Spring Boot Rest Api (best practices)

Presentated by karim

Use Descriptive and Consistent URLs

http(s)://{Domain name (:Port number)}/{A value indicating REST API}/{API version}/{path for identifying a resource}

http://example.com/api/v1/members/M000000001

Use Descriptive and Consistent URLs

✅ /users
❌ /getUser

✅ /employees/45
❌ /employees?employeeId=45

✅ /departments/3/employees
❌ /employeesInDepartment/3

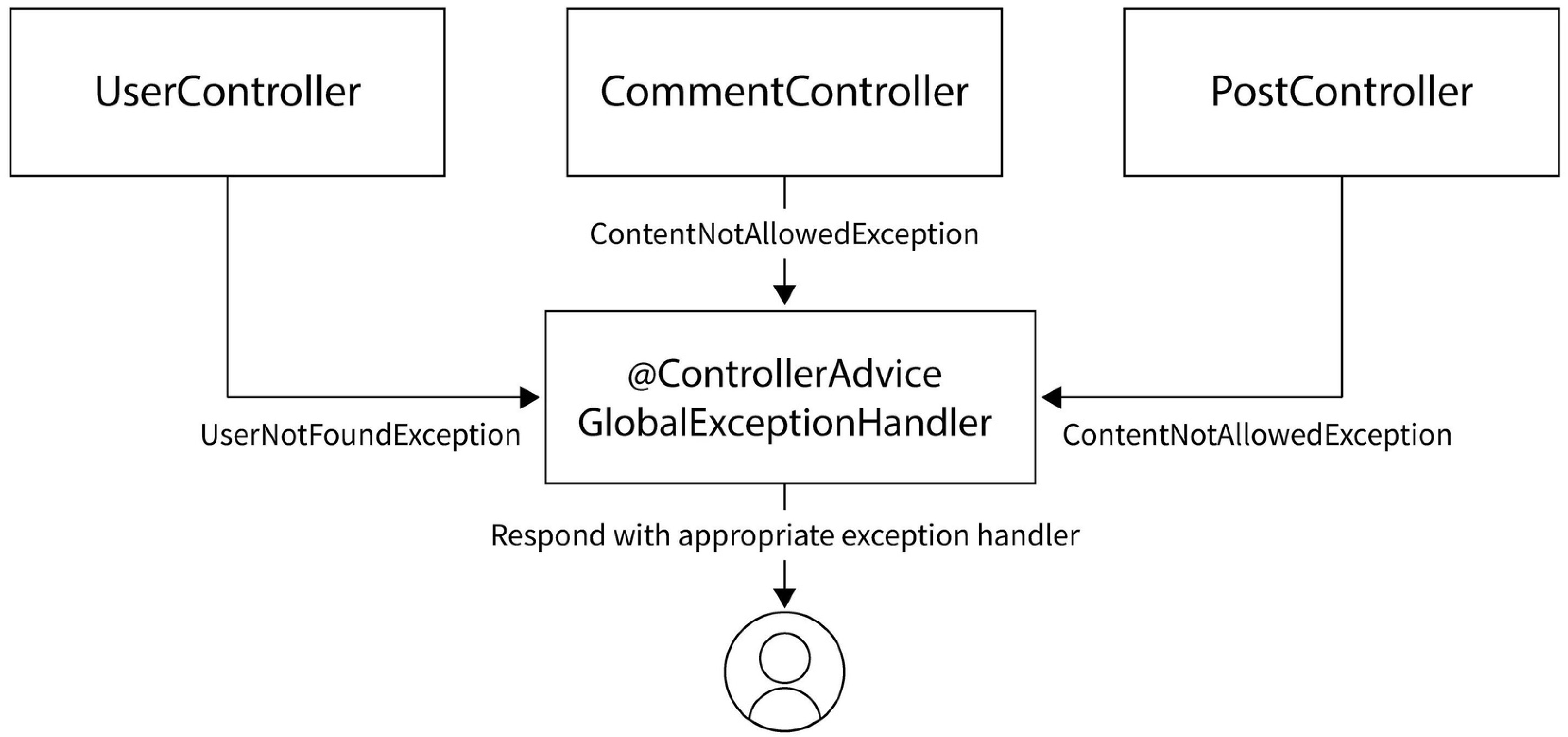
✅ /books?author=Rowling&sort=title
❌ /booksByRowlingSortedByTitle

Use status codes

```
 @PutMapping("/{demandId}/reject")
   public ResponseEntity<DemandResponseDTO>
rejectDemand(@PathVariable Long demandId){
       return
ResponseEntity.ok(demandService.rejectDemand(demandId));
   }
```

202 (Accepted) , 204 (No Content), 201 (Created), 401

Implement Proper Error Handling



Use proper response

```
@AllArgsCostructor
@NoArgsConstructor
@Getters
@Setters
@Builder
public class ResponseMessage {
        private int statusCode;
        private LocalDateTime timestamp;
        private String message;
        private Object data;
        private  String path;
}
```

Use DTOs (Data Transfer Objects)

```
public record UserDTO(

        @NotBlank(message = "Name is required")
        String name,
        @NotBlank(message = "Email is required")
        @Email(message = "Email foramt is invalid")
        String email,
        @NotBlank(message = "Phone number is required")
        @Pattern(regexp = "^\\+?[0-9]{10,13}$", message = "Phone number format is invalid")
        String phoneNumber,
        @Pattern(regexp = "^(Agent|Manager|Client)$", message = "Role must be Agent,Manager or Client")
        String role,
        LocalDateTime createdAt,
        LocalDateTime modifiedAt
) {
}
```

Implement Validation

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @PostMapping
    public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
        // Validate user data using annotations like @NotNull and @Size
        // Create and return a user
    }
}
```

Secure Your API

Implement proper authentication and authorization mechanisms to protect your API. Consider using OAuth 2.0 or JWT (JSON Web Tokens) for token-based authentication.

Documentation

Provide comprehensive API documentation using tools like Swagger,Spring RestDocs or Postman. Clear documentation makes it easier for developers to understand and use your API.

Test Thoroughly

Write unit tests, integration tests, and end-to-end tests to ensure your API functions as expected. Consider using tools like JUnit and Postman for testing.