



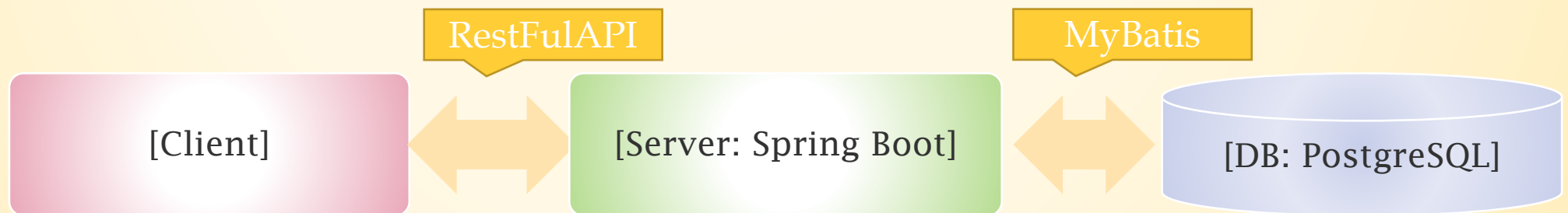
XÂY DỰNG BACKEND VỚI SPRING BOOT VÀ MYBATIS

Trình bày: Bùi Bá Nguyên

Ryomo Vietnam Solutions Co. Ltd

Giới thiệu

- **Mục tiêu:** Tạo API CRUD, kết nối PostgreSQL với MyBatis, kiểm tra API bằng Postman.
- **Vai trò:**
 - Spring Boot: Framework Java, xây dựng API REST.
 - Database: PostgreSQL hệ quản trị CSDL.
- **Công cụ:** Eclipse IDE, Postman, PostgreSQL.



Xây dựng Backend – SpringBoot

- **SpringBoot:**

- Là một module của Spring Framework, được thiết kế để đơn giản hóa quá trình phát triển ứng dụng Java dựa trên Spring

- **Vai trò:**

- Giảm thiểu cấu hình
- Phát triển ứng dụng nhanh (RAD - Rapid Application Development)
- Tạo ứng dụng độc lập

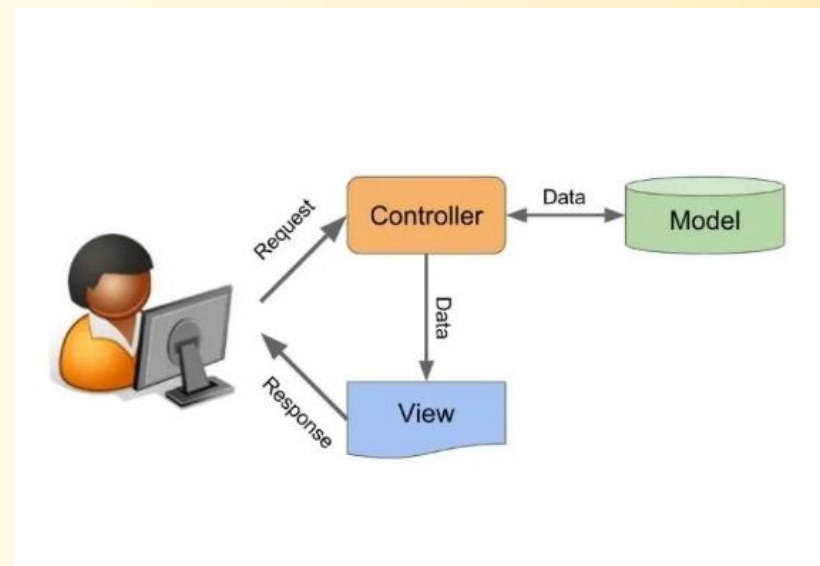
Xây dựng Backend – Mô hình MVC

- **Mô hình MVC (Model-View-Controller) :**

- Là một mô hình kiến trúc phần mềm phổ biến, giúp tổ chức mã nguồn ứng dụng một cách rõ ràng và dễ bảo trì. Nó chia ứng dụng thành ba thành phần chính có vai trò riêng biệt, giúp tách biệt logic nghiệp vụ, giao diện người dùng và xử lý tương tác.

- **Các thành phần của mô hình MVC**

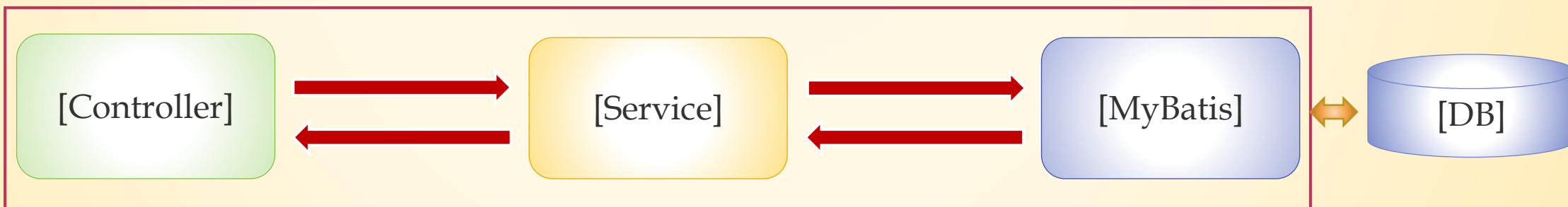
- **Model:** đại diện cho dữ liệu và Logic nghiệp vụ
- **View:** giao diện người dùng (UI)
- **Controller:** đại diện cho logic điều khiển luồng ứng dụng và điều phối luồng dữ liệu giữa Model và View



Xây dựng Backend – Mô hình MVC

- **Mô hình MVC dựa trên SpringBoot:**

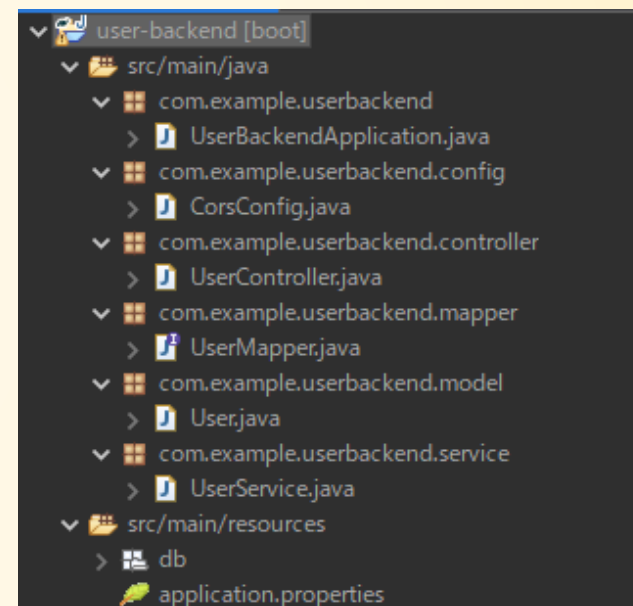
- [Model]: Đây là các object để “mang” dữ liệu từ Controller đến [View].(User.java).
- View:
 - **Spring Boot không trực tiếp tạo ra "View" theo kiểu HTML truyền thống mà sẽ phục vụ dưới dạng RESTful APIs (JSON).**
 - **Client (React) sẽ nhận dữ liệu JSON này và tự xây dựng giao diện người dùng.**
- [Controller]: Xử lý request (UserController.java).
- [Service]: Logic nghiệp vụ (UserService.java).



Xây dựng Backend – Mô hình MVC

- **Cấu trúc thư mục:**

- `aots.rvsc.user.backend`:
 - `model`: folder chứa các object ánh xạ DB.
 - `mapper`: chứa các file XXXMapper.java chứa các method CRUD để service gọi tới.
 - `service`: chứa các file service - làm nhiệm vụ trung gian để controller có thể gọi được mapper
 - `controller`: chứa các file logic điều khiển
- `resources`: chứa các file thiết lập, properties, report...

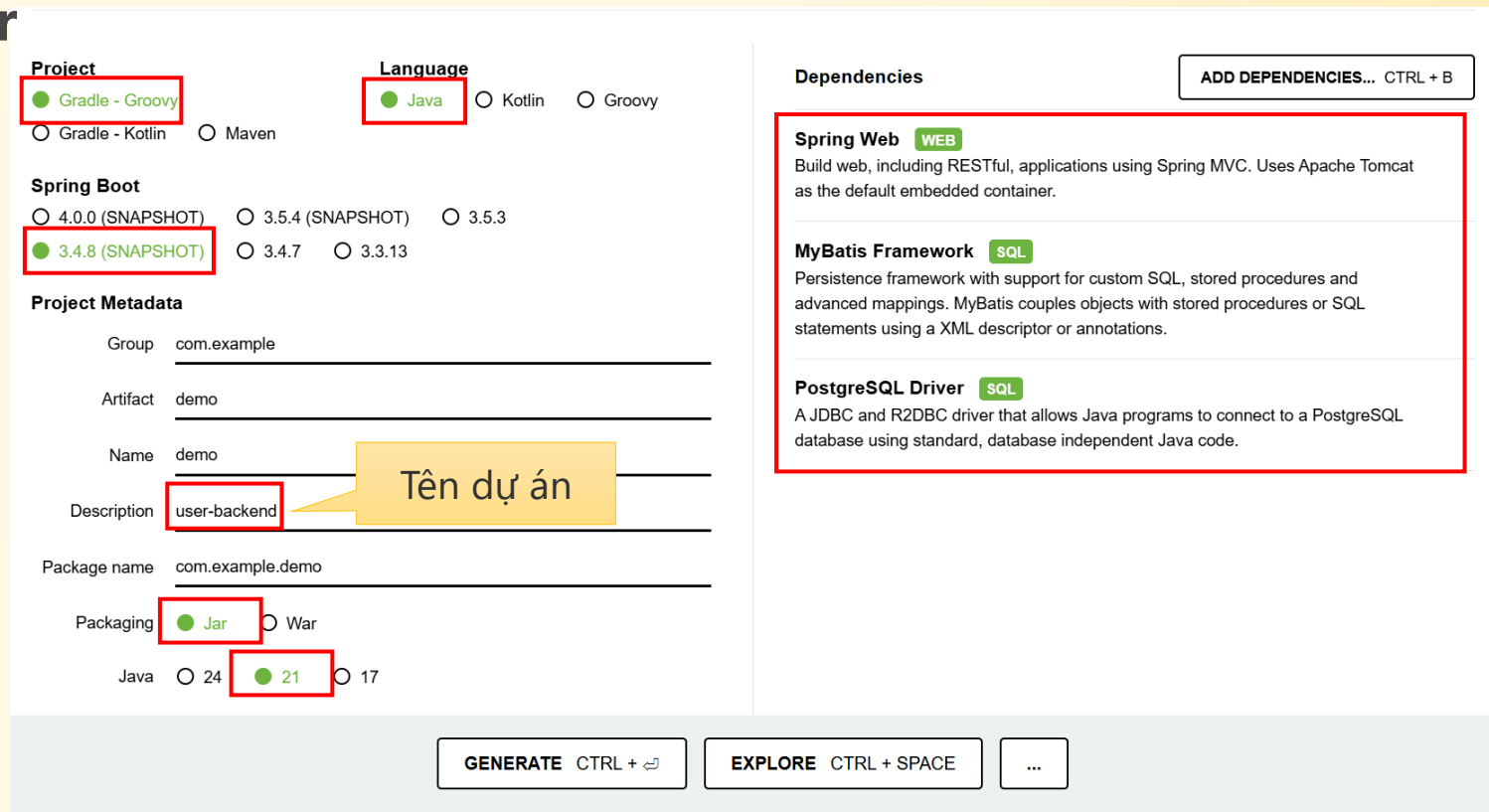


Khởi tạo dự án

▪ Sử dụng Spring Initializr

<https://start.spring.io>

- Dependencies:
 - Spring Web,
 - MyBatis Framework,
 - PostgreSQL Driver.
- Build Tool: **Gradle**.
- Tên dự án: [user-backend].
- Packaging: **jar**



The screenshot shows the Spring Initializr form with several fields highlighted by red boxes and a yellow callout box:

- Project:** ☒ Gradle - Groovy
- Language:** ☒ Java
- Spring Boot:** ☒ 3.4.8 (SNAPSHOT)
- Project Metadata:**
 - Group: com.example
 - Artifact: demo
 - Name: demo
 - Description: user-backend (highlighted by a yellow callout box labeled "Tên dự án")
 - Package name: com.example.demo
- Packaging:** ☒ Jar
- Java:** ☒ 21
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - MyBatis Framework** (SQL): Persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations.
 - PostgreSQL Driver** (SQL): A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Buttons at the bottom: GENERATE CTRL + G, EXPLORE CTRL + SPACE, ...

Khởi tạo dự án

- **Import vào Eclipse:**

- File > Import > Gradle > Existing Gradle Project.

- **Cấu hình application.properties**

- **Datasource.url:** khai báo url của Database
- **Datasource.username:** khai báo username kết nối Database
- **Datasource.password:** khai báo password kết nối Database

```
spring.datasource.url=jdbc:postgresql://localhost:5432/user_db  
spring.datasource.username=postgres  
spring.datasource.password=your_password
```

- **Chạy dự án**

- Right-click project > Run As > Spring Boot App

Tạo Bảng

▪ Tạo bảng users trong PostgreSQL

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    active BOOLEAN NOT NULL
);
```

Data Output
Messages
Notifications

CREATE TABLE

Query returned successfully in 129 msec.

users		id	name	email	active
Columns (4)		[PK] integer	character varying (100)	character varying (100)	boolean
id					
name					
email					
active					

Tạo Model - Object

- Tạo object đại diện cho dữ liệu của bảng user đã tạo **User.java** (aots.rvsc.user.backend.model)

```
public class User {  
    private Long id;  
    private String name;  
    private String email;  
    private boolean active;  
    // Getters and setters  
    public Long getId() { return id; }  
    public void setId(Long id) { this.id = id; }  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
    public String getEmail() { return email; }  
    public void setEmail(String email) { this.email = email; }  
    public boolean isActive() { return active; }  
    public void setActive(boolean active) { this.active = active; }  
}
```

Các phương thức
get/set

Tạo Model - Mapper và Service

- Tạo UserMapper.java (aots.rvsc.user.backend.mapper):

```
@Mapper
public interface UserMapper {
    @Select("SELECT * FROM users ORDER BY id")
    List<User> findAll();
    @Update("INSERT INTO users (name, email, active) values ({user.name}, {user.email}, {user.active})")
    int create(@Param("user") User params);
    @Update("UPDATE users SET active = {user.active} WHERE id = {user.id}")
    int update(@Param("user") User params);
    @Update("DELETE FROM users WHERE id = {user.id}")
    int delete(@Param("user") User params);
}
```

- @Select: được sử dụng để **định nghĩa các câu lệnh SQL truy vấn dữ liệu** (SELECT statements)
- @Update: được sử dụng để **định nghĩa các câu lệnh SQL thay đổi dữ liệu** (INSERT, UPDATE, DELETE statements)

Tạo Model - Mapper và Service

- Tạo UserService.java (aots.rvsc.user.backend.service)

```
@Service
public class UserService {
    private final UserMapper userMapper;
    public UserService(UserMapper userMapper) {
        this.userMapper = userMapper;
    }
    public List<User> getAllUsers() {
        return userMapper.findAll();
    }
    public int create(User user) {
        return userMapper.create(user);
    }
    public int update(User user) {
        return userMapper.update(user);
    }
    public int delete(User user) {
        return userMapper.delete(user);
    }
}
```

- `getAllUsers` : lấy toàn bộ danh sách user
- `create` : tạo user
- `update` : cập nhật thông tin user
- `delete` : xóa user

Tạo Controller

- Tạo UserController.java (aots.rvsc.user.backend.controller):

```
@RestController
@Transactional
@RequestMapping("/api/users")
public class UserController {
    @Autowired
    private UserService userService;
    @GetMapping("get-all")
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }
    @PostMapping("create")
    public Boolean create(@RequestBody User params) {
        userService.create(params);
        return true;
    }
    @PostMapping("update")
    public Boolean update(@RequestBody User params) {
        userService.update(params);
        return true;
    }
    @PostMapping("delete")
    public Boolean delete(@RequestBody User params) {
        userService.delete(params);
        return true;
    }
}
```

- @RestController: được sử dụng để xây dựng các **RESTful web service**
- @Transactional: được sử dụng để **quản lý giao dịch (transaction)** trong các ứng dụng Spring.
- @RequestMapping: được sử dụng để **ánh xạ các yêu cầu HTTP đến các phương thức xử lý (handler methods)** trong controller
- @Autowired: được sử dụng để thực hiện **dependency injection (DI)** trong Spring
- @GetMapping: được sử dụng để ánh xạ các yêu cầu HTTP **GET** đến các phương thức xử lý cụ thể
- @PostMapping: được sử dụng để ánh xạ các yêu cầu HTTP **POST** đến các phương thức xử lý cụ thể

GET/POST

- **GET:**

- **Mục đích:** Lấy dữ liệu từ server.

- **Đặc điểm**

- Dữ liệu được gửi qua URL (trong chuỗi query string, ví dụ: ? key=value).
- Không bảo mật vì dữ liệu hiển thị trên URL.
- Thích hợp cho các yêu cầu không thay đổi trạng thái server (idempotent).
- Giới hạn kích thước dữ liệu (tùy thuộc vào trình duyệt và server).
- Có thể được bookmark hoặc lưu trong lịch sử trình duyệt.

- **Ví dụ:** GET /api/users → Trả về JSON danh sách.

GET/POST

- **POST:**

- **Mục đích:** Gửi dữ liệu đến server để xử lý (ví dụ: gửi form, upload file).

- **Đặc điểm**

- Dữ liệu được gửi trong body của request, không hiển thị trên URL.
- Bảo mật hơn GET (nhưng vẫn cần HTTPS để mã hóa).
- Không bị giới hạn kích thước dữ liệu (tùy thuộc vào cấu hình server).
- Thích hợp cho các yêu cầu thay đổi trạng thái server (non-idempotent).

- **Ví dụ:** POST /api/users với body:

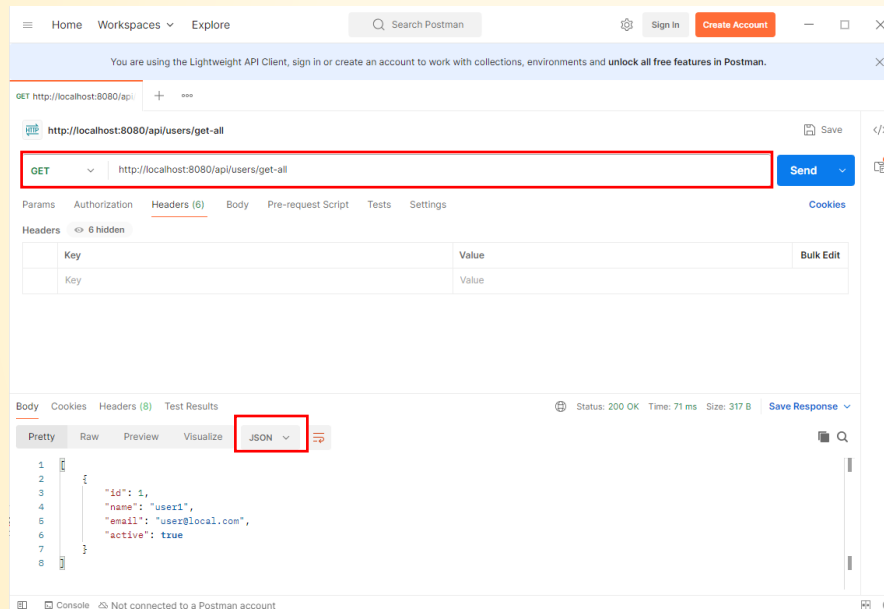
```
{"name": "Trần Thị B", "email": "b@example.com", "active": false}
```

- **Lưu ý:**

- GET an toàn, POST thay đổi dữ liệu server.
- Kiểm tra response code: 200 (OK), 201 (Created).

Kiểm tra API

- Kiểm tra API với Postman:
 - Phương thức GET

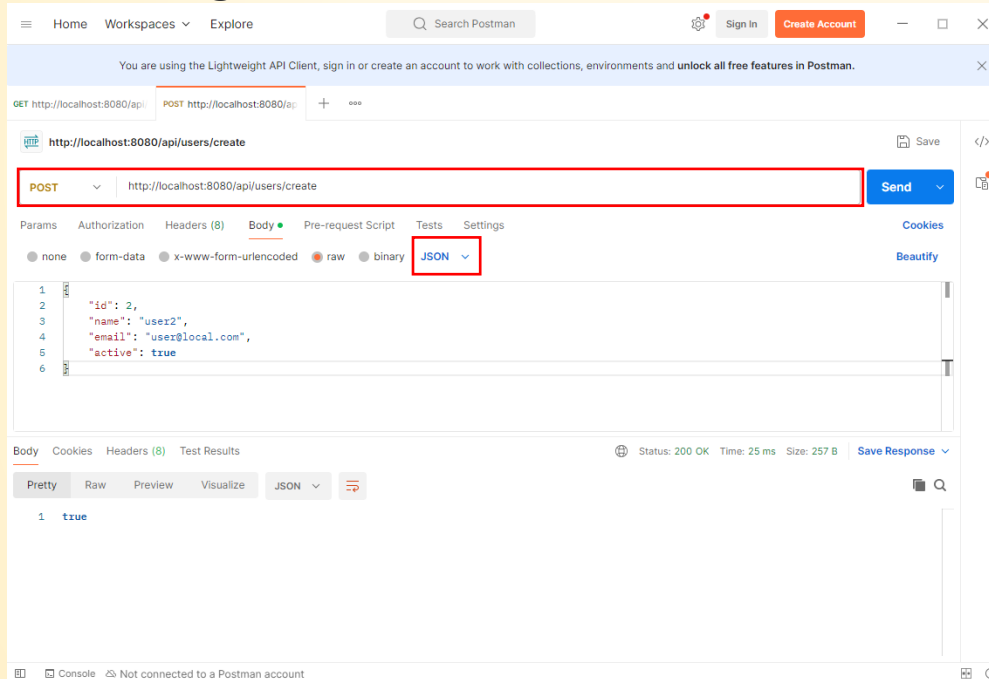


```
{"name": "user1", "email": "user@local.com", "active": true}
```

- Khởi động Postman
- Tạo Request dạng GET
- Nhập URL của API.
 - VD: `http://localhost:8080/api/users/get-all`

Kiểm tra API

- **Kiểm tra API với Postman:**
 - Phương thức POST



- Khởi động Postman
- Tạo Request dạng POST
- Nhập URL của API.
 - VD : `http://localhost:8080/api/users/create`
- Thêm Query Parameters
 - Chọn thẻ Body
 - Chọn raw
 - Chọn JSON
 - Nhập giá trị parameter

Các lỗi thường gặp

- Khai báo sai application.properties
- Nhập sai user/pass kết nối Database
- Khai báo sai dependency

Thực hành – Kiểm tra API với Postman

- **Mục tiêu:**

- Chạy dự án trong Eclipse (Run As > Spring Boot App)
- Dùng Postman:
 - GET: Lấy danh sách người dùng.
 - POST: Thêm 2 người dùng mới.
- Kiểm tra dữ liệu trong pgAdmin (bảng users).

- **Debug lỗi:**

- Kiểm tra application.properties (URL, username, password).
- Đảm bảo annotations trong UserMapper.java đúng cú pháp.

Tổng kết

- **Tổng kết:**

- Tạo dự án Spring Boot, API CRUD.
- Sử dụng MyBatis Mapper với annotations và Service Layer.
- Kiểm tra API với Postman (GET, POST).

- **Bài tập:**

- Thêm API tìm kiếm theo điều kiện:
- Endpoint: GET /api/users?name={keyword}.
- Ví dụ: Tìm người dùng có tên chứa "Nguyễn".
- Kiểm tra API mới với Postman.

Q&A

