**JPA In Spring Boot:**

* Java Persistence API (JPA) in Spring Boot is used for interacting with databases in a more object-oriented way.
* It abstracts the underlying database interactions and lets you work with Java objects (entities) that represent database tables.
* With JPA, you don't need to write SQL queries manually as it manages the database connection, query generation, and object mapping automatically.
* **Entity:** An entity in JPA represents a table in a relational database. Each instance of an entity represents a row in that table.
* **Repository:** Spring Data JPA provides “JpaRepository” to handle common CRUD operations automatically. You define an interface that extends “JpaRepository” and it comes with many useful methods like “findAll()”, “save()”, “deleteById()”, etc.
* **Service Layer:** The service layer is where the business logic is implemented, and it usually calls the repository to interact with the database.
* **Controller:** The controller exposes REST endpoints to interact with your service.
* **Application.properties:**  You need to configure the connection to the database, typically in the “application.properties” file.

**Common Annotations:**

***@Entity:*** Marks a class as an entity representing a table in the database.

***@Id:*** Specifies the primary key field of the entity.

***@GeneratedValue:*** Defines the strategy for generating primary key values.

***@Table:*** Used to specify the table name if it's different from the class name.

***@Column:*** Customizes the column definition for a field.

**HIBERNATE:**

* Hibernate is a popular Object-Relational Mapping (ORM) framework that helps to map Java objects to database tables, allowing for seamless integration between Java applications and relational databases.
* In Spring Boot, Hibernate is often used as the default JPA (Java Persistence API) provider.

**DIALECT:**

* In Spring Framework, a Dialect typically refers to a specific database language or SQL syntax that the application uses to communicate with a database.
* It is especially important when working with Spring Data JPA or Hibernate, where the dialect defines how SQL queries are generated for a particular database.

**Benefits of Dialect:**

* **Database portability:** Since Hibernate automatically generates the SQL based on the dialect, switching between databases is easier with fewer code changes.
* **Optimization:** Hibernate can optimize SQL queries based on the capabilities of the specific database.

**GITHUB REPOSITORY:**

[LalithaRavuri/SpringWeather (github.com)](https://github.com/LalithaRavuri/SpringWeather)