# **REST API**

**REST API** (viết tắt của *Representational State Transfer* - *Application Programming Interface*) là một dạng giao tiếp giữa các hệ thống qua mạng, dựa trên kiến trúc REST. REST API cho phép ứng dụng này (như ứng dụng web, ứng dụng di động) có thể giao tiếp và trao đổi dữ liệu với một hệ thống khác (thường là máy chủ) bằng cách sử dụng giao thức HTTP.

**REST API hoạt động như thế nào?**

REST API sử dụng các phương thức của HTTP để thực hiện các thao tác với dữ liệu. Các thao tác này bao gồm:

1. **GET**: Lấy dữ liệu từ máy chủ (ví dụ, lấy thông tin một danh sách người dùng).
2. **POST**: Gửi dữ liệu mới đến máy chủ để tạo mới (ví dụ, thêm người dùng mới).
3. **PUT**: Gửi dữ liệu để cập nhật thông tin (ví dụ, thay đổi thông tin người dùng).
4. **DELETE**: Xóa dữ liệu trên máy chủ (ví dụ, xóa một người dùng).

**REST API dùng để làm gì?**

REST API thường dùng để **kết nối** các hệ thống khác nhau lại với nhau. Ví dụ:

* **Ứng dụng di động và máy chủ**: Ứng dụng di động có thể sử dụng REST API để lấy thông tin người dùng, bài viết, sản phẩm từ máy chủ và hiển thị trên điện thoại.
* **Website và máy chủ**: Một trang web có thể dùng REST API để lấy dữ liệu, hiển thị cho người dùng, hoặc gửi các thông tin mới đến máy chủ.

**Ví dụ đơn giản về REST API**

Giả sử có một hệ thống quản lý người dùng và bạn muốn sử dụng REST API để thao tác với dữ liệu người dùng.

* **Lấy danh sách người dùng**: GET /users
* **Lấy thông tin người dùng theo ID**: GET /users/{id}
* **Thêm người dùng mới**: POST /users
* **Cập nhật thông tin người dùng**: PUT /users/{id}
* **Xóa người dùng**: DELETE /users/{id}

Trong ví dụ này:

* /users là **endpoint** đại diện cho tài nguyên người dùng.
* {id} là một **biến động** để chỉ định người dùng cụ thể.

REST API cho phép bạn thực hiện các thao tác CRUD (*Create, Read, Update, Delete*) trên tài nguyên thông qua các phương thức HTTP tiêu chuẩn này, giúp các hệ thống dễ dàng trao đổi dữ liệu với nhau.

# **POJO**

# **JDBC (Java Database Connectivity):**

**JDBC** là API chuẩn của Java dùng để kết nối và tương tác với cơ sở dữ liệu. Nó cung cấp một cách tiếp cận thủ công hơn trong việc truy vấn và thao tác dữ liệu từ cơ sở dữ liệu.

**Tầng trong project sử dụng JDBC:**

* **Model**: Đại diện cho các đối tượng trong cơ sở dữ liệu (có thể là các lớp POJO - Plain Old Java Object). Mỗi đối tượng sẽ được ánh xạ thủ công với một bảng trong cơ sở dữ liệu.
* **DAO (Data Access Object)**:
  + **DAO Interface**: Định nghĩa các phương thức truy vấn cơ sở dữ liệu, chẳng hạn như getById(), save(), delete(), update(), v.v.
  + **DAOImpl**: Là lớp thực thi các phương thức trong **DAO** Interface, nơi mà bạn sẽ sử dụng **JDBC API** để kết nối cơ sở dữ liệu và thực thi các câu lệnh SQL (như SELECT, INSERT, UPDATE, DELETE).
* **Service**: Tầng này chứa các logic nghiệp vụ, gọi các phương thức từ **DAO** để thao tác với dữ liệu.
* **Controller**: Nhận yêu cầu từ người dùng, gọi các phương thức từ **Service** để xử lý dữ liệu và trả kết quả về cho người dùng.

Trong mô hình này, bạn phải tự quản lý kết nối cơ sở dữ liệu, tạo các câu lệnh SQL, ánh xạ kết quả từ cơ sở dữ liệu vào đối tượng Java (thủ công).

# **JPA (Java Persistence API):**

**JPA** là một API chuẩn của Java dành cho việc ánh xạ đối tượng Java vào cơ sở dữ liệu theo hướng đối tượng (Object-Relational Mapping - ORM). JPA giúp tự động hóa nhiều công việc mà bạn phải làm thủ công khi sử dụng JDBC, như việc quản lý kết nối, tạo câu lệnh SQL, ánh xạ đối tượng và bảng, v.v.

**Tầng trong project sử dụng JPA:**

* **Model (Entity)**: Các lớp **Entity** đại diện cho các đối tượng trong cơ sở dữ liệu. JPA sẽ tự động ánh xạ các lớp **Entity** với các bảng trong cơ sở dữ liệu thông qua các chú thích (@Entity, @Table, @Column, v.v.). Không cần phải viết câu lệnh SQL thủ công, chỉ cần khai báo các thuộc tính trong lớp Entity, và JPA sẽ tự động ánh xạ với cơ sở dữ liệu.
* **Repository**: Tầng này tương tự như **DAO**, nhưng thay vì phải viết các câu lệnh SQL thủ công, bạn chỉ cần kế thừa các interface từ **JpaRepository** (hoặc **CrudRepository**). JPA sẽ tự động tạo các phương thức như findById(), findAll(), save(), v.v., giúp bạn thao tác dữ liệu mà không cần phải viết câu lệnh SQL.
* **Service**: Chứa logic nghiệp vụ, giống như trong JDBC, nhưng **Service** sẽ gọi các phương thức trong **Repository** để thao tác dữ liệu thay vì phải gọi các phương thức DAO thủ công.
* **Controller**: Xử lý các yêu cầu từ người dùng, gọi **Service** để thực hiện các tác vụ và trả kết quả về cho người dùng.

Tóm lại:

* **JDBC** là cách tiếp cận thủ công hơn trong việc tương tác với cơ sở dữ liệu, yêu cầu bạn phải tự quản lý kết nối, viết câu lệnh SQL, và ánh xạ kết quả thủ công vào các đối tượng Java.
* **JPA** cung cấp một cách tiếp cận dễ dàng hơn bằng cách tự động hóa các công việc như ánh xạ đối tượng và SQL. Bạn chỉ cần định nghĩa các Entity và Repository, còn lại JPA (thường là Hibernate) sẽ tự động xử lý phần còn lại.

Tuy cả hai đều tương tác với cơ sở dữ liệu, nhưng **JPA** giúp bạn tránh việc phải viết mã thủ công nhiều hơn, giúp mã nguồn dễ duy trì và mở rộng hơn so với **JDB**

# **CI / CD**

CI (Continuous Integration) và CD (Continuous Delivery hoặc Continuous Deployment) là hai khái niệm quan trọng trong quy trình phát triển phần mềm hiện đại nhằm tự động hóa và cải thiện quy trình phát triển, kiểm thử, và triển khai.

**1. CI - Continuous Integration (Tích hợp liên tục)**

CI là quá trình mà các nhà phát triển thường xuyên đưa mã nguồn mới vào một nhánh chính trong kho lưu trữ chung. CI sẽ tự động kiểm tra, build và test mã nguồn đó nhằm phát hiện lỗi sớm, giúp tránh các xung đột và lỗi lớn khi tích hợp.

**Lợi ích của CI:**

* Phát hiện lỗi sớm.
* Tiết kiệm thời gian kiểm thử thủ công.
* Đảm bảo mã nguồn luôn ổn định.

**2. CD - Continuous Delivery (Phân phối liên tục) và Continuous Deployment (Triển khai liên tục)**

CD có thể hiểu theo hai cách:

* **Continuous Delivery:** Sau khi mã nguồn vượt qua CI, mã sẽ được chuẩn bị và đóng gói sẵn sàng để triển khai. Ở bước này, người dùng có thể chọn triển khai thủ công hoặc tự động.
* **Continuous Deployment:** Đây là bước cao hơn của Continuous Delivery, khi mọi thay đổi sau khi được kiểm tra thành công sẽ tự động được triển khai vào môi trường sản xuất mà không cần can thiệp thủ công.

**Lợi ích của CD:**

* Giảm thiểu thời gian triển khai.
* Tăng tính ổn định và tin cậy.
* Giúp đưa sản phẩm đến người dùng nhanh hơn.

Tóm lại, CI/CD giúp tối ưu hóa quy trình phát triển phần mềm, giảm thiểu lỗi, và đẩy nhanh tốc độ phát hành sản phẩm.

**GitHub là một nền tảng phổ biến hỗ trợ rất tốt cho CI/CD. Bạn có thể:**

1. **Dùng GitHub Actions**: GitHub có một tính năng tích hợp sẵn gọi là **GitHub Actions** để tự động hóa CI/CD. Bạn có thể tạo các workflow (luồng công việc) để tự động kiểm tra, build, test và triển khai mã ngay khi có thay đổi. GitHub Actions rất linh hoạt và dễ dùng vì có nhiều action (thao tác) có sẵn và dễ dàng cấu hình.
2. **Kết nối GitHub với công cụ CI/CD bên ngoài**: Ngoài GitHub Actions, bạn cũng có thể kết nối GitHub với các công cụ CI/CD khác như **Jenkins**, **CircleCI**, **Travis CI**, hay **GitLab CI/CD**. Các công cụ này sẽ tự động kiểm tra mã của bạn khi có cập nhật trên GitHub.

GitHub giúp quản lý mã nguồn và kết hợp CI/CD giúp tự động hóa các bước kiểm tra và triển khai, tiết kiệm nhiều thời gian và công sức trong quá trình phát triển phần mềm.