Advanced Java

Agenda

- REST services

REST web services

- REST stands for REpresentation State Transfer.
  - Object characteristics: State, Behaviour & Identity
    - State: Values of data members/fields.
  - Object state can be transferred (from server to client & vice-versa) in any format like JSON or XML.
- It is a Protocol to invoke web services from any client (with internet connection).
- Client can use any platform/language.
- REST is lightweight (than SOAP).
  - No stub and proxy classes.
  - No XML grammer (xsd) checks.
- REST works on top of HTTP protocol.
- It uses HTTP protocol request methods.
  - GET: to get records.
  - POST: to create new record.
  - PUT: to update existing record.
  - DELETE: to delete record.
- REST services are stateless.
  - Each REST request is independent of another.
  - Request should include all required inputs and produce expected output.

XML vs JSON

- XML

```
<book>
    <id>1</id>
    <name>Abc</name>
    <price>123.45</price>
</book>
```

- JSON

```
{
    "id": 1,
    "name": "Abc",
    "price": 123.45
}
```

JSON format

- JSON is Java Script Object Notation.
- It is set of key-value pairs and supports few data types like numeric, string, boolean, null, array and objects.

```
{
    "name": "Nilesh",
    "contact": {
        "email": "nilesh@sunbeaminfo.com",
        "mobile": "9527331338"
    },
    "age": 38,
    "rating": 4.5,
    "domains": [ "Enterprise apps", "Big Data", "Embedded", "Operating Systems" ],
    "experience": [
        {
            "company": "Seed",
            "duration": "Aug 2001 to May 2003"
        },
        {
            "company": "Freelance Trainer",
            "duration": "Feb 2004 to May 2004"
        },
        {
            "company": "Sunbeam",
            "duration": "May 2004 to Dec 2022"
        }
    ]
}
```

REST request/response formats

- Variety of request and response formats are followed in industry.
  - JSend: https://github.com/omniti-labs/jsend

REST request/response model

- GET: http://localhost:8080/books
  - Request body: Empty
  - Response body: List of Books - 200

```
[
    { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... },
    { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... },
    { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... }
]
```

- GET: http://localhost:8080/books/1
  - Request body: Empty

    - Response body: Book - 200

      ```
      { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... }
      ```

- POST: http://localhost:8080/books
    - Request body:

      ```
      { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... }
      ```

    - Response body: 200
- PUT: http://localhost:8080/books/1
    - Request body

      ```
      { "_id": ..., "name": "...", "author": "...", "subject": "...", "price": ... }
      ```

    - Response body: 200
- DELETE: http://localhost:8080/books/1
    - Request body
    - Response body: 200

Spring Boot REST services

- Spring REST is subset of Spring Web MVC.
- Spring boot auto-configures WebMvc and messageConverters automatically.
- REST request handlers are implemented similar to web mvc request handlers.
    - Implemented using @Controller
        - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping or @RequestMapping
        - @RequestBody, @ResponseBody / ResponseEntity<>

        - @PathVariable, @RequestParam
    - Can be implemented using @RestController
        - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping or @RequestMapping
        - @RequestBody, ResponseEntity<>
        - @PathVariable, @RequestParam

ResponseEntity

- ResponseEntity object is used to control response status as well as response body.
- If ResponseEntity<> is return type, no need to use @ResponseBody explicitly.
- Important methods:
    - ResponseEntity.ok(body) --> ResponseEntity object
        - status = 200
        - body = given object json representation
    - ResponseEntity.notFound() --> HeaderBuilders object (not to be returned from method)
        - status = 404
    - ResponseEntity.notFound().build() --> ResponseEntity object
        - status = 404
        - body = empty response body
    - ResponseEntity.notFound().body(obj) --> ResponseEntity object
        - status = 404
        - body = given object json representation
    - ResponseEntity.noContent() --> HeaderBuilders object (not to be returned from method)
        - status = 204
    - ResponseEntity.noContent().build() --> ResponseEntity object
        - status = 204
        - body = empty response body
    - ResponseEntity.noContent().body(obj) --> ResponseEntity object
        - status = 204
        - body = given object json representation
    - ResponseEntity.created(uri) --> BodyBuilder object (not to be returned from method)
        - status = 201

  - ResponseEntity.created(uri).build() --> ResponseEntity object
    - status = 201
    - body = empty response body
  - ResponseEntity.created(uri).body(obj) --> ResponseEntity object
    - status = 201
    - body = given object json representation
  - ResponseEntity.internalServerError() --> HeaderBuilders object (not to be returned from method)
    - status = 500
  - ResponseEntity.internalServerError().build() --> ResponseEntity object
    - status = 500
    - body = empty response body
  - ResponseEntity.internalServerError().body(obj) --> ResponseEntity object
    - status = 500
    - body = given object json representation
  - ResponseEntity.status(code) --> BodyBuilder object (not to be returned from method)
    - status = given code
  - ResponseEntity.status(code).body(obj) --> ResponseEntity object
    - status = given code
    - body = given object json representation
- Examples
  - 200 -- for success and response body will contain data.

```
@GetMapping("/url")
public ResponseEntity<?> getAllMovies() {
    // ...
    return ResponseEntity.ok(list);
}
```

  - 201 -- for new object created successfully and response body will contain data.

```
@PostMapping("/url")
public ResponseEntity<?> saveMovies(@RequestBody Movie m) {
    // ...
    return ResponseEntity.created(uri).build();
}
```

- 403 -- API is not accessible with given token.

```
return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
```

- 404 -- for object not found

```
return ResponseEntity.notFound().build();
```

Swagger

- Swagger is an open source set of rules, specifications and tools for developing and describing RESTful APIs.
- The Swagger framework allows developers to create interactive, machine and human-readable API documentation.
- For Spring Boot application you only need to add swagger dependency.

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.7.0</version>
</dependency>
```

- Start the application (having @RestController) and open swagger ui in Browser.

- http://localhost:8080/swagger-ui/index.html

Spring REST Internals

- Spring REST is subset of Spring MVC.
- Like Spring MVC
  - Each request first encountered by Front controller.
  - The request url is mapped to Request handler method (in controller) by HandlerMapping bean.
  - This request handler method is executed by HandlerAdapter bean.
- To deal with @RequestBody and @ResponseBody, handler-adapter bean creates RequestResponseBodyMethodProcessor bean.
- This bean internally use Jackson converter (MappingJackson2HttpMessageConverter bean) to convert request body json to required Java object (as given in method argument).
- Then request handler method is executed that may call service and dao layer. It produces response Java object.
- Again RequestResponseBodyMethodProcessor bean internally use Jackson converter to convert Java object to Json format.
- Finally this Json response is sent back to the client by DispatcherServlet.
- Refer slides.

Manipulating JSON output/input

- Using some Jackson annotations we can modify the JSON response data and control JSON request data.
- This is also called as "Static filtering".
- @JsonIgnore -- do not add this field in json output and do not process this field from json input.
- @JsonProperty("newName") -- Json field name is changed to "newName".
- @JsonInclude -- Control which fields to be added in Json.
  - Include.NON_NULL -- do not generate output for null fields in Java object.
  - Include.NON_EMPTY -- do not generate output for null fields or empty collection in Java object.
  - Include.ALWAYS -- always generate output for all fields though they are null or empty.
- @JsonManagedReference -- output is added into json (forward ref)
- @JsonBackReference -- output is not added into json (backward ref).

Response Util Object

- Response/Result object

- status = "success" or "error"
- data = any object `{ ... }` or array `[ ... ]` -- in case of success
- message = string error -- in case of error
- getters/setters
- static helper methods to create Response objects.

```
public static <T> Response<T> success(T data) {
    Response<T> resp = new Response<>("success", null, data);
    return resp;
}
public static <T> Response<T> error(String message) {
    Response<T> resp = new Response<>("error", message, null);
    return resp;
}
```

- Generated response
  - success

```
{
    "status": "success",
    "data": { ... }
}
```

  - error

```
{
    "status": "error",
    "message": "error message"
}
```

Assignments

1. Implement following REST services
    - get all books
    - get book by id
    - save new book
    - update book
    - delete book by id
    - get all users
    - get user by email
    - get user by id
    - save new user
    - update user
    - delete by user id
2. Create REST APIs as per following requirements. Refer given docs in MobileShopAssignmentDocs folder.
    - GET: http://server:port/users
        - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": [
        {
            "id": int,
            "uname": string,
            "email": string,
            "mobile": string,
            "password": string,
            "birth": string
        },
        ...
    ]
}
```

- GET: http://server:port/users/{id}
  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": {
        "id": int,
        "uname": string,
        "email": string,
        "mobile": string,
        "password": string,
        "birth": string
    }
}
```

- GET: http://server:port/users/email/{email}
  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": {
        "id": int,
        "uname": string,
        "email": string,
        "mobile": string,
        "password": string,
        "birth": string
    }
}
```

- GET: http://server:port/mobiles/{id}
  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": {
        "id": int,
        "mname": string,
        "ram": int,
        "storage": int,
        "company": string,
        "price": double,
        "image": string
    }
}
```

- GET: http://server:port/mobiles
  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": [
        {
        "id": int,
        "mname": string,
        "ram": int,
        "storage": int,
        "company": string,
        "price": double,
        "image": string
        },
        ...
    ]
}
```

- POST: http://server:port/orders
  - Request Body

```
{
    "uid": int,
    "mid": int,
}
```

  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": {
        "id": int,
        "mobile": {
            "id": int,
            "mname": string,
            "ram": int,
            "storage": int,
            "company": string,
            "price": double,
            "image": string
        },
        "user": {
            "id": int,
            "uname": string,
            "email": string,
            "mobile": string,
            "password": string,
            "birth": string
        }
    }
}
```

- GET: http://server:port/orders/user/id
  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": [
    {
        "id": int,
        "mobile": {
            "id": int,
            "mname": string,
            "ram": int,
            "storage": int,
            "company": string,
            "price": double,
            "image": string
        },
        "user": {
            "id": int,
            "uname": string,
            "email": string,
            "mobile": string,
            "password": string,
            "birth": string
        }
    },
    ...
    ]
}
```

- POST: http://server:port/mobiles
  - Request Body

```
{
    "id": int,
    "name": string,
    "ram": int,
    "storage": int,
    "company": string,
    "price": double,
    "imageFile": file
}
```

  - Response Body

```
{
    "status": "success" or "error",
    "message": string (if error),
    "data": {
        "id": int,
        "mname": string,
        "ram": int,
        "storage": int,
        "company": string,
        "price": double,
        "image": string
    }
}
```

- GET: http://server:port/images/{imagename}
  - Response Body
    - content-type: image/jpeg
    - image binary data