Database. (n.d.). Retrieved July 30, 2020, from https://www.goanywhere.com/managed-file-transfer/more/tutorials/parse-json-data-into-database

[29] My experience with Json Web Token. Retrieved May, 03, 2017, from https://x-team.com/blog/my-experience-with-json-web-tokens