A REST API (Representational State Transfer Application Programming Interface) in Spring Boot is a web service that allows interaction with backend systems using HTTP methods. It typically follows a stateless, client-server architecture where resources are identified by URLs, and clients (like web or mobile apps) can interact with these resources using standard HTTP operations.

Here's an explanation of how REST API works in Spring Boot:

Key Concepts:

1. **Resource**: A resource in REST represents any object or entity that can be accessed or manipulated, such as users, products, or orders. In Spring Boot, resources are often modeled as Java classes.
2. **HTTP Methods**: REST APIs in Spring Boot use HTTP methods to perform operations on resources. The most commonly used methods are:
    - `GET`: Retrieve a resource.
    - `POST`: Create a new resource.
    - `PUT`: Update an existing resource.
    - `DELETE`: Delete a resource.
    - `PATCH`: Partially update a resource.
3. **Stateless**: Each request from the client to the server must contain all the necessary information, and the server should not rely on previous requests. There is no session stored on the server side between API calls.
4. **Endpoints**: Endpoints in REST APIs are URLs where resources are exposed. In Spring Boot, you define these endpoints using annotations like `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping`.

Steps to Create a REST API in Spring Boot:

1. **Setup Spring Boot**: First, create a Spring Boot project with dependencies like `Spring Web` to build REST APIs.
2. **Define a Model Class**: The model represents the data structure (resource). For example, a `User` model might look like:

```
java
Copy code
public class User {
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
}
```

3. **Create a Controller**: The controller handles HTTP requests and responses. It maps the endpoints to methods that perform specific actions (like getting, creating, updating, or deleting resources).

```
java
Copy code
@RestController
@RequestMapping("/users")
public class UserController {

    // Get all users
    @GetMapping
    public List<User> getAllUsers() {
        // return a list of users
    }

    // Get a user by ID
    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        // return a user by ID
    }

    // Create a new user
    @PostMapping
    public User createUser(@RequestBody User user) {
        // save and return the new user
    }

    // Update a user by ID
    @PutMapping("/{id}")
    public User updateUser(@PathVariable Long id, @RequestBody User
user) {
        // update and return the user
    }

    // Delete a user by ID
    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) {
        // delete the user
    }
}
```

4. **Service Layer (Optional)**: You can add a service layer to separate business logic from the controller, which would handle fetching, saving, or modifying data.
5. **Repository Layer**: Spring Boot can use `Spring Data JPA` for database interaction, making it easy to perform CRUD operations.

```
java
Copy code
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

6. **Run the Application**: Once you have the controller and service set up, you can run the Spring Boot application, and your REST API will be exposed on the specified endpoints.

Example API Request and Response:

- **Request**:
  - `GET /users/1`
  - Response (JSON):

```
json
Copy code
{
  "id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

- **Request**:
  - `POST /users`
  - Body:

```
json
Copy code
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com"
}
```

  - Response (JSON):

```
json
Copy code
{
  "id": 2,
  "name": "Jane Doe",
  "email": "jane.doe@example.com"
}
```

Common Annotations in Spring Boot REST API:

- `@RestController`: Marks a class as a RESTful controller where each method returns a JSON or XML response.
- `@RequestMapping`: Maps the base URL of the controller.
- `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`: Handle specific HTTP methods for the corresponding endpoints.
- `@RequestBody`: Binds the request body to a method parameter.
- `@PathVariable`: Extracts variables from the URL.

Spring Boot simplifies the development of REST APIs by offering an intuitive structure, built-in HTTP method support, and easy integration with databases using Spring Data JPA.