Bui Tong Nha

REPORT LEARNING

**Table of Contents**

[I. Programming basic: 2](#_Toc179175102)

[1. Development application using Spring Boot and React: 2](#_Toc179175103)

[a. Architecture application: 2](#_Toc179175104)

[b. Results of the Project 3](#_Toc179175105)

[2. Project Structure 6](#_Toc179175106)

[3. Configure MySQL Database 6](#_Toc179175107)

[4. User Registration and Login 7](#_Toc179175108)

[5. App Patching CURD: 11](#_Toc179175109)

[6. Testing APIs using Postman: 15](#_Toc179175110)

[II. Burp Suite 19](#_Toc179175111)

[III. DevSecOps 21](#_Toc179175112)

[1. DevOps 21](#_Toc179175113)

# Programming basic:

## Development application using Spring Boot and React:

### Architecture application:

Overview: The site back end is development by Spring Boot, Spring Web MVC for REST APIs and Spring Data JPA. Interaction with the MySQL database. On the front end site, the application is built with React, React Router, Axios, and Bootstrap to provide user interface.

A diagram of a computer system

Description automatically generated

Technologies using for the project:

* Backend:
  + Spring Boot.
  + SpringData JPA (Hibernate).
  + Maven.
  + Java JDK 17.
  + Tomcat.
  + MySQL 8 Database.
* Frontend:
  + React.
  + Nodejs.
  + Bootstrap.
  + Axios HTTP Library.

### Results of the Project

**Application Patching User Login and Register:**

* Register a user.

A screenshot of a computer

Description automatically generated

* Login Authentication.

A screenshot of a computer

Description automatically generated

**Application Patching Management CURD:**

* Create patching.

A screenshot of a computer

Description automatically generated

* List all patching.

A screenshot of a computer

Description automatically generated

* Update patching.

A screenshot of a computer

Description automatically generated

* Delete patching.

A screenshot of a computer

Description automatically generated

## Project Structure

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

### Configure MySQL Database

Access the application.properties file and append the following configuration:

A screen shot of a computer

Description automatically generated

### User Registration and Login

User and Role JPA entities and develop a many-to-many mapping between them: one user can have numerous roles, and one role can be given to several users.

The @ManyToMany JPA annotation links the source and target entities.

A many-to-many association always stores the association between two entities in an intermediary join table. The @JoinTable JPA annotation defines the join table.

**Code User.java**

A screen shot of a computer program

Description automatically generated

This above code defines a Java class called User that is mapped to a database table called users. The class has several annotations on it:

**@Getter and @Setter:** These annotations are from the Lombok library and automatically generate getter and setter methods for all of the class's fields.

**@NoArgsConstructor:** This annotation is from Lombok and generates a no-argument constructor for the class.

**@AllArgsConstructor:** This annotation is from Lombok and generates a constructor that takes arguments for all of the class's fields.

**@Entity:** This annotation is from the Java Persistence API (JPA) and specifies that the class is a JPA entity, meaning that it is mapped to a database table.

**@Table(name="users"):** This annotation is from JPA and specifies the name of the database table that the entity is mapped to.

**@Id:** This annotation is from JPA and specifies that the field id is the primary key for the database table.

**@GeneratedValue(strategy = GenerationType.IDENTITY):** This annotation is from JPA and specifies that the primary key values are automatically generated by the database using an identity column.

**@Column(nullable=false):** These annotations are from JPA and specify that the fields name, email, and password cannot be null in the database table.

**@Column(nullable=false, unique=true):** This annotation is from JPA and specifies that the email field must be unique in the database table.

**@ManyToMany(fetch = FetchType.EAGER, cascade=CascadeType.ALL):** This annotation is from JPA and specifies that the relationship between User and Role entities is many-to-many.

**@JoinTable:** This annotation is from JPA and specifies the details of the join table that is used to implement the many-to-many relationship between User and Role entities. It specifies the name of the join table (users\_roles) and the names of the foreign key columns in the join table (USER\_ID and ROLE\_ID).

**Code Role.java**

A screen shot of a computer program

Description automatically generated

This above code defines a Java class called Role that is mapped to a database table called roles. The class has several annotations on it:

**@Getter and @Setter:** These annotations are from the Lombok library and automatically generate getter and setter methods for all of the class's fields.

**@NoArgsConstructor:** This annotation is from Lombok and generates a no-argument constructor for the class.

**@AllArgsConstructor:** This annotation is from Lombok and generates a constructor that takes arguments for all of the class's fields.

**@Entity:** This annotation is from the Java Persistence API (JPA) and specifies that the class is a JPA entity, meaning that it is mapped to a database table.

**@Table(name="users"):** This annotation is from JPA and specifies the name of the database table that the entity is mapped to.

**@Id:** This annotation is from JPA and specifies that the field id is the primary key for the database table.

**@GeneratedValue(strategy = GenerationType.IDENTITY):** This annotation is from JPA and specifies that the primary key values are automatically generated by the database using an identity column.

**Code UserRepository.java and RoleRepository.java**

A black background with white text

Description automatically generated

The above code defines a Spring Data JPA repository interface called RoleRepository, which extends the JpaRepository interface. The JpaRepository interface provides several methods for performing CRUD (Create, Read, Update, Delete) operations on a JPA entity. It takes two type parameters: the entity type, Role, and the type of the entity's primary key, Long.

In addition to the methods provided by JpaRepository, the RoleRepository interface also declares a custom method called findByName(). This method uses Spring Data JPA's method name query creation feature to generate a query that finds a role by its name. The method takes a single argument, which is the role name to search for, and it returns a Role object if a match is found or null otherwise.

A screen shot of a computer program

Description automatically generated

The above code defines a Spring Data JPA repository interface called UserRepository, which extends the JpaRepository interface. The JpaRepository interface provides several methods for performing CRUD (Create, Read, Update, Delete) operations on a JPA entity. It takes two type parameters: the entity type, User, and the type of the entity's primary key, Long.

In addition to the methods provided by JpaRepository, the UserRepository interface also declares a custom method called findByEmail(). This method uses Spring Data JPA's method name query creation feature to generate a query that finds a user by their email address. The method takes a single argument, which is the email address to search for, and it returns a User object if a match is found or null otherwise.

**Code Controller Layer**

A computer screen shot of a program code

Description automatically generated

REST controller in a Spring Boot application for handling user authentication, specifically for registering new users and logging in existing users.

**@CrossOrigin("\*"):** Allows Cross-Origin Resource Sharing (CORS) from any origin. This means the API endpoints in this controller can be accessed from any domain. The "\*" allows requests from all origins.

**@AllArgsConstructor:** Automatically generates a constructor for the class that includes all class fields (in this case, AuthService). It eliminates the need to write a constructor manually.

**@RestController:** Marks this class as a Spring REST controller, meaning it will handle HTTP requests and send responses in JSON or XML format.

**@RequestMapping("/api/auth"):** Defines the base URL path for all APIs in this controller. All the endpoints inside this class will start with /api/auth.

This class serves as a controller to handle HTTP requests related to user authentication.

AuthService authService: This is a dependency that handles the business logic of authentication (registering and logging in users). It is injected into the controller via the constructor thanks to @AllArgsConstructor.

**Register API:**

**@PostMapping("/register"):** Defines an HTTP POST endpoint at /api/auth/register for user registration.

**Login API:**

**@PostMapping("/login"):** Defines an HTTP POST endpoint at /api/auth/login for user login.

## App Patching CURD:

Creating the backend application using Spring Boot.

**Code patching controller PatchingController.java**

Spring Boot REST controller for handling operations related to a "patching" resource. The controller uses various HTTP methods to manage patching data, including creating, reading, updating, and deleting (CRUD) operations.

**@CrossOrigin("\*"):** Allows CORS (Cross-Origin Resource Sharing) from any origin, enabling requests from different domains to access the API.

**@RestController:** Marks the class as a RESTful controller, indicating that it handles HTTP requests and sends responses in JSON format.

**@RequestMapping("api/patching"):** Specifies the base URL path for all endpoints in this controller. All endpoints will start with /api/patching.

**@AllArgsConstructor:** Automatically generates a constructor for the class that accepts all the class fields (in this case, PatchingService).

**A screen shot of a computer program

Description automatically generated**

**Add Patching:**

**@PreAuthorize("hasRole('ADMIN')"):** Restricts access to this endpoint to users with the role "ADMIN".

**@PostMapping:** Defines a POST endpoint for creating a new patching resource.

**@RequestBody PatchingDto patchingDto:** Accepts the patching data from the request body and maps it to the PatchingDto object.

**Response:** Returns the created PatchingDto with a CREATED (201) status.

**Get Patching:**

**@GetMapping("{id}"):** Defines a GET endpoint for retrieving a specific patching resource by its ID.

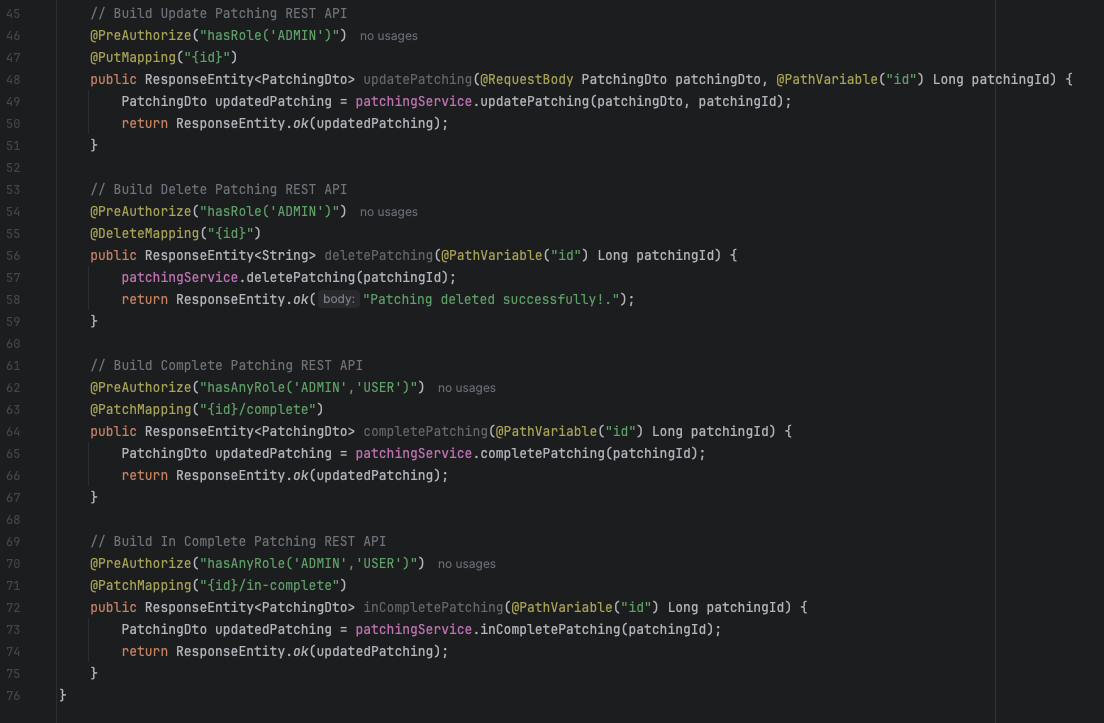
**@PathVariable("id") Long patchingId:** Extracts the id from the URL and maps it to the patchingId variable.

**Response:** Returns the retrieved PatchingDto with an OK (200) status.

**Get All Patching:**

**@GetMapping:** Defines a GET endpoint for retrieving all patching resources.

**Response:** Returns a list of PatchingDto objects with an OK (200) status.



**Update Patching:**

**@PutMapping("{id}"):** Defines a PUT endpoint for updating an existing patching resource by its ID.

**Response:** Returns the updated PatchingDto with an OK (200) status.

**Delete Patching:**

**@DeleteMapping("{id}"):** Defines a DELETE endpoint for removing a patching resource by its ID.

**Response:** Returns a success message with an OK (200) status.

**Complete Patching:**

**@PatchMapping("{id}/complete"):** Defines a PATCH endpoint for marking a patching resource as complete.

**Response:** Returns the updated PatchingDto with an OK (200) status.

**In Complete Patching:**

**@PatchMapping("{id}/in-complete"):** Defines a PATCH endpoint for marking a patching resource as incomplete.

**Response**: Returns the updated PatchingDto with an OK (200) status.

**Code DTO PatchingDto.java**

A screen shot of a computer program

Description automatically generated

**Data Transfer Object (DTO)** class named PatchingDto.java. DTOs are commonly used in applications to transfer data between different layers, such as between the controller and service layers.

private Long id: This field represents the unique identifier for a patching resource. It is of type Long, which is commonly used for IDs in Java.

private String title: This field stores the title of the patching resource. It is of type String.

private String description: This field holds a description of the patching resource, also of type String.

private boolean completed: This boolean field indicates whether the patching resource is completed or not.

**Code Entity Patching.java**

Entity code snippet defines a JPA (Java Persistence API) entity class named Patching. This class represents a table in a relational database, allowing you to map Java objects to database records.

A screen shot of a computer program

Description automatically generated

**@Id:** Marks this field as the primary key of the entity.

**@GeneratedValue(strategy = GenerationType.IDENTITY):** Specifies that the value of the primary key should be generated automatically by the database using the identity column strategy. This means the database will handle the incrementing of the ID value.

**@Column(nullable = false):** This annotation specifies that the title and description fields cannot be null in the database. It adds a NOT NULL constraint on these columns.

**private String title:** This field represents the title of the patching resource.

**private String description:** This field holds a description of the patching resource.

**private boolean completed:** This boolean field indicates whether the patching resource is completed. It does not have any constraints defined in the database, meaning it can be null unless you specify otherwise.

## Testing APIs using Postman:

* Testing the user registration API:

A screenshot of a computer

Description automatically generated

* Testing the user login API:

A screenshot of a computer

Description automatically generated

* Testing the add patching API:

A screenshot of a computer

Description automatically generated

* Testing the API to list all patching:

A screenshot of a computer

Description automatically generated

* Testing the update patching API:

A screenshot of a computer

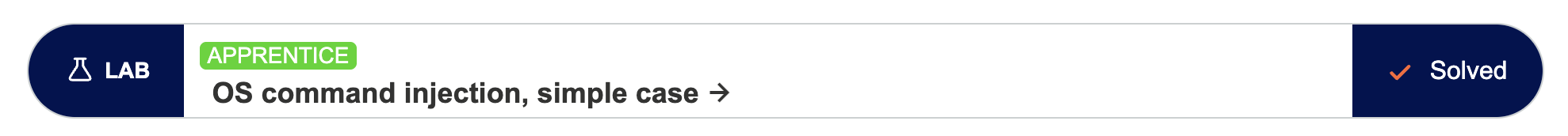
Description automatically generated

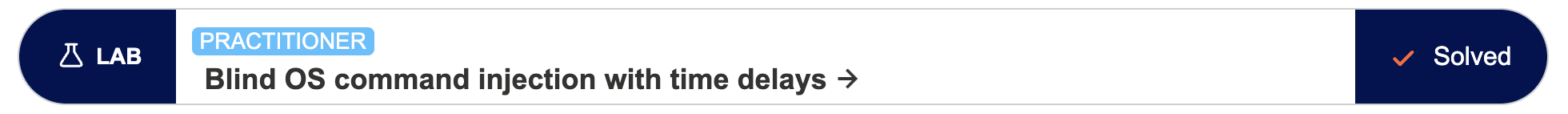
* Testing the delete patching API:

A screenshot of a computer

Description automatically generated

# Burp Suite

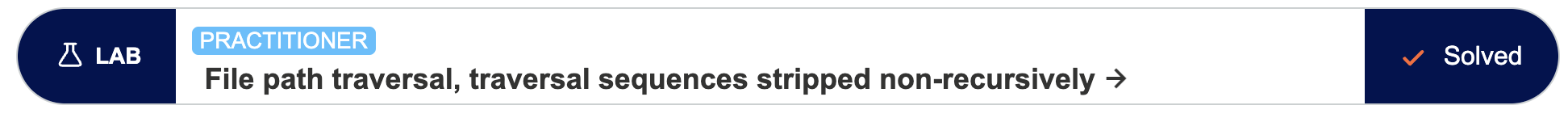


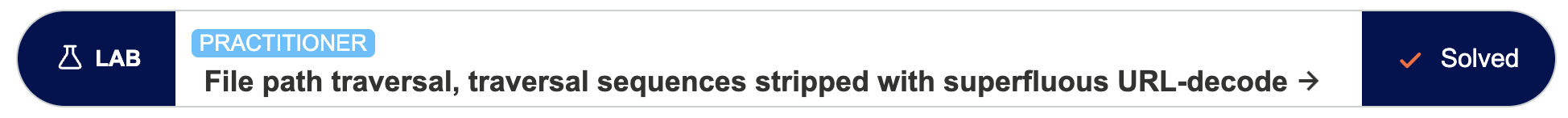










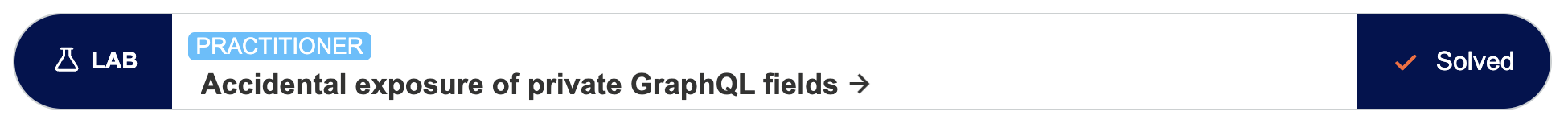


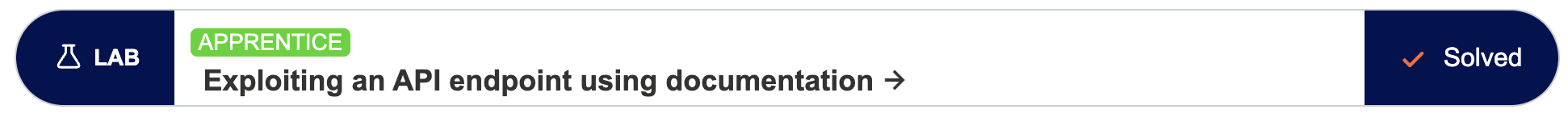












# DevSecOps

## DevOps

* docker-compose Backend for Spring Boot and MySQL:

A screenshot of a computer program

Description automatically generated

* docker-compose Frontend for React:

A screen shot of a computer

Description automatically generated