Changes to the Hotel Application

Introduction

Here, we will have a walk-through implementation of Project Lombok and the creation of a DTO (Data Transfer Object) in a Spring Boot application. This document provides a detailed problem statement to implement Lombok and DTO for the HotelApplication Project.

Find the link to the old **HotelApplication** ahead, use this project, and make the changes as stated below in the document. LINK

1. Step 1:

- Remove the **RatingServiceCommunicator** class from the application.

```
@Service
public class RatingServiceCommunicator {

	private final RestTemplate restTemplate;

	@Autowired
	public RatingServiceCommunicator(RestTemplateBuilder restTemplateBuilder)
{
		this.restTemplate=restTemplateBuilder.build();
	}

	public long getRating(String id)
	{
		String url ="http://localhost:8081/rating/id/";


		 Long ratingResponse = restTemplate.getForObject(url+id,
Long.class);

		return ratingResponse;
	}

	public void addRating(Map<String, Long> ratingsMap) {

		String url ="http://localhost:8081/rating/add";

		HttpEntity<Map<String, Long>> requestEntity = new
HttpEntity<>(ratingsMap);


restTemplate.exchange(url,HttpMethod.POST,requestEntity,Object.class);
	}
```

```
	public void updateRating(Map<String, Long> ratingsMap)
	{
		String url ="http://localhost:8081/rating/update";

		HttpEntity<Map<String, Long>> requestEntity = new
HttpEntity<>(ratingsMap);


restTemplate.exchange(url,HttpMethod.PUT,requestEntity,Object.class);
	}

	public void deleteRating(String id)
	{
		String url ="http://localhost:8081/rating/remove/id/";

		try {

restTemplate.exchange(url+id,HttpMethod.DELETE,null,Object.class);
		}
		catch(HttpClientErrorException e)
		{
			throw new
HttpRatingServiceNotFound(HttpStatusCode.valueOf(HttpStatus.NOT_FOUND.value()));
		}


	}

}
```

- Remove usages of *RatingServiceCommunicator* from the **HotelService** class**.** The line of code to be removed has been marked in red font.

```
@Service
public class HotelService {

	List<Hotel> hotelList = new ArrayList<>();
	Map<String,Hotel> hotelMap= new HashMap<>();
←
	@Autowired
	RatingServiceCommunicator ratingServiceCommunicator;

	public void createHotel(Hotel hotel) {

		Map<String, Long> ratingsMap = new HashMap<>();
		hotelList.add(hotel);
		hotelMap.put(hotel.getId(), hotel);
		ratingsMap.put(hotel.getId(), hotel.getRating());
```

```
		ratingServiceCommunicator.addRating(ratingsMap);
	}


	public Hotel getHotelById(String id) {

		if(ObjectUtils.isEmpty(hotelMap.get(id)))
		{
			throw new HotelNotFoundException("Hotel not found for id:
"+id);
		}
		Hotel hotel = hotelMap.get(id);
	long updatedRating=ratingServiceCommunicator.getRating(id);
		hotel.setRating(updatedRating);
		return hotel;
	}


	public List<Hotel> getAllHotels() {

		return hotelList;
	}


	public void deleteHotelById(String id) {
		Hotel hotel = getHotelById(id);
		hotelList.remove(hotel);
		hotelMap.remove(id);
		ratingServiceCommunicator.deleteRating(id);
	}


}
```

- Here, we are dealing with only four methods of **HotelService,** and the *updateHotel()* method has been removed. If you want, you can keep it and make changes accordingly, as stated in the document.

2. Step 2

Our model-class **Hotel** looks like this:

```
public class Hotel {

	private String id;

	@Size(min=3)
```

```
	private String name;

	@Min(value = 1)
	@Max(value = 10)
	private long rating;
	private String city;


	public Hotel(String id, String name, long rating, String city) {
		super();
		this.id = id;
		this.name = name;
		this.rating = rating;
		this.city = city;
	}

	@Override
	public String toString() {
		return "Hotel [id=" + id + ", name=" + name + ", rating=" + rating
+ ", city=" + city + "]";
	}


	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public long getRating() {
		return rating;
	}
	public void setRating(long rating) {
		this.rating = rating;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}

}
```

Change this model class **Hotel** to entity class for mapping it to a database table by doing the following:

- Add proper *@Entity* and *@Table* annotations for the **Hotel** class to change this model class to the entity.
- Add *@Id* and *@GeneratedValue(strategy = GenerationType.IDENTITY)* annotation for the field *private String id* to mark it as a unique id necessary for any entity class and to auto-increment the *id* values.

3. Step 3

Create a DTO class for the model class **Hotel** by doing the following:

- Create a **HotelRequest** class and provide the following attributes:
  private String name, private Long rating, and *private String city.*

- Generate getters and setters for the attributes.

- Create constructors for the attributes if required.

As you can see here, we are only using name, rating, and city attributes and encapsulating them in the DTO class (**HotelRequest**) to allow relevant data transfer between layers of the application, such as between the client and the server. Thus, this DTO class will enable us to expose only the necessary data to external clients or other parts of the application and reduce the overhead transfer of unnecessary data.

4. Step 4

- Add Lombok annotations to **HotelRequest** class to auto-generate getter setter for the attributes and constructors to reduce the boilerplate code.

- **This step is optional**, as the instructor will guide you through the implementation of adding Lombok annotations to the DTO class. However, you can try for yourself beforehand to have good practice before viewing the session.

5. Step 5

Our DTO class is now ready to be used. Therefore, we need to make specific changes in the **HotelController** class.

- Replace the argument **HotelRequest** objectentity of the *createHotel()* method of the **HotelController** class with the **HotelRequest** object.

- Pass the **HotelRequest** object as the argument to the *createHotel()* method of the **HotelService** class.

Previous **HotelController** class

```
@RestController
@RequestMapping("/hotel")
public class HotelController {

	@Autowired
	HotelService hotelService;

	@PostMapping("/create")
	public void createHotel(@Valid @RequestBody Hotel hotel, BindingResult
bindingResult)
	{
		if(bindingResult.hasErrors())
		{
			throw new RuntimeException("Request Not Valid");
		}
		hotelService.createHotel(hotel);
	}
//no changes below
```

As you can see, we initially used the direct **Hotel** entity class. Now, we are replacing the **HotelRequest object** as a client request body, which acts as a Data Transfer Object (DTO) specifically designed to represent the data received in an HTTP request. It will help prevent unnecessary exposure of our **Hotel** entity class to the client.

6. Step 6

After the **HotelController** class, we are now supposed to make changes accordingly in the **HotelService** class.

- Change the argument of the createHotel() method in the **HotelService** class from the **Hotel** object to the **HotelRequest** object.
- Map the data from the HotelRequest object to the Hotel entity object.
- Pass this **Hotel** Entity object to the *save()* method of the **HotelRepository** class.

7. Step 7

- Create a **HotelRepository** interface which will act as the Data Access Layer.
- Extend the **HotelRepository** interface with **JpaRepository** with proper type parameters.

8. Step 8

- Change the ‘application.properties’ file to ‘application.yml’ to make the file more readable, as the yml configuration provides proper indentation and serialization.
- Add configuration code for connecting with MySql Database.

```
spring:
  datasource:
    url:jdbc:mysql://localhost:3306/hotel
    username: root
    password: root@123
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate.ddl-auto: update
server:
  port: 8082
```

Please find the Final code of the **Hotel Application** provided below for reference: Link.