**Frontend (React):**

* Use fetch or libraries like axios to make HTTP requests.

axios.post('http://localhost:8080/api/data', { name: 'John Doe' })

.then(response => console.log(response.data))

.catch(error => console.error(error));

**Backend (Spring Boot):**

* Create a REST controller to handle HTTP requests.

@RestController

@RequestMapping("/api")

public class ApiController

{

@PostMapping("/data")

public ResponseEntity<String> receiveData(@RequestBody Map<String,   
 String> payload)

{

return ResponseEntity.ok("Received: " + payload.get("name"));

}

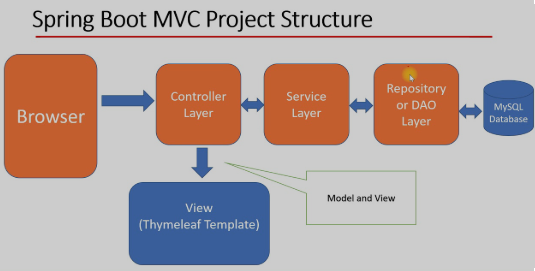
}

**Spring Boot**

* Spring Boot is designed to
* reduce complexity and
* improve productivity in Java application development.
* providing opinionated defaults,
* embedded servers, and
* production-ready features,
* enables developers to build robust, scalable applications quickly and efficiently.

**Spring Boot Application Flow**

1. **Client Request**: A user or client sends an HTTP request.
2. **Controller**: The request is routed to a controller, where it is processed.
3. **Service Layer**: The controller delegates the request to the service layer for business logic processing.
4. **Data Layer**: The service layer interacts with the repository to fetch or save data.
5. **Response**: The controller sends the response back to the client.



**Advantages of Spring Boot Architecture**

1. **Separation of Concerns**: Each layer focuses on a specific responsibility.
2. **Loose Coupling**: Layers communicate through well-defined interfaces, improving modularity.
3. **Scalability**: The architecture is easily extensible for enterprise-level applications.
4. **Reduced Boilerplate**: Spring Boot provides sensible defaults and simplifies configuration.

**Annotations**

Spring Boot provides a wide range of annotations to simplify and organize application development.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Annotation** | | **Description** |
| @SpringBootApplication | A combination of @Configuration, @EnableAutoConfiguration, and @ComponentScan.  It marks the main class as the entry point for a Spring Boot application. |
| |  |  | | --- | --- | | @Service |  | | Specialization of @Component for service layer classes. |
| |  |  | | --- | --- | | @Repository |  | | Specialization of @Component for the data access layer. |
| |  |  | | --- | --- | | @Controller |  | | Marks a class as a Spring MVC controller. |
| |  | | --- | | @RestController | | Combines @Controller and @ResponseBody for REST APIs. |
| |  | | --- | | @Autowired |  |  | | --- | |  | | Automatically injects a bean by type. Can be applied to fields, constructors, or methods. |
| |  |  | | --- | --- | | @Entity |  | | Marks a class as a JPA entity (maps to a database table). |
| |  |  | | --- | --- | | @Id |  | | Specifies the primary key field of an entity. |
| |  |  | | --- | --- | | @GeneratedValue |  | | Specifies how the primary key value is generated (e.g., AUTO, IDENTITY). |
| |  |  | | --- | --- | | @Table |  | | Provides table-level details like name and schema for an entity. |
| |  |  | | --- | --- | | @Column |  | | Maps a field to a database column. Allows customization of column name, length, etc. |
| |  |  | | --- | --- | | @Repository |  | | Marks a class as a repository for data access. |
| |  |  | | --- | --- | | @Query |  | | Defines a custom JPQL or SQL query for a repository method. |
| |  |  | | --- | --- | | @RequestMapping |  | | |  |  | | --- | --- | |  | Maps HTTP requests to handler methods (generic mapping). | |
| |  |  | | --- | --- | | @GetMapping |  | | |  |  | | --- | --- | |  | Shortcut for @RequestMapping (method = RequestMethod.GET). | |
| |  |  | | --- | --- | | @PostMapping |  | | |  |  | | --- | --- | |  | Shortcut for @RequestMapping (method = RequestMethod.POST). | |
| |  |  | | --- | --- | | @PutMapping |  | | |  |  | | --- | --- | |  | Shortcut for @RequestMapping (method = RequestMethod.PUT). | |
| |  |  | | --- | --- | | @DeleteMapping |  | | |  |  | | --- | --- | |  | Shortcut for @RequestMapping (method = RequestMethod.DELETE). | |
| |  |  | | --- | --- | | @PathVariable |  | | |  |  | | --- | --- | |  | Binds a path variable from the URL to a method parameter. | |
| |  |  | | --- | --- | | @RequestParam |  | | |  |  | | --- | --- | |  | Binds a query parameter from the URL to a method parameter. | |
| |  |  | | --- | --- | | @RequestBody |  | | |  |  | | --- | --- | |  | Binds the body of an HTTP request to a method parameter. | |
| |  |  | | --- | --- | | @ResponseBody |  | | |  |  | | --- | --- | |  | Sends the method’s return value as the HTTP response body. | |

**Advantages of Using JpaRepository**

1. **Reduced Boilerplate Code**: No need to write typical CRUD implementations.
2. **Custom Queries**: Easily create derived or annotated custom queries.
3. **Pagination and Sorting**: Built-in support for pageable data retrieval.
4. **Integration**: Seamlessly integrates with Spring Boot, leveraging features like dependency injection and transaction management.
5. **Scalability**: Supports large datasets with features like paging and lazy loading.

#### CRUD Operations

JpaRepository provides built-in methods to perform Create, Read, Update, and Delete operations on the database.

| Method | Description |
| --- | --- |
| save(entity) | Saves a given entity. If the entity already exists, it updates it. |
| findById(id) | Retrieves an entity by its ID. |
| findAll() | Retrieves all entities from the database. |
| delete(entity) | Deletes a given entity. |
| deleteById(id) | Deletes an entity by its ID. |
| count() | Returns the total number of entities in the table. |

You can define custom queries using the @Query annotation.

Example:

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

@Query("SELECT u FROM User u WHERE u.email = ?1")

User findByEmail(String email);

@Query(value = "SELECT \* FROM users WHERE age > ?1", nativeQuery = true)

List<User> findUsersAboveAge(int age);

}

| **Operation** | **MySQL Query** |
| --- | --- |

|  |  |
| --- | --- |
| **Create Table** | CREATE TABLE users ( id INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(50) NOT NULL, email VARCHAR(100) NOT NULL, age INT, password VARCHAR(50)); |

|  |  |
| --- | --- |
| **Insert Record** | INSERT INTO users (username, email, age, password)VALUES ('RAMA, 'john.doe@example.com', 25, 'password123'); |

|  |  |
| --- | --- |
| **Read All Records** | SELECT \* FROM users; |

|  |  |
| --- | --- |
| **Read Specific Data** | SELECT username, email FROM users WHERE age > 18; |

|  |  |
| --- | --- |
| **Update Record** | UPDATE users SET email = 'rama@example.com', age = 26 WHERE username = 'RAMA; |

|  |  |
| --- | --- |
| **Delete Record** | DELETE FROM users WHERE username = 'RAMA; |