To understand how **Controller**, **Entity**, **Repository**, and **Service** work together in a **Spring Boot** application, think of them as layers in a software architecture that handle different aspects of the application:

---

1. Entity (Data Model)

The **Entity** represents the data structure and maps to a database table. It is the foundation of the application because it defines what data will be stored.

- **Purpose**: Represents the database table.
- **Annotations**: Typically annotated with `@Entity` and `@Table`.
- **Example**:

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Getters and Setters
}
```

---

2. Repository (Data Access Layer)

The **Repository** handles database operations like saving, updating, deleting, and querying the data. It abstracts the complexities of interacting with the database.

- **Purpose**: Provides methods to interact with the database.
- **Annotations**: Extends `JpaRepository` or `CrudRepository`.
- **Example**:

```java
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByName(String name); // Custom query
}
```

3. Service (Business Logic Layer)

The **Service** contains the business logic of the application. It acts as a bridge between the **Controller** and **Repository**, ensuring that data is processed and validated before interacting with the database.

- **Purpose**: Implements the application's business rules and processes.
- **Annotations**: Annotated with `@Service`.
- **Example**:

```java
@Service
public class UserService {
    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User saveUser(User user) {
        return userRepository.save(user);
    }
}
```

---

4. Controller (Presentation Layer)

The **Controller** handles HTTP requests (e.g., GET, POST) and sends responses. It interacts with the **Service** to retrieve or process data and return it to the user, often in JSON format.

- **Purpose**: Handles incoming web requests and sends appropriate responses.
- **Annotations**: Annotated with `@RestController` and request mapping annotations like `@GetMapping`, `@PostMapping`.
- **Example**:

```java
@RestController
@RequestMapping("/users")
public class UserController {
    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.saveUser(user);
    }
}
```

Sequence of Flow in a Spring Boot Application:

1. **Client sends an HTTP request** to the Controller (e.g., `GET /users`).
2. The **Controller** processes the request and calls the appropriate method in the **Service**.
3. The **Service** implements business logic and interacts with the **Repository**.
4. The **Repository** queries or updates the **Entity** in the database.
5. The **Service** returns the result to the **Controller**.
6. The **Controller** sends the response back to the client.

Visual Representation:

1. **Controller**: Handles the HTTP request/response.
2. **Service**: Contains business logic.
3. **Repository**: Communicates with the database.
4. **Entity**: Represents the database table.