스프링시작하기

스프링 API 7장

박명회

1.1 My RESTful Services

Description	REST API	HTTP Method
사용자 전체 보기	/users	GET
사용자 만들기	/users	POST
한명의 사용자 보기	/users/{id}	GET
사용자 삭제	/users/{id}	DELETE
사용자의 모든 게시물보기	/users/{id}/posts	GET
사용자의 게시물 만들기	/users/{id}/posts	POST
사용자의 게시물 보기	/users/{id}/posts/{post_id}	GET

1.2 HelloController

@RestController 는 Spring Framework에서 제공하는 애노테이션으로, RESTful 웹 서비스를 개발할 때 사용됩니다. 이를 이용하면 간단하게 HTTP 요청에 대한 응답을 생성할 수 있습니다. 아래는 간단한 예제를 통해 를 사용하는 방법을 보여줍니다.

```
import org.springframework.web.bind.annotation.*;

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }

    @PostMapping("/greet")
    public String greet(@RequestParam String name) {
        return "Hello, " + name + "!";
    }
}
```

1.3 HelloController

@PathVariable은 Spring에서 제공하는 애노테이션 중 하나로, URL 경로에서 변수 값을 추출하여 메서드의 매개변수로 전달할 때 사용됩니다. 이를 통해 동적인 URL을 처리할 수 있습니다. 아래의 예제는 을 사용하여 동적인 경로 변수를 처리하는 방법을 보여줍니다.

```
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserController {

    @GetMapping("/{id}")
    public String getUserById(@PathVariable Long id) {
        return "User ID: " + id;
    }
}
```

1.4 USER 도메인 클래스 생성

@Data : getter(), setter(), equals(), hashCode(), toString() 메서드를 생성합니다.
@NoArgsConstuctor : 매개변수가 없는 생성자를 생성합니다.
@AllArgsConstructor : 모든 필드를 매개변수로 받는 생성자를 생성합니다.

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private Long id;
    private String username;
    private String email;
}
```

1.5 USER Service 생성

getAllUsers() 메서드는 모든 사용자를 반환하고,
getUserById() 메서드는 주어진 ID에 해당하는 사용자를 찾아서 반환하며,
addUser() 메서드는 사용자를 추가하고,
removeUser() 메서드는 주어진 ID에 해당하는 사용자를 삭제합니다.

```java
public class UserServiceImpl implements UserService {
    private List<User> userList = new ArrayList<>();

    @Override
    public List<User> getAllUsers() {
        return userList;
    }

    @Override
    public User getUserById(Long id) {
        for (User user : userList) {
            if (user.getId().equals(id)) {
                return user;
            }
        }
        return null;
    }

    @Override
    public void addUser(User user) {
        userList.add(user);
    }

    @Override
    public void removeUser(Long id) {
        userList.removeIf(user -> user.getId().equals(id));
    }
}
```

1.6 UserController 생성

경로가 오면 해당 ID의 사용자를 반환합니다.

POST 요청은 새로운 사용자를 추가하며, DELETE 요청은 특정 ID의 사용자를 삭제합니다.
주의할 점은 UserController가 UserService를 의존하도록 만들어졌는데, 이는 생성자 주입을 통해 이루어집니다. 이렇게 함으로써 UserController는 UserService의 구현체에 의존하게 되는데, 이를 통해 테스트 용이성과 느슨한 결합을 유지할 수 있습니다.
이제 이 UserController를 사용하여 사용자 관련 HTTP 요청을 처리할 수 있습니다. 요청을 보내면 UserController는 UserService를 호출하여 적절한 작업을 수행하고 결과를 반환합니다.

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

    private final UserService userService;

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

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

    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
    }

    @PostMapping
    public void addUser(@RequestBody User user) {
        userService.addUser(user);
    }

    @DeleteMapping("/{id}")
    public void removeUser(@PathVariable Long id) {
        userService.removeUser(id);
    }
}
```

고맙습니다.