REST API Deep Dive 1

Manipulating HTTP Responses

Content

- Introduction
- ResponseEntity
- ResponseEntity Alternatives
- Java Optional class
- Builder and Optional Patterns
- Indexing In Practice
- Creating Transactions Endpoints

ResponseEntity

- **ResponseEntity** represents the whole HTTP **response**: **status code**, **headers**, and **body**
- **ResponseEntity** is a generic type. As a result, we can use any type as the response body

```
@GetMapping("/api/users/{id}")
public ResponseEntity<User> fetchUser(@PathVariable Long id) {
   User user = this.userRepository.findById(id).get();
   return new ResponseEntity<>(user, HttpStatus.OK);
}
```

ResponseEntity Different Status Codes

```
@GetMapping("/age")
ResponseEntity<String> age(
      @RequestParam("yearOfBirth") int yearOfBirth) {
   if (isInFuture(yearOfBirth)) {
      return new ResponseEntity<>(
            "Year of birth cannot be in the future",
            HttpStatus.BAD_REQUEST);
   }
   return new ResponseEntity<>(
         "Your age is " + calculateAge(yearOfBirth),
         HttpStatus.OK);
}
```

ResponseEntity Headers

```
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
   HttpHeaders headers = new HttpHeaders();
   headers.add("Custom-Header", "foo");

   return new ResponseEntity<>(
         "Custom header set", headers, HttpStatus.OK);
}
```

ResponseEntity Header Builder and BodyBuilder

ResponseEntity provides two nested builder interfaces: **HeadersBuilder** and its subinterface, **BodyBuilder**. Hence we can access their capabilities through the **static** methods of **ResponseEntity**.

```
@GetMapping("/hello")
ResponseEntity<String> hello() {
   return ResponseEntity.ok("Hello World!");
}
```

```
BodyBuilder accepted();
BodyBuilder badRequest();
BodyBuilder created(java.net.URI location);
HeadersBuilder<?> noContent();
HeadersBuilder<?> notFound();
BodyBuilder ok();
BodyBuilder status(HttpStatus status);

BodyBuilder status(int status)
```

ResponseEntity<T> BodyBuilder.body(T body)

```
@GetMapping("/age")
ResponseEntity<String> age(@RequestParam("yearOfBirth") int yearOfBirth) {
   if (isInFuture(yearOfBirth)) {
       return ResponseEntity.badRequest()
               .body("Year of birth cannot be in the future");
   }

   return ResponseEntity.status(HttpStatus.OK)
           .body("Your age is " + calculateAge(yearOfBirth));
}
```

With Custom Headers

```
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
   return ResponseEntity.ok()
           .header("Custom-Header", "foo")
           .body("Custom header set");
}
```

***BodyBuilder.body()* returns a *ResponseEntity* instead of *BodyBuilder,* it should be the last call.**

Alternatives

- @ResponseBody - Spring treats the result value of the method as the HTTP response body
- @ResponseStatus - Spring returns with a custom HTTP status

Alternatives: Manipulate The Response Directly

```
@GetMapping("/manual")
void manual(HttpServletResponse response) throws IOException {
   response.setHeader("Custom-Header", "foo");
   response.setStatus(200);
   response.getWriter().println("Hello World!");
}
```

Java Optional

1. Optional class was introduced in Java 8
2. The purpose of the class is to provide a type-level solution for representing optional values instead of null references

Java Optional JPA CrudRepository

```
@GetMapping("/{id}")
public ResponseEntity<User> findOne(@PathVariable Long id) {

   Optional<User> optionalUser = userRepository.findById(id);
   if(optionalUser.isPresent()){
      return ResponseEntity.status(HttpStatus.OK).body(optionalUser.get());
   }
   return  ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

JPA CrudRepository With Try Catch

```
@GetMapping("/{id}")
public ResponseEntity<User> findOne(@PathVariable Long id) {

   try{
      return
ResponseEntity.status(HttpStatus.OK).body(userRepository.findById(id).get());
   }catch (NoSuchElementException e){
      return ResponseEntity.notFound().build();
   }
}
```

Adding **Indexing** To User and Transactions

```
@Entity
@Table(name = "users", indexes = {@Index(name = "first_name_index",columnList =
"first_name",unique = false),
      @Index(name = "last_name_index", columnList = "last_name", unique = false)
})
public class User {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "native")
   private Long id;
```

Transactions

Endpoints To Create

1. Read All transactions
2. Read Transactions by user id
    a. /api/users/1/transactions
3. Read Transaction By Id

Transaction Currencies

1. Add Currency Entity
2. Add Transaction <-> Currency Relationship
3. Add Currency Information To Transaction Endpoints

- https://www.baeldung.com/spring-response-entity
- https://www.baeldung.com/java-optional