BSc (Hons) in Information Technology
Specializing in Software Engineering
Year 3 - 2021
SE3040 – Application Frameworks
Lab 09 – Spring Boot + MongoDB

1. Add Spring Boot Starter Mongo DB dependency.
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

2. Make sure MongoDB is running on localhost port 27017.
3. Add following configurations to connect to MongoDB to applications.properties file.
```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=posts
```

4. Add an interface to the 'domain' package named PostDataAdapter.java. This will invert the DB dependency for the Post.java class. Add a method to save. It should accept and return a Post.java instance.
```
public interface PostDataAdapter {

    Post save(Post post);

}
```

5. Add a package named 'dal'. Add another sub-package named 'adapter' Add a class named PostDataAdapterMongoImpl.java to the package created.
6. Add another two sub-packages called 'model' and 'repository' to package 'dal'.
7. Now add the following Mongo data model to 'model' package to represent Post.java data. Please note that I have introduced a new property 'postedDate' to the Post.java.
```
@Document ("posts")

public class PostModel {

    @Id

    private String id;

    private String name;

    private String description;

    private LocalDateTime postedDate;


    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 String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public LocalDateTime getPostedDate() {
        return postedDate;
    }

    public void setPostedDate(LocalDateTime postedDate) {
        this.postedDate = postedDate;
    }
}
```

This class is annotated with @Document to mark it as a Mongo document to Spring. Also, ID is marked with @Id annotation. The creation of the unique ID will be handed over to Mongo.

8. Add a class named PostMongoRespority.java to

```
@Repository
public interface PostMongoRepository extends MongoRepository<PostModel, String> {
}
```

Repositories are a convenient way to access DB. By default, Spring provides all basic operations in a repository. Customizations can be added using methods in a defined pattern which will be converted to Mongo queries by Spring.

9. Add following implementation to PostDataAdapterMongoImpl.java class.

```
@Component
public class PostDataAdapterMongoImpl implements PostDataAdapter {

    private final PostMongoRepository repository;

    @Autowired
    public PostDataAdapterMongoImpl(PostMongoRepository repository) {
        this.repository = repository;
    }

    @Override
    public Post save(Post post) {
        PostModel postModel = new PostModel();
        postModel.setName(post.getName());
        postModel.setDescription(post.getDescription());
        postModel.setPostedDate(post.getPostedDate());
        postModel = repository.save(postModel);
        post.setId(postModel.getId());
        return post;
    }
}
```

10. In PostApi.java autowire PostDataAdapter in the constructor. And change the addPost method body as follows.

```
public Post addPost(Post post) {
    post.setPostedDate(LocalDateTime.now());
    post = postDataAdapter.save(post);
    return post;
}
```

11. Test the endpoint. New entry should be added to the database. If the posts collection exists on the database, make sure to remove that because due to the change in Id field there maybe empty fields in newly added entries.

12. Add a method to PostDataAdapter.java to return all Posts.

```
List<Post> getAll();
```

13. Implement the method in PostDataAdapterMongoImpl.java

```
@Override
public List<Post> getAll() {
    List<PostModel> postModels = repository.findAll();
    List<Post> posts = new ArrayList<>();
    for (PostModel postModel : postModels) {
        Post post = new Post();
        post.setId(postModel.getId());
        post.setName(postModel.getName());
        post.setDescription(postModel.getDescription());
        post.setPostedDate(postModel.getPostedDate());
        posts.add(post);
    }
    return posts;
}
```

14. Call this method in PostApi.java
15. Extend this to support Delete and Update operations. Once done remove the Map in PostApi.java class.

Following code can be used to update only the name and description fields. To use this MongoTemplate should be autowired to PostDataAdapterMongoImpl.java class. MongoTemplate is another mechanism for dealing with Mongo DB it's more raw level and more customizable.

```
@Override
public Post update(Post post) {
    PostModel postModel =
mongoTemplate.findAndModify(Query.query(Criteria.where("id").is(p
ost.getId())), new Update().set("name",
post.getName()).set("description", post.getDescription()),
PostModel.class);
    post.setPostedDate(postModel.getPostedDate());
    return post;
}
```

BSc (Hons) in Information Technology

Specializing in Software Engineering

Year 3 - 2021

SE3040 – Application Frameworks

Lab 09 – Spring Boot + MongoDB

Ref:

1. https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-mongodb
2. https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.core