Final Report

GP connect – Spring Boot Web app

Student Name	Student Number	email address
Sebastian Konefal	x21197458	x21197458@student.ncirl.ie

Higher Diploma in Science in Computing

Software Development

Date

08/01/2023

Table of Contents

1. Introduction

1.1 Project Overview

The project offers streamlined registration of new patients to a GP surgery via Web app based on Spring Boot framework. A prospective patient is able to search for surgeries that are currently accepting new patients and submit his/her application after successful log in and inserting personal data required to submit an application. Given the fact that waiting time varies in each surgery, a prospective patient is able to add multiple surgeries to his/her registration application.

The application serves as the first point of contact for patients that are looking to change to register or change a GP surgery.

1.2 Background

The process of registration with a new GP or medical practitioner is quite cumbersome and involves contacting each surgery manually by phone to enquire whether that particular surgery is accepting new patients and if the answer is positive, to attend the surgery in person to fill out a registration form.

The amount of people that will need to register is expected to increase due to the fact that Ireland's Budget for 2023 will increase the eligibility of people for GP cards which enables a patient to visit the survey free of charge. At the end of 2022, the program is being expanded to include all kids aged 6 and 7, and starting on the 1st of April 2023, any person in a household earning less than the median household income of €46,000 will be eligible for a GP card. This will result in approximately additional 430,000 people entitled to the GP card which enables free of charge visits with GP.

Currently, HSE provides a list of GP practitioners that might be accepting new patients https://www2.hse.ie/services/find-a-gp/. However, that information consists only of contact details and no information is provided on whether this particular surgery is, in fact, accepting new patients, In reality, most of the surgeries listed on the HSE website are not accepting new patients.

In addition, HSE provides that a patient will be assigned a GP in the event of having been declined by 3 GP surgeries.

The above results in a lack of transparency and a delay in access to medical services.

1.3 Technologies

HTML	Markup language used to structure and format content on the World Wide Web. It is the standard markup language for creating web pages and web applications.
CSS:	Stylesheet language used for describing the look and formatting of a document written in HTML. It is used to control the presentation of web pages and applications, and can be used to define the layout, colors, fonts, and other visual aspects of a document.
Bootstrap	Front-end framework for building responsive, mobile-first websites web applications. It provides a set of predesigned, customizable UI components and layout templates that can be use to build web interfaces. Bootstrap version 5.0 is the latest version of the framework at the time of writing.

Java	Programming language and computing platform. One of the main advantages of Java is that it is designed to be "write once, run anywhere". This means that Java code can be compiled into a format that can run on any device that has a Java Virtual Machine (JVM) installed, regardless of the underlying hardware and operating system.
Spring Boot	Java-based framework used to build web applications and services and makes it easy to create standalone, production-grade Spring-based applications. Spring Boot provides a number of benefits, including: • It reduces the amount of boilerplate code and configuration required to build an application. • It provides defaults for commonly used libraries and frameworks, so you don't have to spend time configuring them. • It provides built-in support for common application features such as security, database access, and more.
Spring-Boot-Starter	Dependency that can be included in your project to enable a particular feature or group of features. Spring Boot starters are a convenient way to include a set of commonly used dependencies in your project. • Spring-Boot-Starter-Web - includes dependencies for building web applications, including support for web application servers, JSON handling, • Spring-Boot-Starter-Data-JPA - dependencies for working with databases using the Java Persistence API (JPA), including support for popular databases like MySQL and H2 that were used in the project and include Hibernate((a popular implementation of JPA). • Spring-Boot-Starter-Security - Spring Security - a framework that provides customizable authentication and authorization to Java applications. It protects inter alia against fixation, clickjacking, cross site request forgery
Spring Security	Java/Java EE highly customizable and powerful authentication and access-control framework
Spring-Boot-Starter-Test	Dependency for Spring Boot applications that provides a number of libraries for writing tests and includes Junit, Hamcrest, Mockito, Spring Test and Spring Boot Test.
Spring-Boot-Devtools	Library that provides support for fast application development in a Spring Boot application. It includes a number of features that can help you develop and test your application more quickly, such as: • Automatic restart: When you make changes to your application's code, the application is automatically restarted so that the changes take effect. This can save you time by eliminating the need to manually stop and start the application. • Live reload: When you make changes to your application's code, the application is automatically reloaded in the browser, so you can see the changes immediately. This can be especially useful when working on the front-end of your application. Remote debugging: You can attach a debugger to a running application and debug it remotely. This can be useful when you need to debug an issue that only occurs in the live environment

Spring Data JPA	Java specification for managing relational data in Java applications. It helps to reduce the work required to develop data access layers to the minimum.
Spring-boot-starter-validation	Starter dependency for Spring Boot applications that provides support for bean validation, which is the process of verifying that the values of an object's properties are valid. It includes the following libraries: • Hibernate Validator: A library for performing bean validation. • Validation API: The specification for bean validation, which provides the API for performing validation.
Lombok	Java library that can be used to reduce the amount of boilerplate code that is typically required when working with the Java programming language. It provides a set of annotations that you can use to automatically generate common code, such as getters and setters, equals and hashcode methods.
Spring-boot-starter-thymeleaf	Java library that is used to generate HTML, XML, or other markup languages for display in a web browser. The dependency includes the Thymeleaf library and several additional libraries that are commonly used with it, such as Spring Web MVC and Spring Boot Web. By including this dependency in a project, a developer can easily set up a Spring Boot application to use Thymeleaf for rendering the user interface and handle requests and responses using Spring Web MVC.
MySQL database:	Open-source database management system that is widely used on the web. based on the Structured Query Language (SQL), which is a standard language for managing and manipulating data stored in relational databases. In a MySQL database, data is stored in tables. Each table has a set of columns (fields) and rows (records). Tables can be related to one another using keys, allowing you to create complex data structures and perform queries that retrieve data from multiple tables at once.
Hibernate:	Object-relational mapping (ORM) framework for the Java programming language. It is used to map Java objects to relational database tables and vice versa. It provides a number of features for working with data, including support for lazy loading and caching.
Sqlectron	Cross-platform database management tool that allows you to connect to and manage multiple databases using a graphical user interface (GUI). It provides features such as a SQL editor, table data viewer, and database schema viewer, making it a useful tool for developers and database administrators.
Postman	Tool that allows developers to test and work with APIs (Application Programming Interfaces). It provides a convenient interface for sending HTTP requests and viewing the responses.
Amazon RDS from AWS	Manages relational database service that makes it easy to set up, operate, and scale a variety of databases, including MySQL, PostgreSQL and Oracle.

1.4 Structure

In the Introduction section, the project overview, background, technologies, and structure are presented. The project overview provides an overview of the project, including its goals and objectives. The background section discusses the context and motivation for the project, while the technologies section describes the tools and frameworks used in the project. The structure section outlines the major components and their relationships within the system.

The System section begins with a description of the requirements for the project. This is followed by a discussion of the design and architecture of the system, including diagrams and figures to illustrate the various components and their interactions. The Implementation section provides details on the implementation of the system, including code snippets and additional diagrams as needed.

The Testing section discusses the various testing activities that were carried out on the system, including unit testing, integration testing, and system testing.

The GUI section provides screenshots and explanations of the user interface of the system, highlighting the main features and functionality.

The Conclusion section summarizes the main achievements of the project and discusses any limitations or challenges encountered. The Future Scope of the project section discusses potential areas for future development and expansion.

The References section lists the sources used in the report, while the Appendix provides any additional materials, such as code listings or additional diagrams.

2. System

This section of the report covers the details of the system, including the requirements, design and architecture, implementation, testing, GUI layout, and evaluations. The information provided in this section will give a thorough understanding of the system and how it was developed

2.1 Requirements

2.1.1 Functional Requirements

The functional requirements specifies capabilities or features that the system must have in order to meet the needs of its users and should describe the functions that the system must be able to perform in order to meet the stated objectives of the project.

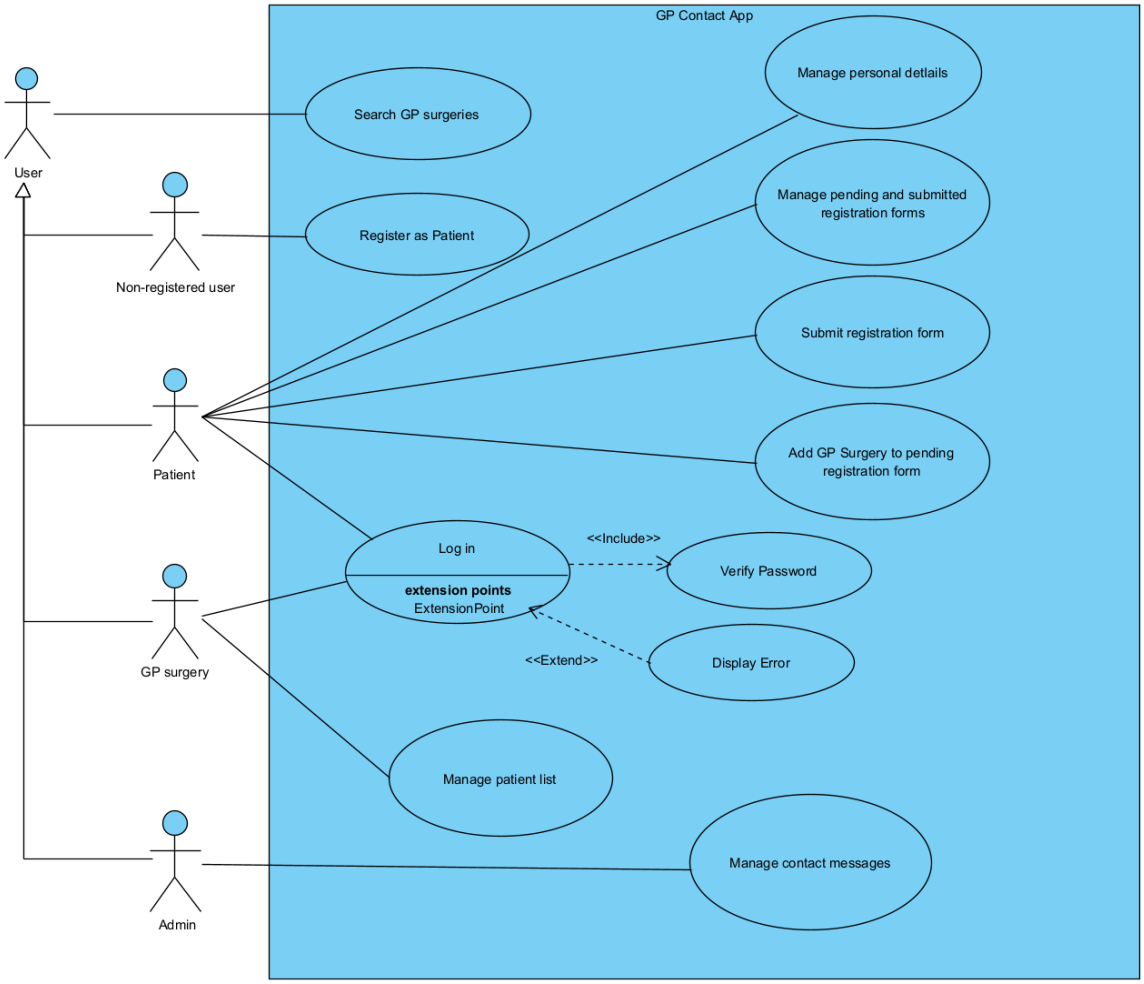
Below is a list of functional requirements that has been achieved by the Project:

1. Registered and non-registered user search GP surgeries and are able to view their details and directions (map).
2. Non-registered user registers to the app.
3. Non-registered user is able to contact administrator by using contact form.
4. Registered user logs in to the app as Patient or GP surgery.
5. Patient adds GP surgeries to pending applications for registration as patient.
6. Patient reviews his/her pending applications and deletes them as needed.
7. Patient progresses his/her pending applications by inserting required personal details and submitting all application at once.

8. Patient reviews his/her submitted applications and deletes them as needed.
9. Patient is able to amend personal details which are also updated to the existing pending and submitted applications.
10. Patient manages the submitted registration forms (views or withdraws).
11. GP surgery/ system manages patient list by viewing sorted by date and deletes accepted patients from the list.
12. Administrator approves GP surgeries accounts.

2.1.1.1 Use Case Diagram

All functional requirements are outlined in the Use Case Diagram below.



Please refer to “Project Requirement Specification” report that includes a thorough description of the functional requirements for a specific use case.

2.1.2 Non-functional Requirements

Performance/Response time requirement - According to research, users lose interest in an application when response times exceed three seconds, and the ideal response time for any application is no longer

than one second. This is because users begin to lose attention to the work at hand when response times exceed one second. Due to that fact, I will aim to keep the response time (for example for login) to no longer than 3 seconds, loading a page should not take longer than 3 seconds.

Availability requirement- The application aims to be available to 100% mark as much as possible however, due to required regular maintenance the availability of the web application may be restricted from time to time. The 100% availability is also hard to be technically feasible as some solutions needs to be considered in the events of components failure such as a „mirror" running in parallel in the background.

Recover requirement - The system should be able to recover as using one of the cloud service will keep a backup of the web application facilitating continuing access to the system functions.

Robustness requirement - The system will aim to function correctly and in the manner that is expected regardless of events in the environment. I will include in my project such aspects as handling unexpected termination and unexpected action.

Security requirement - The system will use Spring Security which is a framework that provides authorization, authentication and protection against common attacks. All passwords in the database must be encrypted using a hashing algorithm.

Reliability requirement - A cloud service was chosen from Amazon Web Services to host the web application database and the system should perform without failure in 99.99%.

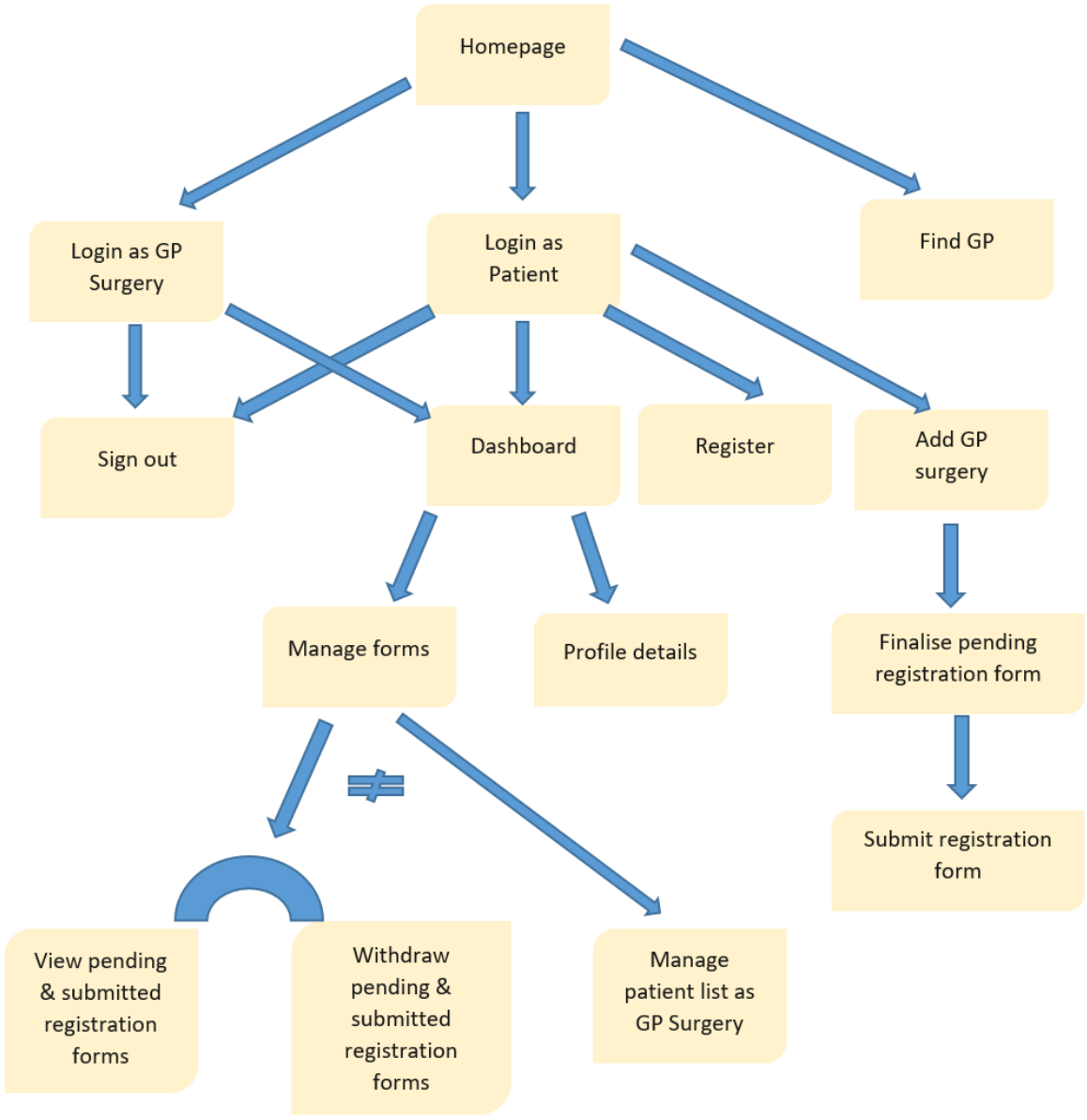
Maintainability requirement - The web application should be easily modified, extended and improved when required for commercial implementation. The web application will be kept to high standards and it will aim to perform a successful repair action within the shortest possible time.

Portability requirement - The web application should be easily accessed and managed from different types of devices that have access to the Internet

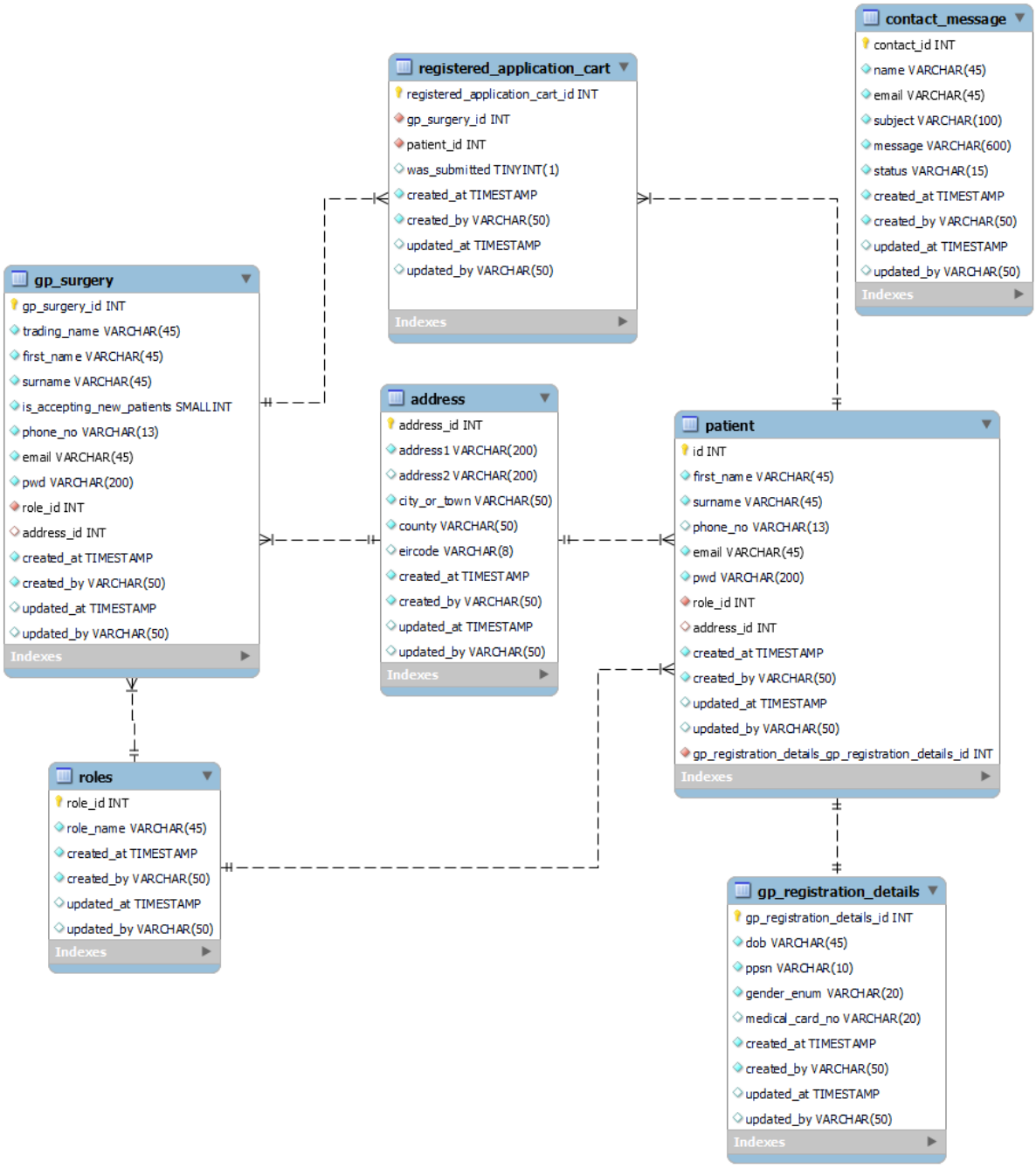
Extendibility requirement - The web application is developed for the given project's requirements however, it's further development for commercial se can be performed in the future.

2.2 Design and Architecture

2.2.1 Updated page flow



2.2.2 Database design



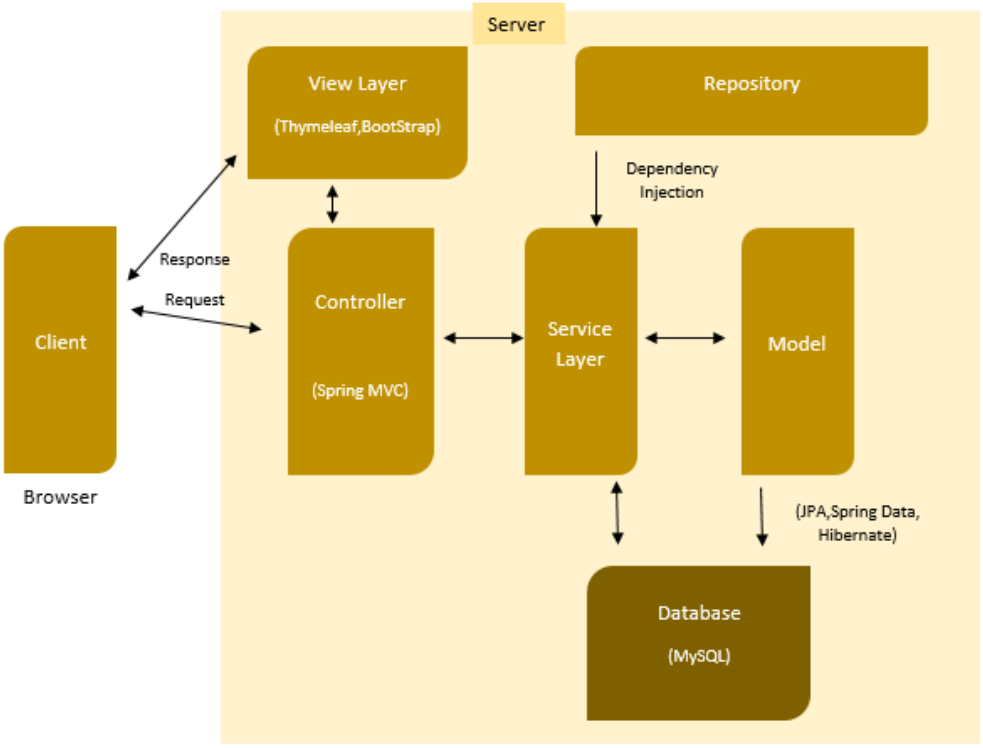
2.2.3 SQL Statements

```
CREATE DATABASE GPConnect;

USE GPConnect;

CREATE TABLE IF NOT EXISTS `contact_message` (
                            `contact_id` int AUTO_INCREMENT  PRIMARY KEY,
                            `name` varchar(45) NOT NULL,
                            `email` varchar(45) NOT NULL,
                            `subject` varchar(100) NOT NULL,
                            `message` varchar(600) NOT NULL,
                            `status` varchar(15) NOT NULL,
                            `created_at` TIMESTAMP NOT NULL,
```

```
                                `created_by` varchar(50) NOT NULL,
                                `updated_at` TIMESTAMP DEFAULT NULL,
                                `updated_by` varchar(50) DEFAULT NULL
  );

CREATE TABLE IF NOT EXISTS `roles` (
                               `role_id` int NOT NULL AUTO_INCREMENT,
                               `role_name` varchar(45) NOT NULL,
                               `created_at` TIMESTAMP NOT NULL,
                               `created_by` varchar(50) NOT NULL,
                               `updated_at` TIMESTAMP DEFAULT NULL,
                               `updated_by` varchar(50) DEFAULT NULL,
                               PRIMARY KEY (`role_id`)
  );

CREATE TABLE IF NOT EXISTS `address` (
                                `address_id` int NOT NULL AUTO_INCREMENT,
                                `address1` varchar(200) NOT NULL,
                                `address2` varchar(200) DEFAULT NULL,
                                `cityOrTown` varchar(50) NOT NULL,
                                `county` varchar(50) NOT NULL,
                                `eircode` varchar(8) NOT NULL,
                                `created_at` TIMESTAMP NOT NULL,
                                `created_by` varchar(50) NOT NULL,
                                `updated_at` TIMESTAMP DEFAULT NULL,
                                `updated_by` varchar(50) DEFAULT NULL,
                               PRIMARY KEY (`address_id`)
  );



CREATE TABLE `patient` (
                    `id` int NOT NULL AUTO_INCREMENT,
                    `first_name` varchar(45) NOT NULL,
                    `surname` varchar(45) NOT NULL,
                    `phoneNo` varchar(13) NOT NULL,
                    `email` varchar(45) NOT NULL,
                    `pwd` varchar(200) NOT NULL,
                    `role_id` int NOT NULL,
                    `address_id` int NULL,
                    `created_at` TIMESTAMP NOT NULL,
                    `created_by` varchar(50) NOT NULL,
                    `updated_at` TIMESTAMP DEFAULT NULL,
                    `updated_by` varchar(50) DEFAULT NULL,
                    FOREIGN KEY (role_id) REFERENCES roles(role_id),
                    FOREIGN KEY (address_id) REFERENCES address(address_id),
                    PRIMARY KEY (`id`)
);

ALTER TABLE patient
   ADD `gp_registration_details_id` int NULL AFTER `address_id`;

ALTER TABLE patient
   ADD FOREIGN KEY (gp_registration_details_id) REFERENCES
gp_registration_details(gp_registration_details_id);
```

```
CREATE TABLE `GPSurgery` (
                    `gp_surgery_id;` int NOT NULL AUTO_INCREMENT,
                    `trading_name` varchar(45) NOT NULL,
                    `first_name` varchar(45) NOT NULL,
                    `surname` varchar(45) NOT NULL,
                    `is_accepting_new_patients` smallint(1) NOT NULL,
                    `phoneNo` varchar(13) NOT NULL,
                    `email` varchar(45) NOT NULL,
                    `pwd` varchar(200) NOT NULL,
                    `role_id` int NOT NULL,
                    `address_id` int NULL,
                    `created_at` TIMESTAMP NOT NULL,
                    `created_by` varchar(50) NOT NULL,
                    `updated_at` TIMESTAMP DEFAULT NULL,
                    `updated_by` varchar(50) DEFAULT NULL,
                    FOREIGN KEY (role_id) REFERENCES roles(role_id),
                    FOREIGN KEY (address_id) REFERENCES address(address_id),
                    PRIMARY KEY (`gp_surgery_id;`)
);


ALTER TABLE `gp_surgery`
  ADD FULLTEXT INDEX `GPSurgery_FTS` (`tradingName`, `first_name`, `surname`) VISIBLE;
```

2.2.4 MVC System Architecture



System architecture view for Spring Boot

High-level overview of the architecture flow in a Spring Boot application is as follows:

1. A client (such as a web browser) sends a request to the server for a specific resource.
2. The request is received by the server, which routes it to the appropriate controller.
3. The controller is responsible for handling the request and invoking any necessary business logic.
4. The business logic interacts with the model, which represents the data and the business rules for manipulating that data.
5. The model updates the database as necessary. If the business logic requires data from the database, the service class will interact with a repository to retrieve the data. The repository uses the Java Persistence API (JPA) to query the database and map the results to Java objects. If the business logic requires data from the database, the service class will interact with a repository to retrieve the data. The repository uses the Java Persistence API (JPA) to query the database and map the results to Java objects. The results are returned to the controller.
6. The controller prepares the data for presentation to the client and selects the appropriate view (an HTML page) to render the response.
7. The view is then rendered by the server and sent back to the client as a response to the original request.
8. The client receives the response and displays it to the user.

2.3 Implementation

2.3.1 Annotations [1,2]

@Constraint is a Java annotation that is used to specify that a given class is a constraint validation implementation.

@InputValueMatchValidator is a class that is being specified as the implementation for the constraint validation. This class must implement the ConstraintValidator interface and provide a method to validate the constraint.

@Target({ElementType.TYPE}) is an annotation that specifies the elements of a program to which an annotation type is applicable. In this case, the annotation is applicable to a type (e.g., a class, interface, or enum).

@Retention(RetentionPolicy.RUNTIME) is an annotation that specifies the visibility of an annotation type. In this case, the annotation will be available at runtime.

@Documented is a Java annotation that indicates that elements annotated with this annotation should be documented by JavaDoc.

@Component is a Spring framework annotation that indicates that an annotated class is a "component". These classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

"auditAwareImpl" is the value of the "value" attribute of the @Component annotation. It specifies the bean name of the annotated class. If the "value" attribute is not specified, the default bean name will be the uncapitalized class name with the first letter lowercased.

[1] https://springframework.guru/spring-framework-annotations/
[2] https://www.geeksforgeeks.org/spring-framework-annotations/

@Configuration - indicates that an annotated class is a source of bean definitions for the application context.

@Controller - indicates that an annotated class is a "Controller" (e.g., a web controller).

@Slf4j - a Lombok annotation that generates a Logger field with the name "log" and the correct type for the targeted environment (e.g., Log4j, Logback, etc.).

@ControllerAdvice - used to define @ExceptionHandler, @InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods.

@ExceptionHandler(Exception.class) - specifies a method that will handle exceptions of a specific type. In this case, the method will handle exceptions of the type Exception.

@Autowired - specifies that a field, constructor, or setter method should be autowired by Spring's dependency injection facilities.

@RequestMapping- used to map web requests onto specific handler classes and/or handler methods.

@GetMapping - specialized version of @RequestMapping for HTTP GET requests.

@RequestParam(value = "gpAdded", required = false) - used to bind a web request parameter to a method parameter. In this case, the value of the request parameter "gpAdded" will be bound to the method parameter. The "required" attribute specifies whether the parameter is required or not.

@PathVariable(name = "pageNum") int pageNum) -used to bind a URI template variable to a method parameter. In this case, the value of the "pageNum" path variable will be bound to the method parameter.

@Valid - used to specify that a method parameter should be validated by the Bean Validation framework.

@ModelAttribute("contactMessage") ContactMessage contactMessage - used to bind a method parameter or return value to a named model attribute. In this case, the "contactMessage" object will be added to the model with the name "contactMessage".

@Entity - JPA (Java Persistence API) annotation that is used to mark a Java class as an entity. This class will be mapped to a database table.

@Data - a Lombok annotation that generates getters, setters, toString, equals, and hashCode methods for all fields in the class.

@Id - JPA annotation that is used to mark a field as the primary key of an entity.

@GeneratedValue(strategy= GenerationType.AUTO, generator = "native") - JPA annotation that is used to specify the primary key generation strategy. In this case, the strategy is "GenerationType.AUTO" and the generator is "native". "GenerationType.AUTO" means that the persistence provider will choose the most appropriate strategy depending on the database being used. "native" is a generator that uses a database-specific function to generate a value.

@GenericGenerator(name = "native", strategy = "native") - a Hibernate-specific annotation that is used to specify a primary key generator. In this case, the generator is named "native" and its strategy is "native".

@Size(min=5, message="Address Line 1 must be at least 5 characters long") - Bean Validation constraint that is used to specify the minimum and maximum number of characters that a string field should have. In this case, the field must have at least 5 characters.

@Pattern(regexp = "^[AC-Y]{1}[0-9]{1}[0-9W]{1}[\\-]?[0-9AC-Y]{4}$", message="EirCode must be at least 7 characters long") - Bean Validation constraint that is used to specify a regex pattern that a field must match. In this case, the field must match the given pattern in order to be considered valid.

@CreatedDate - Spring Data JPA annotation that is used to mark a field as the date of creation for the entity.

@Column(updatable = false) - JPA annotation that is used to specify the mapping for a persistent field or property of an entity to a column in a database table. "updatable = false" specifies that the column should not be included in SQL UPDATE statements. This means that the value of the field will not be updated once it is persisted to the database.

@CreatedBy - Spring Data JPA annotation that is used to mark a field as the user who created the entity.

@LastModifiedDate - Spring Data JPA annotation that is used to mark a field as the date of the last modification for the entity.

@Column(insertable = false) - JPA annotation that is used to specify the mapping for a persistent field or property of an entity to a column in a database table. "insertable = false" specifies that the column should not be included in SQL INSERT statements. This means that the value of the field will not be set when the entity is persisted to the database.

@LastModifiedBy - Spring Data JPA annotation that is used to mark a field as the user who last modified the entity.

@Email(message = "Please provide a valid email address") - used to specify that a field must be a well-formed email address with the provided error message.

@NotBlank - used to specify that a field must not be blank (i.e., must not be null or empty). This constraint can be applied to fields of type String, CharSequence, or Collection.

@Transient – specifies that a field should not be persisted to the database.

@OneToOne(fetch = FetchType.EAGER,cascade = CascadeType.PERSIST, targetEntity = Roles.class) - specify a one-to-one relationship between two entities. "fetch = FetchType.EAGER" specifies that the relationship should be fetched eagerly (i.e., when the parent entity is fetched, the related entity should also be fetched). "cascade = CascadeType.PERSIST" specifies that the persistence operation should be cascaded to the related entity. "targetEntity = Roles.class" specifies the target entity class for the relationship.

@JoinColumn(name = "role_id", referencedColumnName = "roleId",nullable = false) - JPA annotation that is used to specify a column that is used to join two tables. In this case, the "role_id" column of the current table will be used to join with the "roleId" column of the "Roles" table. "nullable = false" specifies that the column cannot be null.

@ManyToOne is a JPA annotation that is used to specify a many-to-one relationship between two entities. This means that one instance of an entity is related to many instances of another entity.

@InputValueMatch.List is an annotation that is used to specify that a class or method has multiple @InputValueMatch annotations.

@InputValueMatch is an annotation that is used to specify that the value of a field must match a specific pattern. This annotation is typically used in conjunction with the @Constraint annotation to specify that a class is a constraint validation implementation.

For example, @InputValueMatch annotation might be use to ensure that a field contains a valid email address or phone number. When the constraint is applied to a field, the field's value will be validated against the specified pattern. If the value does not match the pattern, a constraint violation will be reported.

@Repository - marks a class as a data repository. This annotation is typically used to create DAO (Data Access Object) classes.

@Query - specifies a custom JPQL (Java Persistence Query Language) or native SQL query for a repository method.

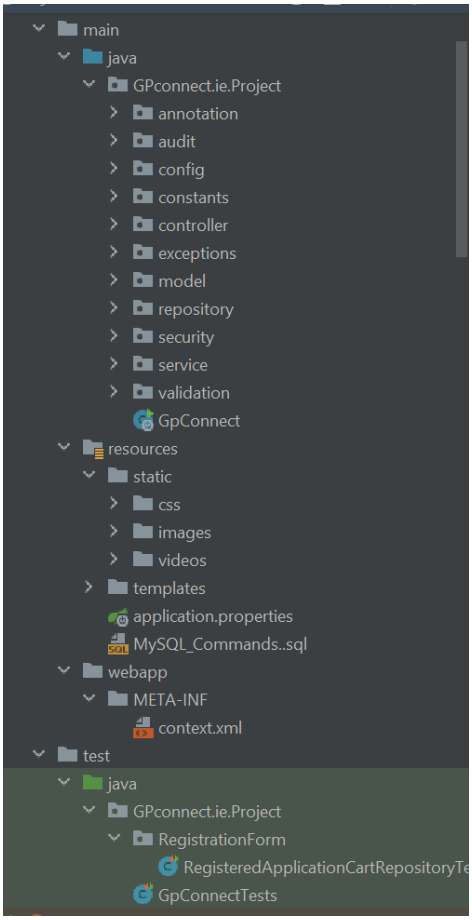
@Override is a Java annotation that is used to specify that a method is overriding a method from a superclass.

@Service - Spring framework annotation that is used to mark a class as a business service. This annotation is typically used to create service classes that perform business logic tasks.

@EntityListeners is a JPA annotation that is used to specify callback methods for an entity. These methods will be called by the JPA provider when specific events occur (e.g., when an entity is persisted, updated, or deleted).

@EntityListeners(AuditingEntityListener.class) specifies that the entity has an AuditingEntityListener" that should be used for the callback methods. An "AuditingEntityListener" is a class that provides support for auditing entities. This class typically contains callback methods that are annotated with @PrePersist, @PreUpdate, and @PreRemove to handle auditing functionality. These methods will be called by the JPA provider before an entity is persisted, updated, or removed. For example, an "AuditingEntityListener" might be use to set the created and modified dates of an entity when it is persisted or updated.

2.3.2 Main structure of the project



2.3.3 Database connection

Initially, for testing purposes I decided to use in-memory data to log in:

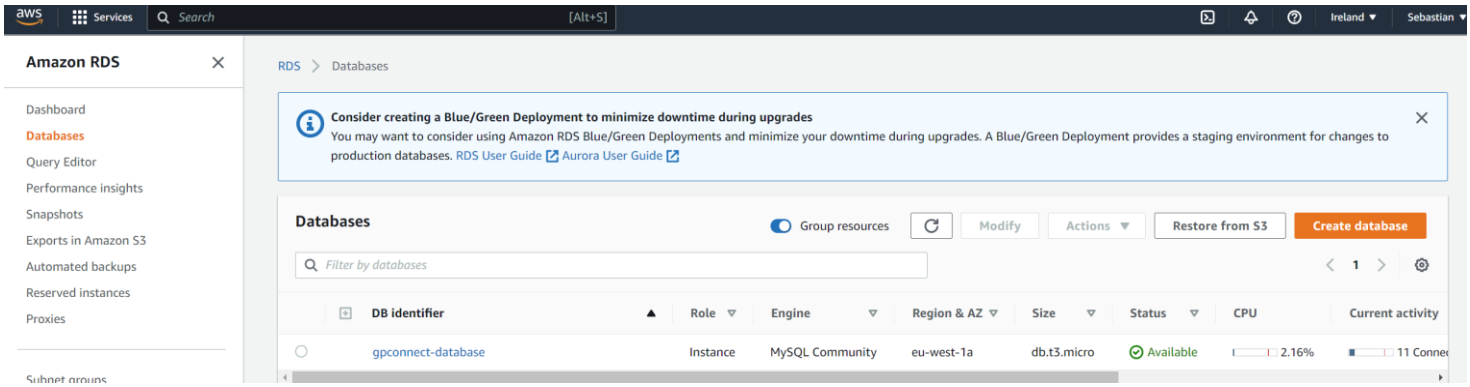
```
//    TESTING in memory login:
//    @Bean
//    public UserDetailsService users() {
//        // The builder will ensure the passwords are encoded before saving in memory
//        User.UserBuilder users = User.withDefaultPasswordEncoder();
//        UserDetails user = users
//                .username("user")
//                .password("password")
//                .roles("USER")
//                .build();
//        UserDetails admin = users
//                .username("admin")
//                .password("password")
//                .roles("USER", "ADMIN")
//                .build();
//        return new InMemoryUserDetailsManager(user, admin);
//    }

//    @Bean
//    public UserDetailsService userDetailsService(DataSource dataSource){
//        return new JdbcUserDetailsManager(dataSource);
//    }
```

As the project progressed, I decided to use AWS services for database connection of MySQL type.

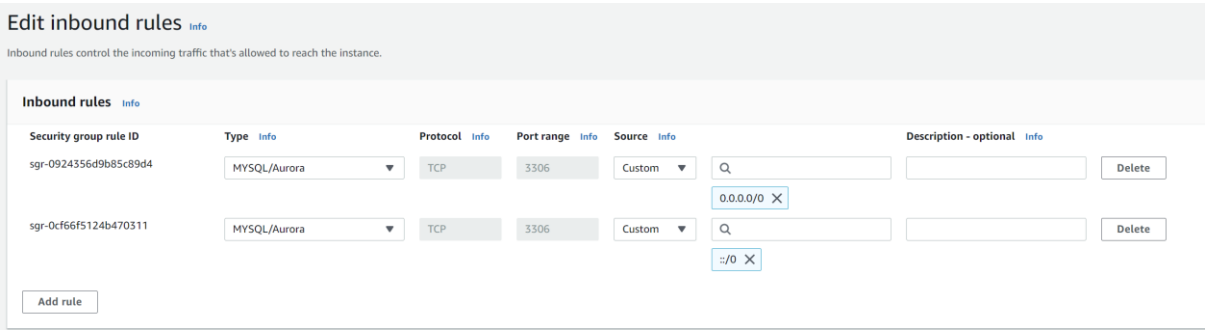
Connection properties are stated in the properties (credentials were removed):

```
spring.datasource.url=jdbc:mysql://gpconnect-database.ckult9bnkjxe.eu-west-1.rds.amazonaws.com/GPConnect

spring.datasource.username=********
spring.datasource.password=********

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
server.servlet.session.persistent=false

spring.jpa.properties.javax.persistence.validation.mode=none
```

AWS dashboard:



Required inbound configuration:



2.3.4 Security and Filters

The SecurityFilterChain represents a chain of filters that are applied to an HTTP request.

The method configures various security-related options for the HttpSecurity object, such as Cross-Site Request Forgery (CSRF) protection, authentication, and authorization. It also specifies the login and logout pages, as well as the URLs to which the user should be redirected upon successful login or logout.

```java
@Configuration
public class ProjectSecurityConfig {
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf().ignoringRequestMatchers( ...patterns: "/saveMsg").ignoringRequestMatchers( ...patterns: "/public/**").and() HttpSecurity
                .authorizeHttpRequests( (auth)->auth
                    .requestMatchers( ...patterns: "/dashboard", "/logout", "/logoutGP", "/displayProfile", "/updateProfile","/displaySubmittedGpRegistrationForms"
                            ,"/displayGpRegistrationDetails", "/updateGpRegistrationDetails").authenticated()
                        .requestMatchers( ...patterns: "/displayMessages", "/closeMsg").hasRole("ADMIN")
                        .requestMatchers( ...patterns: "/displaySubmittedGpRegistrationForms", "/gp/**").hasRole("GPSURGERY")
                        .requestMatchers( ...patterns: "/patient/**").hasRole("PATIENT")
                    .requestMatchers( ...patterns: "/home","/contact", "/login", "/loginGP", "/saveMsg", "/searchGP/**","/images/*","/videos/*",
                            "/GPSurgerySearch","/css/*", "/register", "/public/**").permitAll()
                ).formLogin().loginPage("/login") FormLoginConfigurer<HttpSecurity>
                    .defaultSuccessUrl( defaultSuccessUrl: "/dashboard", alwaysUse: true).failureUrl( authenticationFailureUrl: "/login?error=true").permitAll()
                    .and().logout().logoutSuccessUrl("/login?logout=true").invalidateHttpSession(true).permitAll() LogoutConfigurer<HttpSecurity>
                    .and().httpBasic();
            return http.build();
    }
```

In terms of Authentication, I initially used implementation of UserDetailsService and UserDetails but later I have switched to implementing AuthenticationProvider instead. The final result is applicable to both log in as Patient and GP Surgery and is as follows:

```java
    //This method will apply to both patient and gp surgery log in attempts
    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String email = authentication.getName();
        String pwd = authentication.getCredentials().toString();
        Patient patient = patientRepository.readByEmail(email);

        GpSurgery gpSurgery = gpSurgeryRepository.readByEmail(email);
        //Check if exists in patient table
        if(null != patient && patient.getId()>0 &&
                //checking hash values
                passwordEncoder.matches(pwd, patient.getPwd())){
            //UsernamePasswordAuthenticationToken is implementation of Authentication class
            //pwd will be passed as null because credentials are not stored by spring boot
            return new UsernamePasswordAuthenticationToken(
                    email, credentials: null, getGrantedAuthorities(patient.getRoles()));

        }//Else check if exists in patient table
        else if(null == patient && null != gpSurgery && gpSurgery.getGpSurgeryId()>0 &&
                //checking hash values
                passwordEncoder.matches(pwd, gpSurgery.getPwd())) {
            //UsernamePasswordAuthenticationToken is implementation of Authentication class
            //pwd will be passed as null because credentials are not stored by spring boot
            return new UsernamePasswordAuthenticationToken(
                    email, credentials: null, getGrantedAuthorities(gpSurgery.getRoles()));
        }else{
            throw new BadCredentialsException("Invalid credentials!");
        }
    }
    2 usages
    private List<GrantedAuthority> getGrantedAuthorities(Roles roles) {
        List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
        grantedAuthorities.add(new SimpleGrantedAuthority( role: "ROLE_"+roles.getRoleName()));
        return grantedAuthorities;
    }
    @Override
    public boolean supports(Class<?> authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
```

In addition the application use hashing algorithm of BCrypt type encapsulated in BCrypt PasswordEncoder. It is specifically designed to be resistant to brute-force attacks, which are a type of attack that involves trying many different password combinations in quick succession in an attempt to guess the correct password. Password is passed through the BCrypt function along with a "salt" value. The salt is a random string of characters that is generated for each password and is used to make the resulting hash value unique. The BCrypt function then generates a hash value based on the password and salt, which can be stored in a database or used for comparison purposes. One of the key benefits of BCrypt is that it is designed to be computationally expensive to execute.

```
@Bean
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Another security feature is CSRF token which is a unique, secret value that is generated by the server and included in the HTML of a web page. When a user submits a form or makes an HTTP request, the CSRF token is sent back to the server along with the request. The server can then verify that the request is genuine by checking that the CSRF token is valid. If the CSRF token is not present or is invalid, the request is considered to be a potential CSRF attack and is rejected.

CSRF token is generated by the below code in login form:

```
<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
```

2.3.5 Dashboard – initial page after login

The below controller will be always called after successful login and store user details from Authentication into HttpSession to enable usage of this data through the application. The result of using one component with the below code for login is that GP may log in using view/html designated for patient, however despite the fact of additional database query, the result is satisfactory for the purpose of the project. However, when scaling up the application, refactoring and separate login and queries will be beneficial.

```
//User is redirected to this method straight after Login to display dashboard and set attributes to Session
@RequestMapping("/dashboard")
public String displayDashboard(Model model, Authentication authentication, HttpSession httpSession){
    Patient patient = patientRepository.readByEmail(authentication.getName());

    //In case the patient was not found, check GPSurgery in database
    if(null == patient){
        GpSurgery gpSurgery = gpSurgeryRepository.readByEmail(authentication.getName());
        model.addAttribute( attributeName: "username", gpSurgery.getFirstName().concat( str: " " + gpSurgery.getSurname()));
        httpSession.setAttribute( s: "loggedInGpSurgery", gpSurgery);
    }else{
        model.addAttribute( attributeName: "username", patient.getFirstName().concat( str: " " + patient.getSurname()));
        httpSession.setAttribute( s: "loggedInPatient", patient);
    }

    //add role details to the model
    model.addAttribute( attributeName: "roles", authentication.getAuthorities().toString());
    return "dashboard.html";
}
```

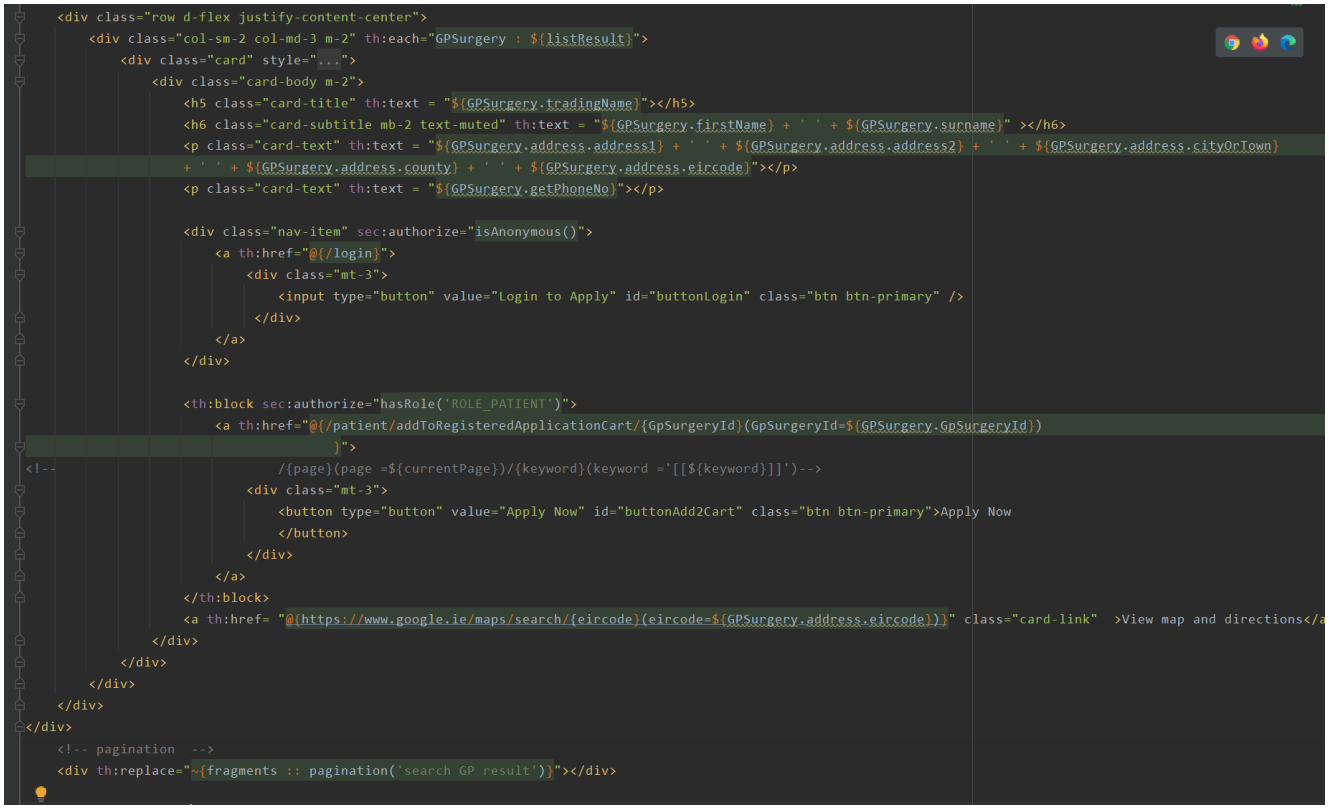
2.3.6 Example of flow in GP search use case

I decided to use Full-Text Search in the database:

```
ALTER TABLE `gp_surgery`
    ADD FULLTEXT INDEX `GPSurgery_FTS` (`tradingName`, `first_name`, `surname`) VISIBLE;
```

It provided the ability to search for words and phrases within a database table that has a Full-Text index. A Full-Text index allows to search for words and phrases within a column of a table, and returns rows that contain the search term or terms. Full-Text indexing is available in MySQL for MyISAM and InnoDB tables, and can be used with CHAR, VARCHAR, and TEXT columns.

View/HTML for GP search:



HTML fragment responsible for front-end pagination:

```
<div th:fragment="pagination(entityName)" th:remove="tag">
    <div class="text-center m-1" th:if="${totalItems > 0}">
        <span>Showing [[${entityName}]] # [[${startCount}]] to [[${endCount}]] of [[${totalItems}]]</span>
    </div>
    <div class="text-center m-1" th:unless="${totalItems > 0}">
        <span>No [[${entityName}]] found</span>
    </div>

    <div th:if="${totalPages > 1}">
        <nav>
            <ul class="pagination justify-content-center flex-wrap">
                <li th:class="${currentPage > 1 ? 'page-item' : 'page-item disabled'}">
                    <a th:replace="~{fragments :: page_link(1, 'First')}"></a>
                </li>
                <li th:class="${currentPage > 1 ? 'page-item' : 'page-item disabled'}">
                    <a th:replace="~{fragments :: page_link(${currentPage - 1}, 'Previous')}"></a>
                </li>

                <li th:class="${currentPage != i ? 'page-item': 'page-item active'}"
                    th:each="i : ${#numbers.sequence(1, totalPages)}">
                    <a th:replace="~{fragments :: page_link(${i}, ${i})}"></a>
                </li>

                <li th:class="${currentPage < totalPages ? 'page-item' : 'page-item disabled'}">
                    <a th:replace="~{fragments :: page_link(${currentPage + 1}, 'Next')}"></a>
                </li>

                <li th:class="${currentPage < totalPages ? 'page-item' : 'page-item disabled'}">
                    <a th:replace="~{fragments :: page_link(${totalPages}, 'Last')}"></a>
                </li>
            </ul>
        </nav>
    </div>
</div>
```

Controller layer:

```
@GetMapping("/searchGP")
public ModelAndView search(String keyword, Model model) {

    return searchByPage(keyword, model, pageNum: 1);
}

1 usage
@GetMapping("/searchGP/page/{pageNum}")
//The method receives @Param annotated keyword and page num parameter which is used to search for GP Surgeries
public ModelAndView searchByPage(@Param("keyword") String keyword, Model model,
                                 @PathVariable(name = "pageNum") int pageNum) {

    //get the search results from the returned Page object and stores them in a List object named listResult
    Page<GpSurgery> result = gpSurgeryService.search(keyword, pageNum);
    List<GpSurgery> listResult = result.getContent();

    ModelAndView modelAndView = new ModelAndView( viewName: "GPSurgerySearch.html");

    //To display pagination correctly, we add various data to the Model object,
    // including the total number of pages, the total number of items, the current page number,
    // and the list of search results.
    model.addAttribute( attributeName: "totalPages", result.getTotalPages());
    model.addAttribute( attributeName: "totalItems", result.getTotalElements());
    model.addAttribute( attributeName: "currentPage", pageNum);

    long startCount = (pageNum - 1) * gpSurgeryService.SEARCH_RESULT_PER_PAGE + 1;
    model.addAttribute( attributeName: "startCount", startCount);

    long endCount = startCount + gpSurgeryService.SEARCH_RESULT_PER_PAGE - 1;
    if (endCount > result.getTotalElements()) {
        endCount = result.getTotalElements();
    }

    model.addAttribute( attributeName: "endCount", endCount);
    model.addAttribute( attributeName: "listResult", listResult);
    model.addAttribute( attributeName: "keyword", keyword);

    modelAndView.addObject( attributeName: "listResult",listResult);
    return modelAndView;
}
@GetMapping("/displaySubmittedGpRegistrationForms")
```

Service layer:

```
1 usage
public Page<GpSurgery> search(String keyword, int pageNum) {
    //We can also pass Sort class as 3rd parameter, but database will decide here using full text
    Pageable pageable = PageRequest.of( page: pageNum - 1, SEARCH_RESULT_PER_PAGE);
    return gpSurgeryRepository.searchAvailGPSurgeriesByName(keyword, pageable);
}
```

Repository layer using native MySQL Query:

```
@Repository
public interface GPSurgeryRepository extends PagingAndSortingRepository<GpSurgery, Integer> {
    2 usages
    GpSurgery readByEmail(String email);

    //Using Full Text search - must have custom query as it is specific to MySQL
    1 usage
    @Query(value = "SELECT * FROM gp_surgery WHERE is_accepting_new_patients = 1 AND " +
            " MATCH(trading_name, first_name, surname) " +
            "AGAINST (?1)", nativeQuery = true)
    public Page<GpSurgery> searchAvailGPSurgeriesByName(String keyword, Pageable pageable);

    1 usage
    GpSurgery findById(Integer gpSurgeryId);
}
```

2.3.7 Example of Custom validations & annotations

Custom interface for checking input in password upon registration of a user:

```
@Documented
@Constraint(validatedBy = PwdStrenghtValidator.class)
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface PwdValidator {
    String message() default "Please use a strong password";
    Class<?>[] groups() default{};
    Class<? extends Payload>[] payload() default {};
}
```

Custom interface for checking input match upon registration of a user in password and email:

```java
@Constraint(validatedBy = InputValueMatchValidator.class)
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface InputValueMatch {
    String message() default "Input does not match";
    Class<?>[] groups() default{};
    Class<? extends Payload>[] payload() default {};

    String input();

    3 usages
    String inputMatch();

    @Target({ElementType.TYPE})
    @Retention(RetentionPolicy.RUNTIME)
    @interface List {
        InputValueMatch[] value();
    }

}
```

Password validator with most common password on black list:

```java
public class PwdStrenghtValidator implements ConstraintValidator<PwdValidator, String> {

    2 usages
    List<String> weakPwd;

    @Override
    public void initialize(PwdValidator pwdValidator){
        weakPwd = Arrays.asList("123456", "qwerty","password","qwerty123","abc123","password123","000000");
    }

    @Override
    public boolean isValid(String pwdField, ConstraintValidatorContext constraintValidatorContext) {
        return pwdField != null && (!weakPwd.contains(pwdField));
    }
}
```

Input match class logic taking into account BCtypr encoder:

```java
public class InputValueMatchValidator implements ConstraintValidator<InputValueMatch, Object> {
    2 usages
    private String input;
    2 usages
    private String inputMatch;
    @Autowired
    PasswordEncoder passwordEncoder;

    @Override
    public void initialize(InputValueMatch constraintAnnotation){
        this.input = constraintAnnotation.input();
        this.inputMatch = constraintAnnotation.inputMatch();
    }

    @Override
    public boolean isValid(Object value, ConstraintValidatorContext constraintValidatorContext) {
        Object inputValue = new BeanWrapperImpl(value)
                .getPropertyValue(input);
        Object inputMatchValue = new BeanWrapperImpl(value)
                .getPropertyValue(inputMatch);
        if (inputValue != null) {
            if(inputValue.toString().startsWith("$2a")){
               return true;
            }else {
                return inputValue.equals(inputMatchValue);
            }
        } else {
            return inputMatchValue == null;
        }
    }
}
```

Annotation used on class to enable the validation:

```java
@InputValueMatch.List({
        @InputValueMatch(
                input = "pwd",
                inputMatch = "confirmPwd",
                message = "Passwords do not match!"
        ),
        @InputValueMatch(
                input = "email",
                inputMatch = "confirmEmail",
                message = "Email addresses do not match!"
        )
})
public class Patient extends BaseEntity{
```

Variables that are validated:

```
@NotBlank(message="Email cannot be blank")
@Email(message = "Please provide a valid email address" )
private String email;

@NotBlank(message="Email cannot be blank")
@Email(message = "Please provide a valid email address" )
@Transient
private String confirmEmail;

@NotBlank(message = "Password cannot be blank")
@Size(min = 5, message = "Password have to be longer than 5 characters")
@PwdValidator
private String pwd;

@NotBlank(message = "Password cannot be blank")
@Size(min = 5, message = "Password have to be longer than 5 characters")
@Transient
private String confirmPwd;
```

2.3.8 Example of populating inputs for UI

I have decided to implement logic that queries database of a login patient and populates data back to the view to enhance user experience in all inputs fields. It operation can be presented when a patient filled out Profile details but did not submitted GP registration details and nonetheless proceeds with pending applications.

Controller:

```
@RequestMapping ("/continue")
public ModelAndView continiueApplication(Model model, HttpSession httpSession){
    Patient patient = (Patient) httpSession.getAttribute( s: "loggedInPatient");

    //Method to extract ProfileDetails details from Patient object and return it
    ProfileDetails profileDetails = patientService.extractProfileDetailsFromPatient(patient);

    //Method to extract GpRegistrationDetails details from Patient object and return it
    GpRegistrationDetails gpRegistrationDetails = patientService.extractGpRegistrationDetailsFromPatient(patient);

    ModelAndView modelAndView = new ModelAndView( viewName: "submittGpApplication.html");
    modelAndView.addObject( attributeName: "profileDetails",profileDetails);
    modelAndView.addObject( attributeName: "gpRegistrationDetails",gpRegistrationDetails);

    return modelAndView;
}
```

Service layer:

```
public GpRegistrationDetails extractGpRegistrationDetailsFromPatient(Patient patient){
    GpRegistrationDetails gpRegistrationDetails = new GpRegistrationDetails();

    if(patient.getGpRegistrationDetails() !=null && patient.getGpRegistrationDetails().getGpRegistrationDetailsId()>0) {
        gpRegistrationDetails.setDob(patient.getGpRegistrationDetails().getDob());
        gpRegistrationDetails.setPpsn(patient.getGpRegistrationDetails().getPpsn());
        gpRegistrationDetails.setGenderEnum(patient.getGpRegistrationDetails().getGenderEnum());
        gpRegistrationDetails.setMedicalCardNo(patient.getGpRegistrationDetails().getMedicalCardNo());
    }
    return gpRegistrationDetails;
}
```

```
public ProfileDetails extractProfileDetailsFromPatient(Patient patient){

    ProfileDetails profileDetails = new ProfileDetails();

    profileDetails.setFirstName(patient.getFirstName());
    profileDetails.setSurname(patient.getSurname());
    profileDetails.setPhoneNo(patient.getPhoneNo());
    profileDetails.setEmail(patient.getEmail());

    if(patient.getAddress() !=null){
        profileDetails.setAddress1(patient.getAddress().getAddress1());
        profileDetails.setAddress2(patient.getAddress().getAddress2());
        profileDetails.setCityOrTown(patient.getAddress().getCityOrTown());
        profileDetails.setCounty(patient.getAddress().getCounty());
        profileDetails.setEircode(patient.getAddress().getEircode());
    }
    return profileDetails;
}
```

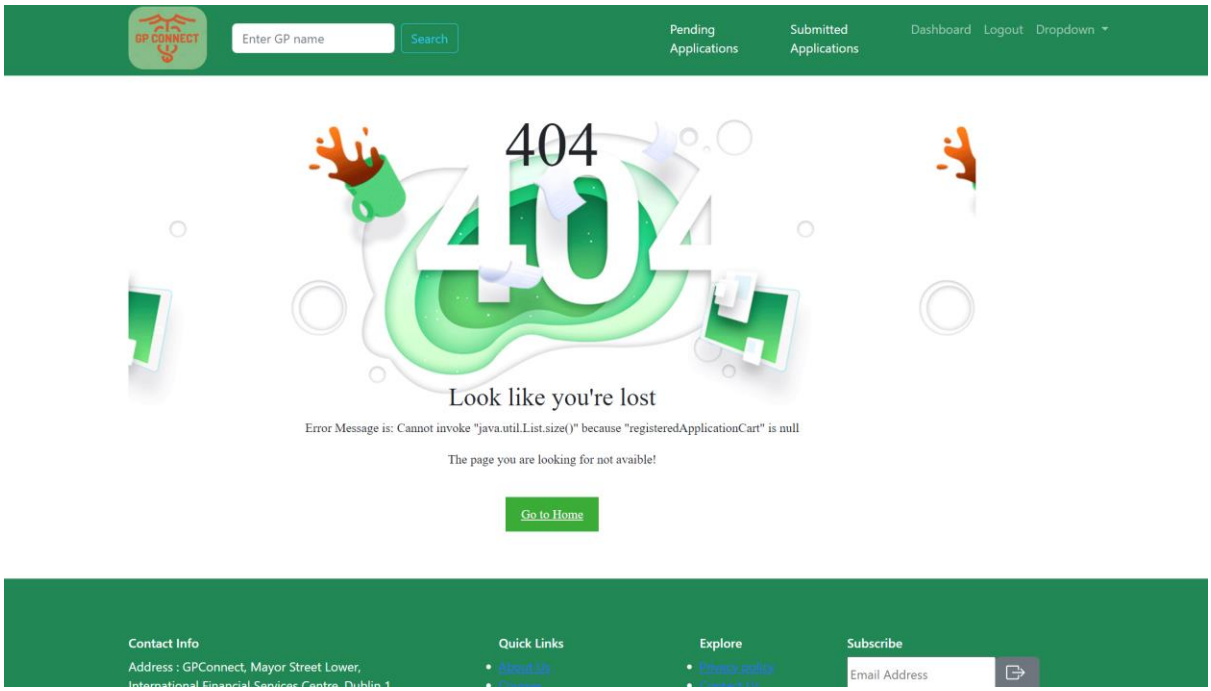
2.3.9 Global Exception handler

I have defined a global exception handler to handle exceptions that are thrown from any controller in your application. This can be useful for providing a consistent error handling mechanism across application, and for centralizing the logging of exceptions.

```
@ControllerAdvice
@Slf4j
//This will set default global error page and display the error message
public class GlobalExceptionController {
    @ExceptionHandler(Exception.class)
    public ModelAndView exceptionHandler(Exception exception){
        ModelAndView errorPage = new ModelAndView();
        errorPage.setViewName("error");
        errorPage.addObject( attributeName: "errorMessage", exception.getMessage());
        return errorPage;
    }
}
```

To display global error, I have introduced an error by commenting the below return and returning null value on purpose and the result is as follows:

```
public List<RegisteredApplicationCart> findByPatient (Patient patient, Boolean bool){
    List<RegisteredApplicationCart> registeredApplicationCartList = registeredApplicationCartRepository.findByPatientAndWasSubmitted(patient, bool);
    //return registeredApplicationCartList;
    return null;
}
```



2.3.10 Auditing Component

Spring Boot Auditing is used to automatically populate created-by, created-date, last-modified-by and last-modified-date fields in an entity. This can be achieved by using the @EnableJpaAuditing annotation.

```
@SpringBootApplication
@EnableJpaRepositories("GPconnect.ie.Project.repository")
@EntityScan("GPconnect.ie.Project.model")
@EnableJpaAuditing(auditorAwareRef = "auditAwareImpl")
public class GpConnect {

    public static void main(String[] args) {
        SpringApplication.run(GpConnect.class, args);
    }

}
```

Creating optional component to override the default behaviour for populating the created-by and last-modified-by fields.

```java
@Component("auditAwareImpl")
public class AuditAwareImpl implements AuditorAware<String> {

    @Override
    public Optional<String> getCurrentAuditor() {
        //Return null if anonymous user
        return Optional.ofNullable(SecurityContextHolder.getContext().getAuthentication().getName());
    }
}
```

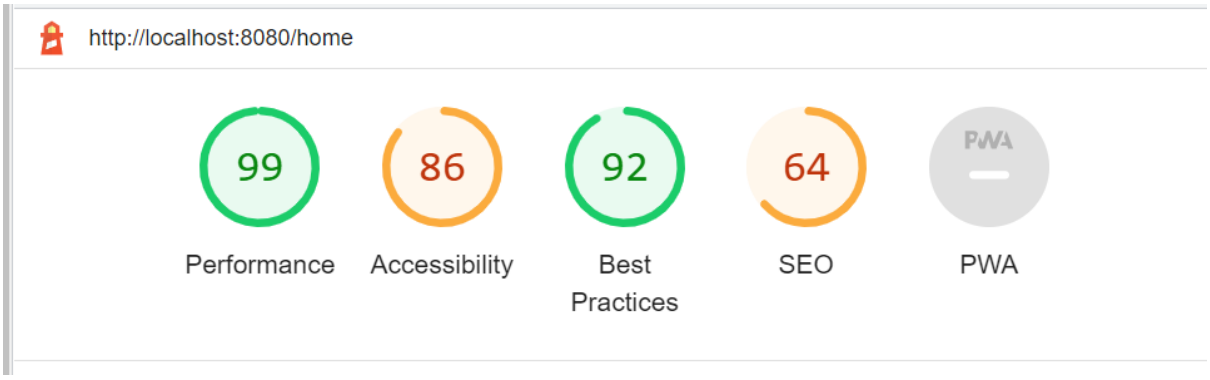
Base entity using auditing that is then extended by other java classes:

```java
@Data
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
public class BaseEntity {

    @CreatedDate
    @Column(updatable = false)
    private LocalDateTime createdAt;
    @CreatedBy
    @Column(updatable = false)
    private String createdBy;
    @LastModifiedDate
    @Column(insertable = false)
    private LocalDateTime updatedAt;
    @LastModifiedBy
    @Column(insertable = false)
    private String updatedBy;
}
```

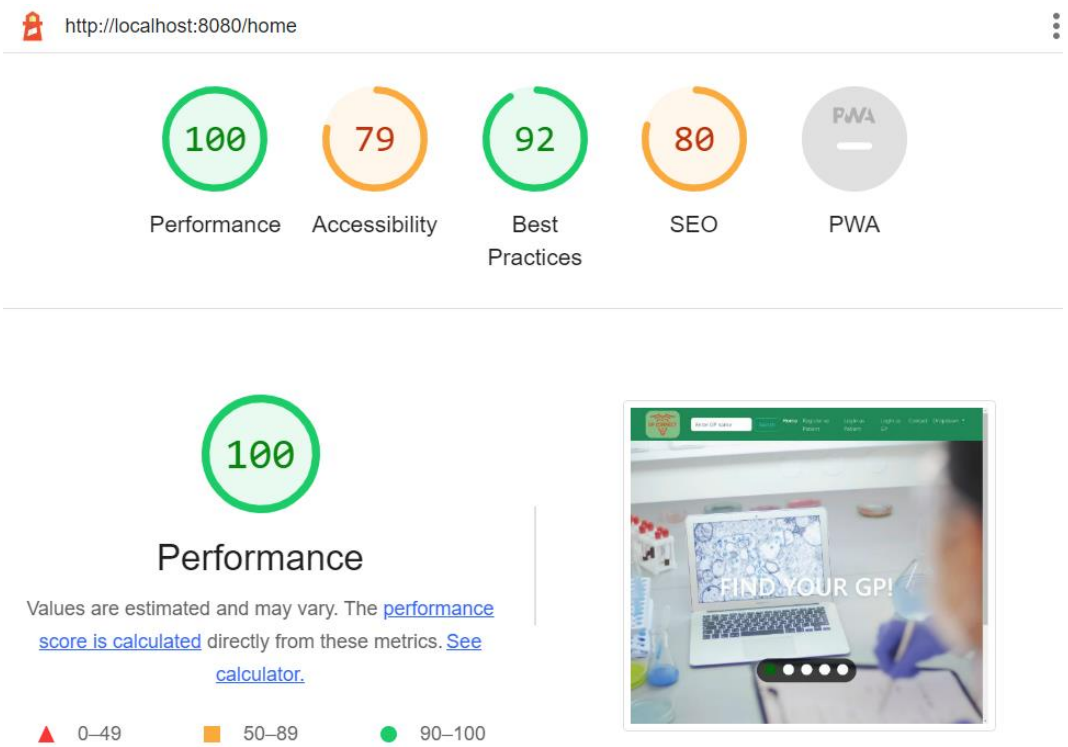
3 Testing

3.1 Google Lighthouse

Google Lighthouse initial result on home page:



Google Lighthouse after some improvement:



3.2 Main Flow test cases

Main logic flow test cases:

Project Name	GP Connect - Main logic flow
Module Name	Testing Registration
Reference Document	
Created By	Sebastian Konefal Student Number 21197458
Date Created	06/01/2023
Date Reviewed	

TEST CASE ID	TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS
Main_Logic _Flow_TC_01	Successful registration	1. Website is online 2. No account associated with email	1. Insert valid first name, surname, email, password and mobile numb 2. Click Register	Patient6, Patient6, patient6@gmail.com, 12345, 089365658	Registration completed, data persisted, user informed	Registration completed, data persisted, user informed, Alert pop up "Your registration was successful. You may now	Pass

Database state:

id	first_name	surname	phone_no	email	pwd	role_id	gp_registratio...	created_at	created_by	updated_at	updated_by
8	Patient6	Patient6	089365658	patient6@gmail....	$2a$10$OjxyCl...	2	NULL	NULL	2023-01-08 16:...	anonymousUser	NULL	NULL

Main_Logic _Flow_TC_02	Log in with created credentials	1. Website is online	1. Insert valid first name, surname, email, password and mobile numb	Patient6, Patient6, patient6@gmail.com, 12345, 089365658 N/A	User logged in, CSFR Token created, directed to dashboard	User logged in, CSFR Token created, directed to dashboard	Pass

Database state:

gp_surgery_id	trading_name	first_name	surname	is_accepting_...	phone_no	email	pwd	role_id	address_id	created_at	created_by
3	Beechwood Me...	Rita	Brennan	1	01 4961150	rita@beechwoo...	$2a$12$bKttDC...	3	2	2022-12-19 15:...	

Main_Logic _Flow_TC_03	Search a GP surgery	1. Website is online	1. GP name inserted into search box 2. Click Search	rita brennan rita brennan	Database returned the result and was displayed	Database returned the result and was displayed	Pass
Main_Logic _Flow_TC_04	Apply to GP Surgery	1. Website is online	2. Click Apply Now	<Clicked on button - Apply Now>	Data persisted, user informed	Data persisted, Message pop up "You have succesfully added GP to your pending applications"	Pass

Database state:

registered_application_cart_id	gp_surgery_id	patient_id	was_submitted	created_at	created_by	updated_at	updated_by
32	3	8	0	2023-01-08 18:...	patient6@gmail.com	NULL	NULL

TEST CASE ID	TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS
Main_Logic _Flow_TC_05	Confirm or Insert personal details and Submitt	1. Website is online 2. View Pending Applications 3. Repeat Main_Logic _Flow_TC_03-04 to add second GP surgery	1. Insert valid Address 1, Address 2, Town/City, County, Eircode 2. Insert valid DOB, PPSN, Medical card, gender 3. Click submit all applications	98 Baggot Ln, Ballsbridge, Dublin 4, Co. Dublin, D04 PP27 15.05.1993, 2154354BP, 5F78562T, GenderEnum.MALE	Data persisted for all pending applications	Data persisted, Alert pop up "You have succesfully submitted all you applications"	Pass

Database state:

registered_application_cart_id	gp_surgery_id	patient_id	was_submitted	created_at	created_by	updated_at	updated_by
32	3	8	1	2023-01-08 18:50:13	patient6@gmail.com	2023-01-08 19:06:09	patient6@gmail.com
33	9	8	1	2023-01-08 19:08:16	patient6@gmail.com	2023-01-08 19:14:13	patient6@gmail.com

3.3 Example of Junit testing:

```
@DataJpaTest
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
@Rollback(true)
public class RegisteredApplicationCartRepositoryTest {
    1 usage
    @Autowired
    private RegisteredApplicationCartRepository registeredApplicationCartRepository;

    2 usages
    @Autowired
    private TestEntityManager entityManager;

    @Test
    public void testSaveSubmittedGpApplication(){
        Integer patientId = 1;
        Integer gpSurgeryId = 1;

        Patient patient = entityManager.find(Patient.class, patientId);
        GpSurgery gpSurgery = entityManager.find(GpSurgery.class, gpSurgeryId);

        RegisteredApplicationCart newRegisteredApplicationCart = new RegisteredApplicationCart();
        newRegisteredApplicationCart.setPatient(patient);
        newRegisteredApplicationCart.setGpSurgery(gpSurgery);
        newRegisteredApplicationCart.setWasSubmitted(Boolean.TRUE);

        RegisteredApplicationCart savedRegisteredApplicationCart = registeredApplicationCartRepository.save(newRegisteredApplicationCart);

        assertThat(savedRegisteredApplicationCart.getRegisteredApplicationCartId()).isGreaterThan( other: 0);
    }
}
```

3.4 Test cases – registration validation:

Project Name	GP Connect
Module Name	Testing Registration
Reference Document	
Created By	Sebastian Konefal Student Number 21197458
Date Created	06/01/2023
Date Reviewed	

TEST CASE ID	TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	STATUS
Registration_TC_01	Successful submit	1. Website is online 2. No account associated with email	1. Insert valid first name, surname, email, password and mobile numb 2. Click Register	Patient6, Patient6, patient6@gmail.com, 12345, 089365658	Registration completed, data persisted, user informed	Registration completed, data persisted, user informed, Alert pop up "Your registration was successful. You may now login with the credentials"	Pass
Registration_TC_02	Label is displayed for all the fields	1. Website is online	N/A		Labels are visible to user	Asterisks are visible to user	Pass
Registration_TC_03	Validation of password inserted into two separate fields	1. Website is online 2. No account is associated with email	1. Insert valid first name, surname, email, password and mobile numb 2. Password entered into two field are different 3. Click Register	Patient7, Patient7, patient7@gmail.com, 089365658 12345, 54321	Data not persisted, User informed about error	Data not persisted, Message pop up "Passwords do not match!"	Pass
Registration_TC_04	Validation of email inserted into two separate fields	1. Website is online 2. No account is associated with email	1. Insert valid first name, surname, email and mobile number 2. Emal entered into two field are different 3. Click Register	Patient7, Patient7, 12345, 089365658 patient7@gmail.com, patient8@gmail.com	Data not persisted, User informed about error	Data not persisted, Message pop up "Email addresses do not match!"	Pass
Registration_TC_05	Validation on emial already exisiting in database	1. Website is online 2. An Account is already associated with email	1. Insert valid first name, surname, email, password and mobile numb 2. Email entered already exists in database 3. Click Register	Patient7, Patient7, 12345, 089365658 patient3@gmail.com	Data not persisted, User informed about error	Data not persisted, Message pop up "Email is already registered"	Pass
Registration_TC_06	Irish landline phone numbers are accepted	1. Website is online 2. No account is associated with email	1. Insert valid first name, surname, email, password 2. Insert irish landline number 3. Click Register	Patient7, Patient7, patient7@gmail.com 12408787	Registration completed, data persisted, user informed	Registration completed, data persisted, user informed, Alert pop up "Your registration was successful. You may now login with the credentials"	Pass
Registration_TC_07	Irish Mobile phone numbers are accepted	1. Website is online 2. No account is associated with email	1. Insert valid first name, surname, email, password 2. Insert Irish Mobile number 3. Click Register	Patient7, Patient7, patient7@gmail.com 893562554	Registration completed, data persisted, user informed	Registration completed, data persisted, user informed, Alert pop up "Your registration was	Pass

4 GUI screenshots and explanations

4.1 Header and footer

The application contains header and footer in a separate files to reduce code repetition, therefore it is consistent throughout the application allowing for small changes depending the authorisation type details such as Role or Authorisation/isAnonnymous.

Thymeleaf fragment for header:

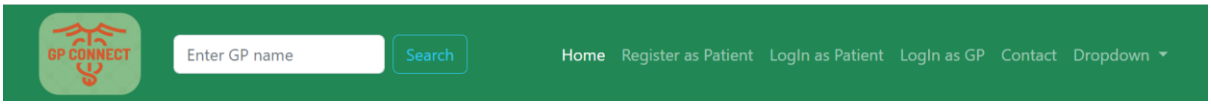
```
<!-- header -->
<div th:replace="~{header :: header}"></div>
<!-- //header -->
```

Part of header that enables various links depending on Authorisation type:

```
<li class="col-sm-6 col-lg-3" sec:authorize="hasRole('ROLE_PATIENT')">
    <a
            class="nav-link active" aria-current="page" th:href="@{/patient/viewSubmittedApplications}">Submitted Applications</a>
</li>
<li class="col-sm-6 col-lg-3" sec:authorize="hasRole('ROLE_GPSURGERY')">
    <a
            class="nav-link active" aria-current="page" th:href="@{/displaySubmittedGpRegistrationForms}">View Applications</a>
</li>
<li class="nav-item" sec:authorize="isAuthenticated()">
    <a
            th:href="@{/dashboard}" class="nav-link">Dashboard</a>
</li>
```

Logo and application name is displayed in the top left corner of the header which also contains a navigation bar to various subpages to engage with three use cases such as find GP, login and register or submit contact form.

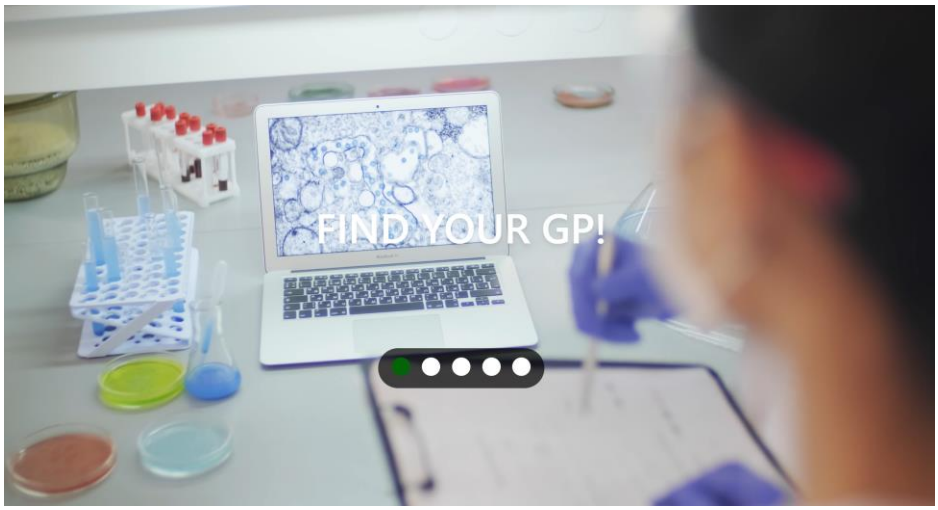
The Homepage contains a search bar to find a GP surgery.



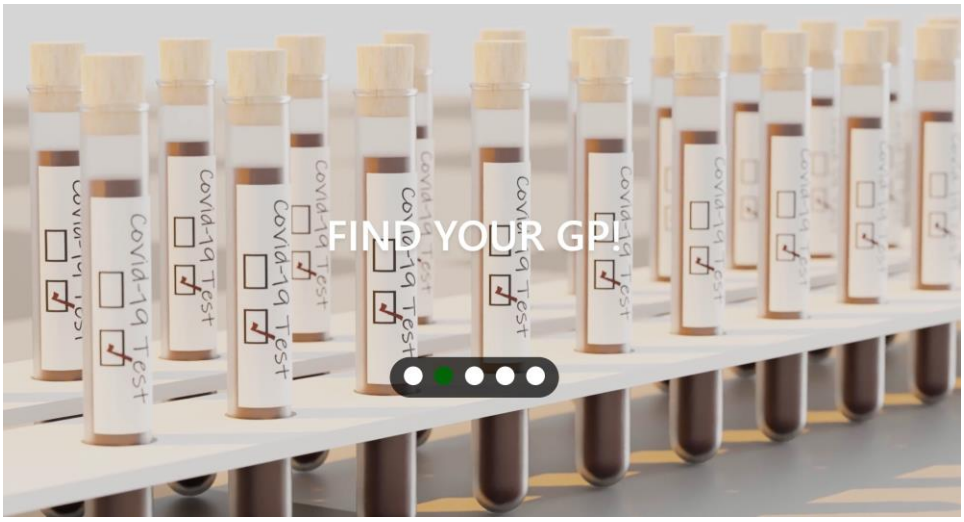
4.2 Home Page and video slider

Home page contains video slider with 5 videos:

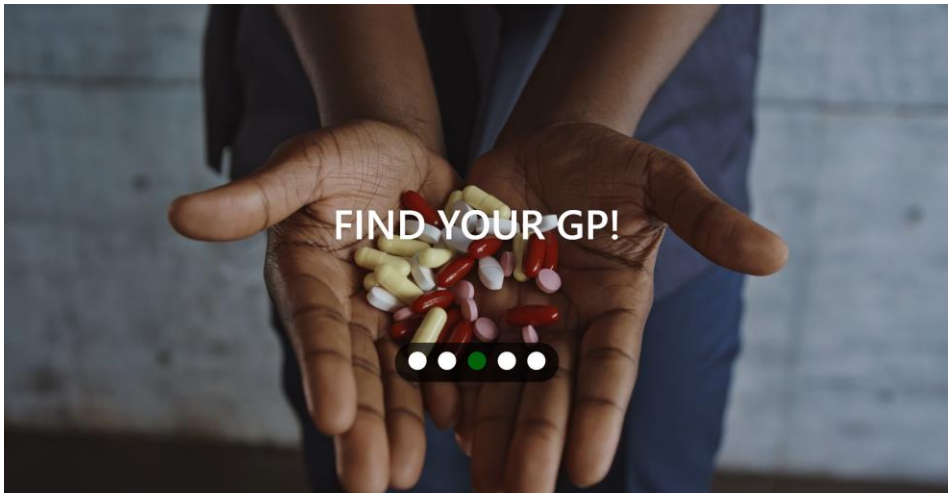
Video 1:



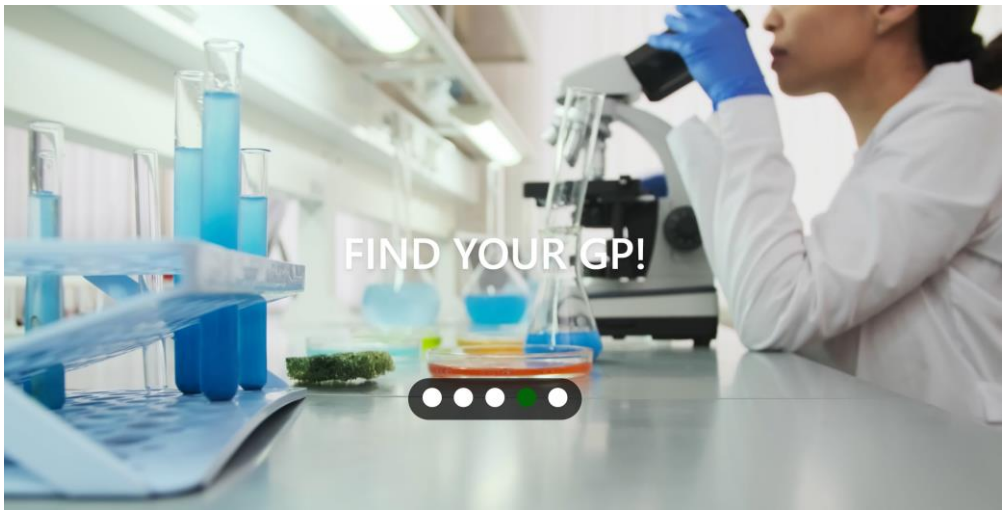
Video 2:



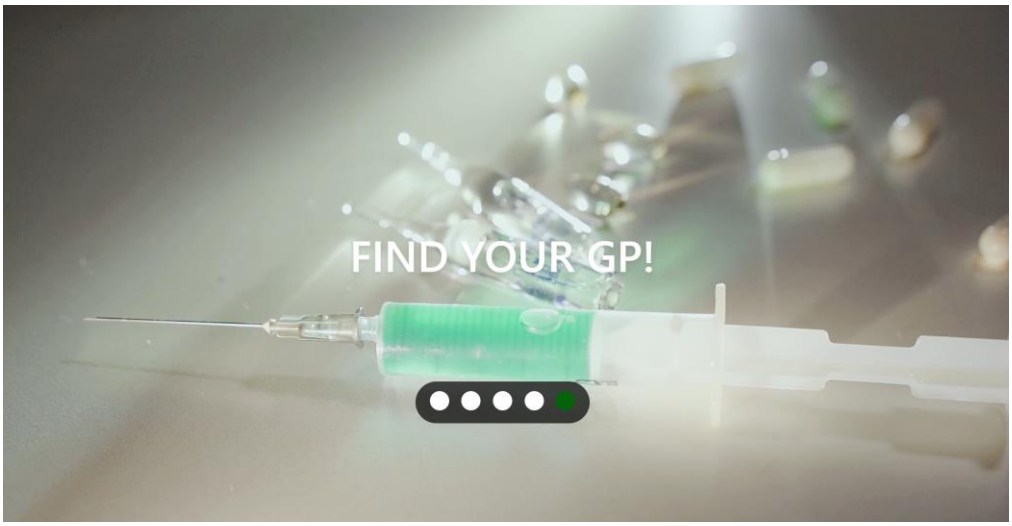
Video 3:



Video 4:



Video 5:



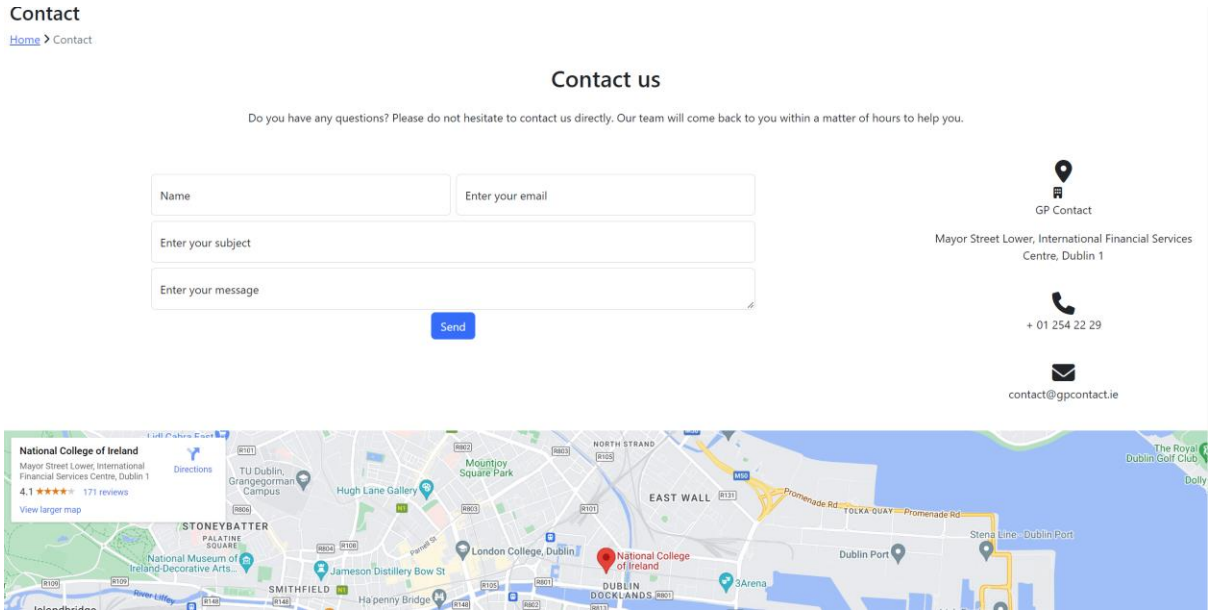
Code responsible for change of the videos:

```
<div class="controls">
    <span class="vid-btn active" data-src="/videos/video1.mp4" alt="GP video - labolatory"></span>
    <span class="vid-btn" data-src="/videos/video2.mp4" alt="GP video - vials of blood"></span>
    <span class="vid-btn" data-src="/videos/video3.mp4" alt="GP video - pills"></span>
    <span class="vid-btn" data-src="/videos/video4.mp4" alt="GP video - microscope"></span>
    <span class="vid-btn" data-src="/videos/video5.mp4" alt="GP video - syringe"></span>
</div>
<div class="video-container">
    <video src="/videos/video1.mp4" id="video-slider" loop autoplay muted></video>
</div>
```

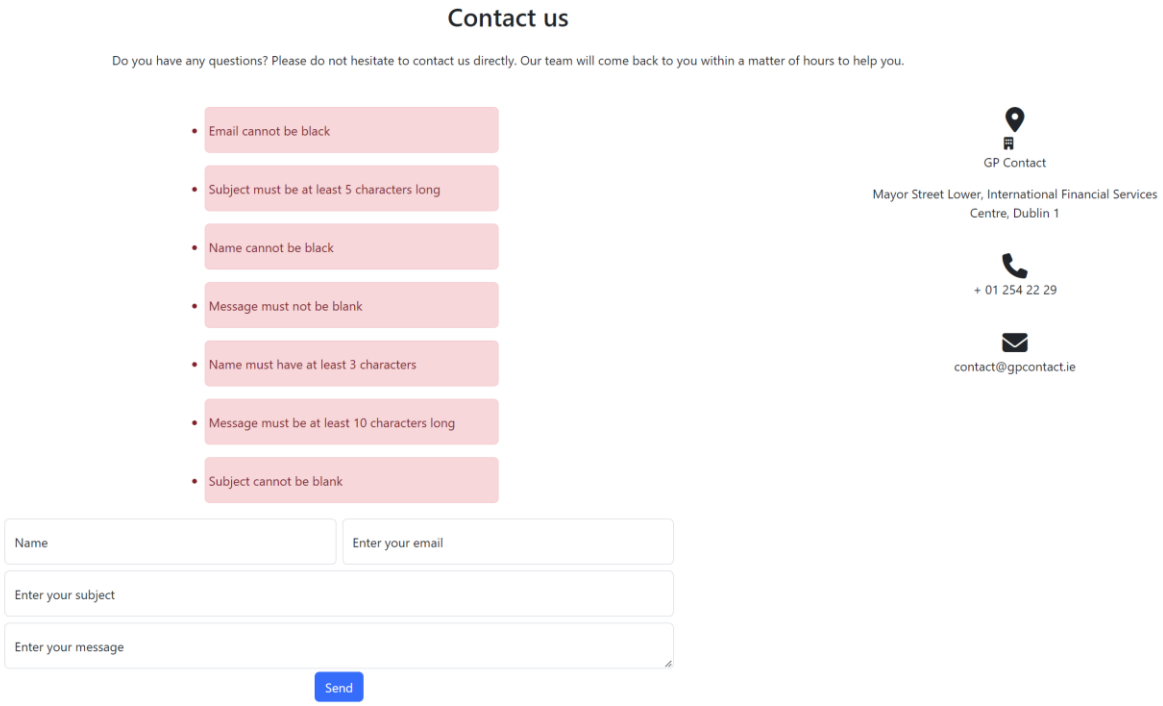
```
<script>
    let changeVideoBtn = document.querySelectorAll('.vid-btn');
    changeVideoBtn.forEach(btn =>{
        btn.addEventListener('click', ()=>{
            document.querySelector('.controls .active').classList.remove('active');
            btn.classList.add('active');
            let src = btn.getAttribute('data-src');
            document.querySelector('#video-slider').src = src;
        });
    });
</script>
```

4.3 Contact page

Any user may submit contact message that will be persisted to the database upon bean successful validation.



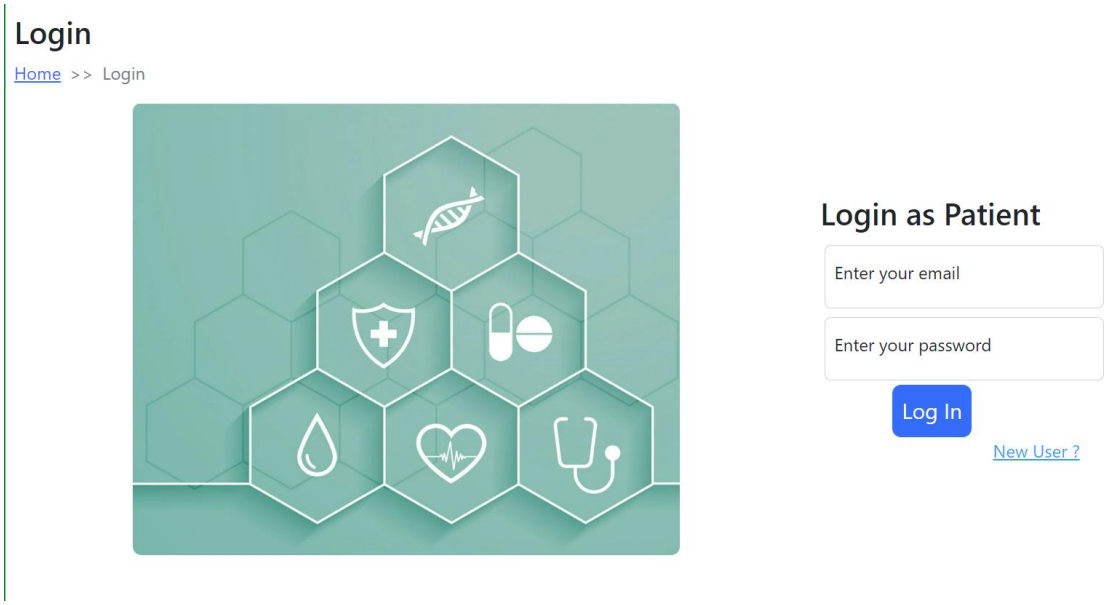
Back end validations:



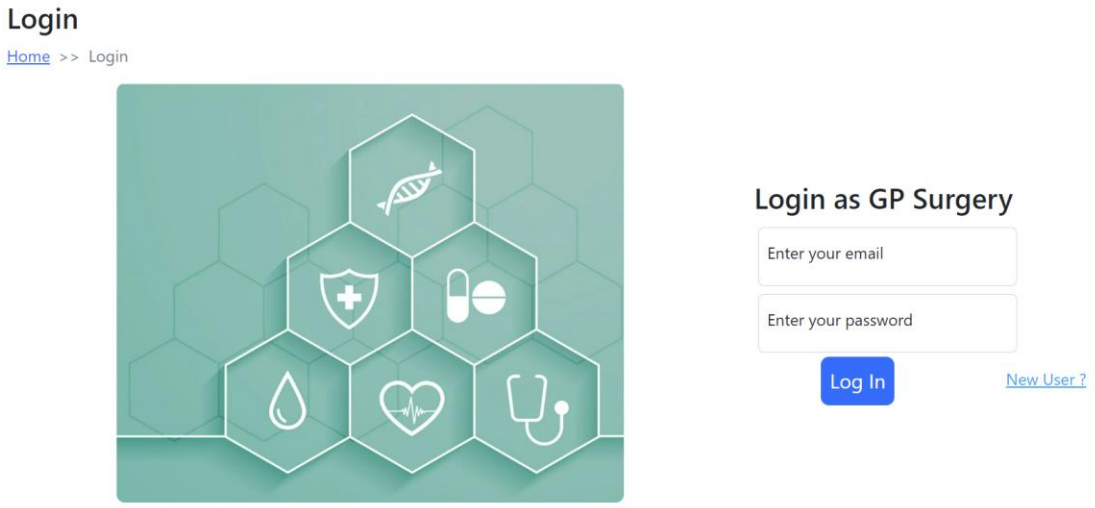
4.4 Log in page

Login page use CSRF protection by generating a token.

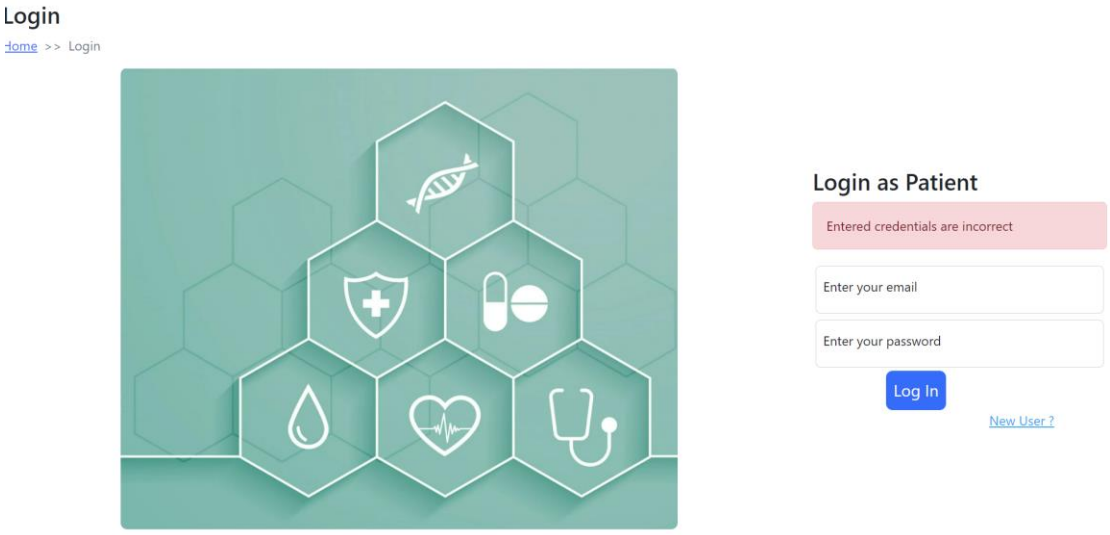
Patient:



GP Surgery



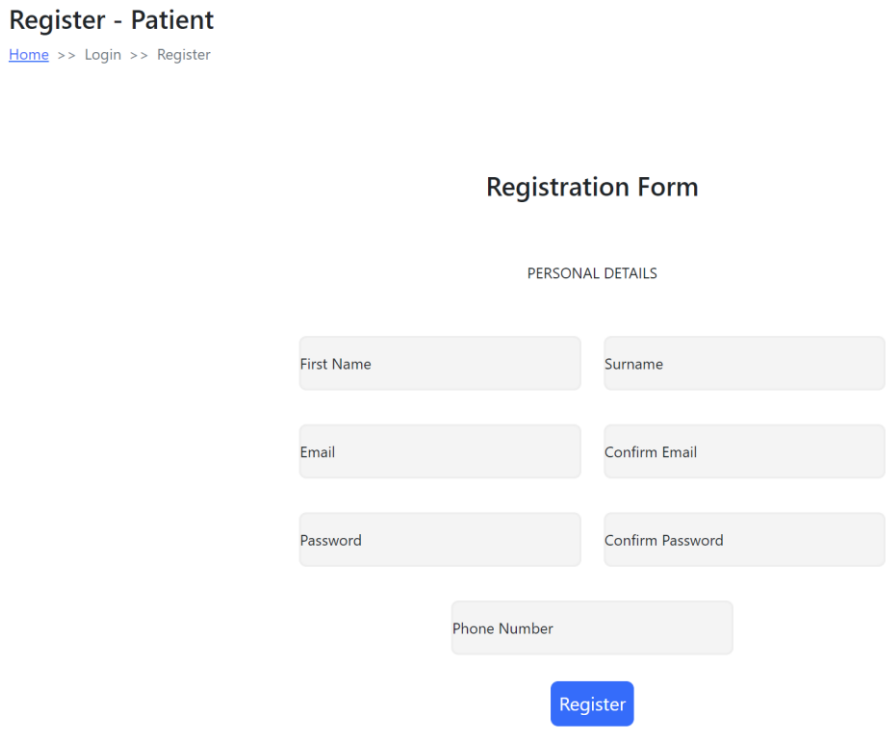
Error page:



4.5 Registration

Any user may register as a patient and data is persisted upon successful bean validation.

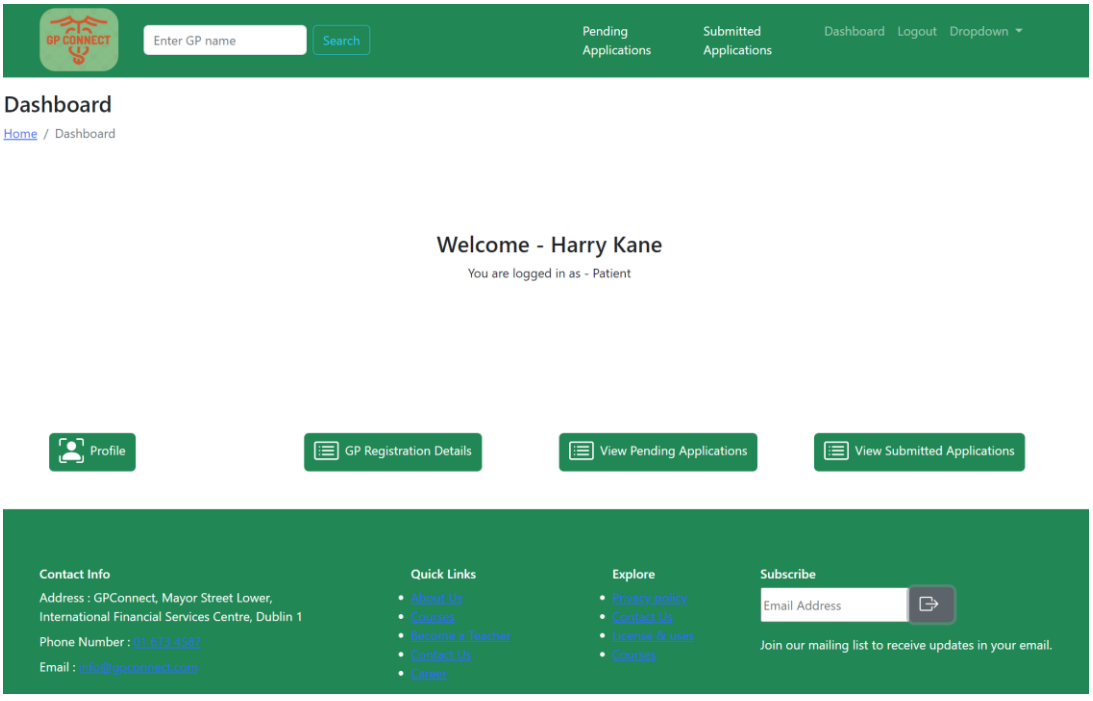
Registration page



4.6 Dashboard

When a user will log in, he/she will be brought to a dashboard where additional use cases are realised such as view Profile, GP Registration Details, Pending Applications, Submitted and Applications. Only logged in user is permitted to access add GP surgery use case and subsequently submit registration form.

Dashboard option is also added to the navigation bar and sign out will replace login/register option.

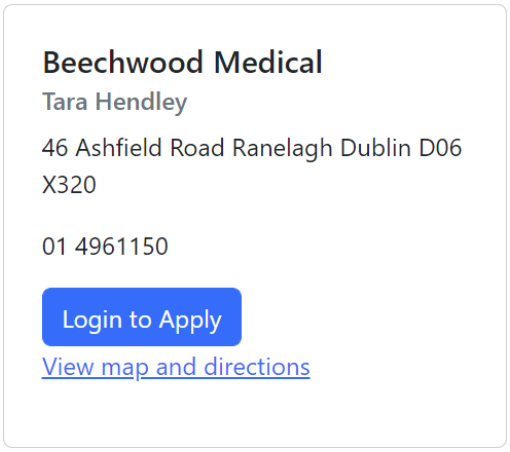


4.6 Search GP surgery

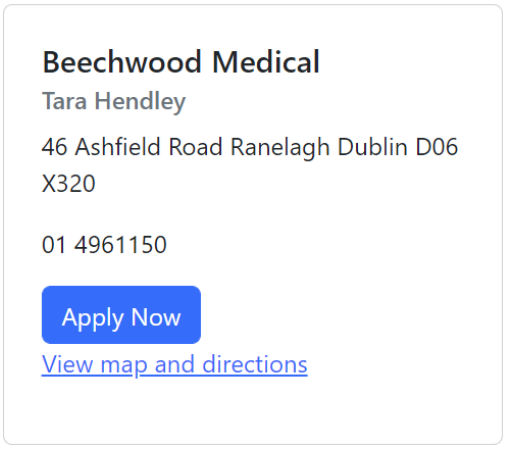
4.6.1 Anonymous user search vs Authorised user search

Only logged in user is permitted to access add GP surgery use case and subsequently submit registration form. One search result below for comparison:

(Anonymous user):

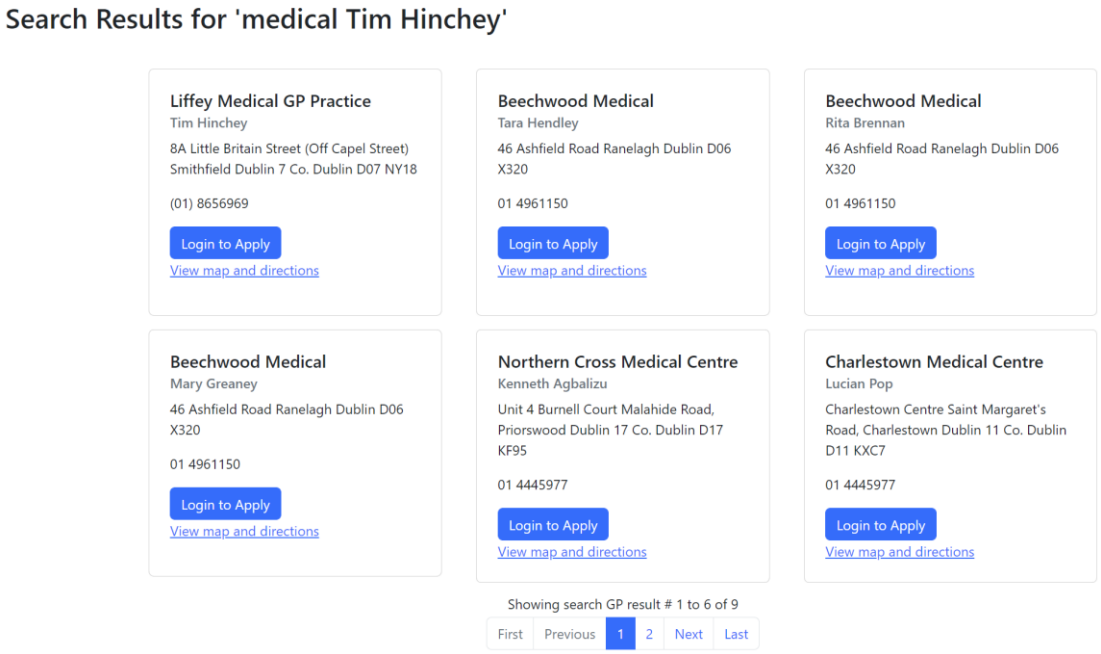


(Patient):



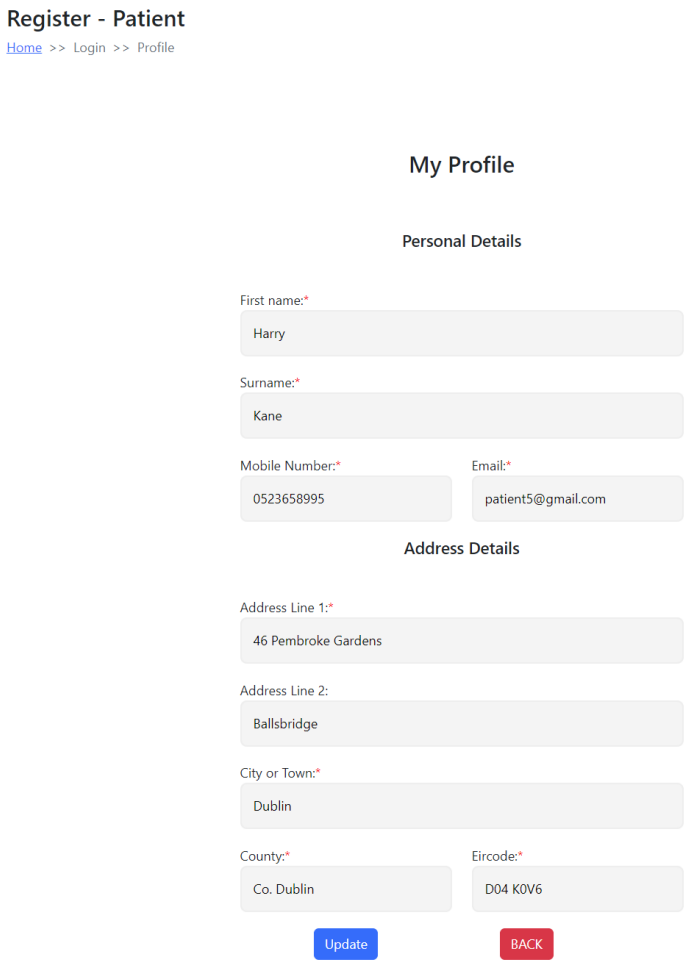
4.6.2 Search Result & Pagination

Once registered, the patient is able to add any number of GP Surgeries that opted to accept new patient on a waiting list, the result is paginated:



4.7 My profile page

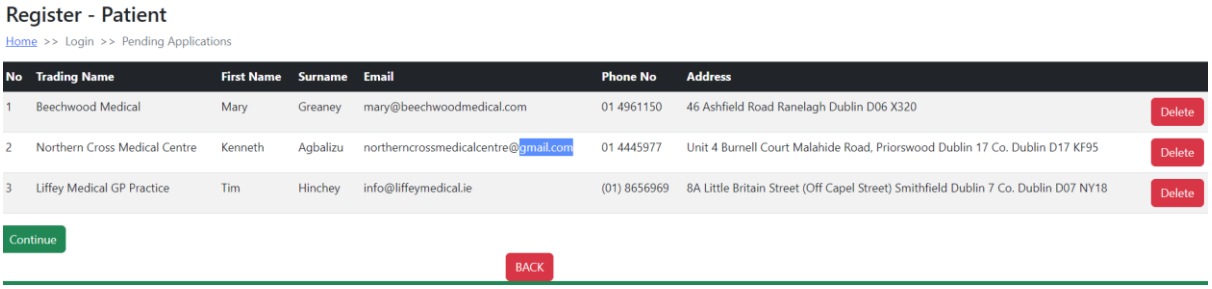
Inserted data is persisted via update and if data already exists, a query is run to populate already existing data to enhance user experience



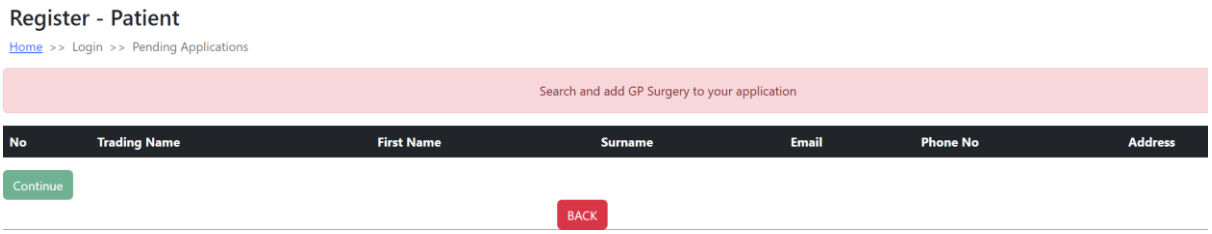
The above mentioned input is valid however, in case of errors in bean validation, the following will appear:

4.8.1 View pending application

The app takes into account a case where a patient has not updated his/her profile via dashboard and instead proceeds to Add GP Surgery to pending applications and proceeds via the following page:



In case there are no pending application user is informed and continue button is disabled:



4.8.2 Continue with pending applications

The patient has an option to remove the pending application or proceed with all displayed applications. The system will prompt the user to confirm or insert his/her details that will be validated prior to persisting to database. Example of Errors below and back-end code that triggers the validation:

- Please insert a valid PPS number
- Please insert a valid date of birth (dd.mm.yyyy)

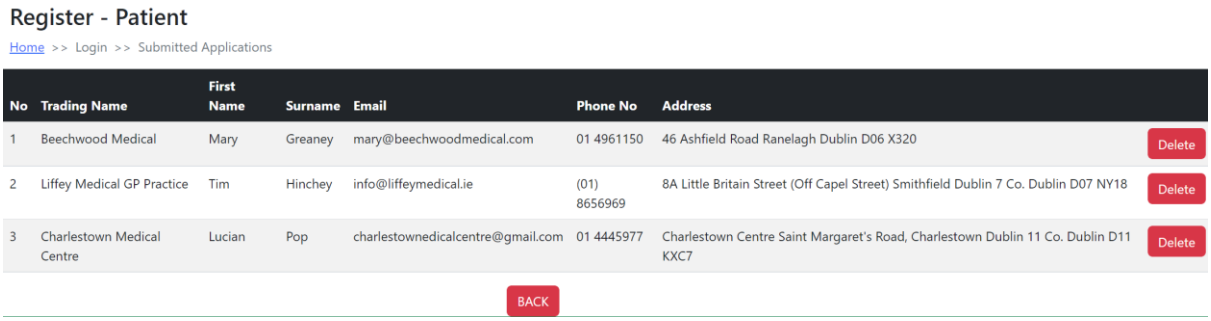
```
@Pattern(regexp = "^(?:0[1-9]|[12]\\d|3[01])([\\/.-])(?:0[1-9]|1[012])\\1(?:19|20)\\d\\d$", message = "Please insert a valid date of birth (dd.mm.yyyy)")
private String dob;

@Pattern(regexp = "\\d{7}[A-Z]{1,2}", message = "Please insert a valid PPS number")
private String ppsn;

private GenderEnum genderEnum;

//Pattern allows for empty string
@Pattern(regexp = "|\\d{1}[A-Z]{1}\\d{5}[A-Z]{1}", message = "Please insert a valid medical card number")
private String medicalCardNo;
```

4.8.3 Successful validation and application submission

After successful validation on all mandatory field, the user can use submit button and proceed:

You have succesfully submitted all you applications

The patient is able to view and delete if necessary the submitted applications:



4.9 View Submissions by GP Surgery

GP is also able to view submitted application ordered by date and delete those application that have been accepted. Applications that were not submitted and are still pending (not submitted by a Patient) are not visible.

5 Conclusion

After completing my first Spring Boot project, I have a good understanding of the following:

- How to use Spring Boot to quickly set up a new project with a variety of useful dependencies, such as Spring MVC, Spring Security, and Spring Data.
- How to use Spring MVC to create controllers, handle HTTP requests, and render templates.
- How to use Spring Data to easily implement a data access layer for my application.
- How to use Spring Security to secure your application with user authentication and authorization including hashing algorithms and CSRF protection.
- How to use Spring Boot's autoconfiguration and starter dependencies to easily set up and configure common application components.

I also got familiar with the Spring framework's dependency injection and aspect-oriented programming features, which are key components of the Spring ecosystem.

Overall, Spring Boot is a powerful and convenient framework for building web applications and microservices in Java. By completing my first project, I should have a strong foundation for building more complex and sophisticated applications in the future.

6 Future Scope of project

The web application could bring more features in the future:

- Password recovery function for registered users
- Remember me option while logging in
- While using the GP surgery search option, the user can use his location
- Miniature map is displayed next to a result of GP surgery to enhance user experience.
- Creating another form to be submitted with the registration form for release of patient's medical records upon successful acceptance by the GP surgery
- Automation of acceptance of the particular patient by GP Surgery according to availability.
- Data scraping from HSE website: https://www2.hse.ie/services/find-a-gp/ that provides contact details for GP surgeries can be used to create non active accounts for GP surgeries and enable the users to indicate whether the GP accepts new patients
- Additional feature of booking a visit with a GP or any medical practitioner can be introduced together with a secure payment
- A streamlined system of automated notification of applied registration form can be implemented in case the patient or GP confirms finalisation and acceptance of registration application

7 References

- https://techwithmaddy.com/spring-boot-architecture
- https://www.w3.org/standards/webdesign/htmlcss
- https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide
- https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5
- https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html
- https://www.gov.ie/en/press-release/6e1f6-ministers-for-health-announce-investment-of-23-4-billion-in-irelands-health-and-social-care-services
- https://www.rte.ie/news/health/2022/0928/1326003-health-briefing/
- https://www2.hse.ie/services/schemes-allowances/medical-cards/about-the-medical-card/gps-who-accept-medical-cards/
- https://www.youtube.com/watch?v=cehTm_oSrqA&ab_channel=Amigoscode
- https://www.w3.org/standards/webdesign/htmlcss
- https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide
- https://springframework.guru/spring-framework-annotations/
- https://www.geeksforgeeks.org/spring-framework-annotations/