Technische Universität München
Fakultät für Informatik

Exercise Sheet 10: Component-based Architectures

Welcome to the tenth week of Advanced Topics of Software Engineering. In this week's theoretical exercise, we will practice concepts from the component-based architectures with a focus on Java Enterprise Edition and Android.

Theoretical Exercise

Exercise 1: JEE and Android

Component-based development is the practice of "defining, implementing and composing loosely coupled independent components into systems". This practice enables the separation of concerns, low coupling, and high cohesion of newly developed systems. As an example, let us look at *Java Enterprise Edition* (JEE - formerly known as J2EE) and *Android*.

1. JEE architecture supports component-based development of multi-tier enterprise applications. It provides a programming model that improves development productivity, standardizes the platform for hosting enterprise applications, and ensures the portability of developed applications with an extensive test suite. Name four different JEE technologies (components). For each, specify the tier and the function that it is used for.
2. In the lecture you learned about the component-based reference architecture of Android, a Linux-based operating system (OS), and the most used mobile OS in the world. Mobile clients (e.g. Android applications) have become tremendously important in the past decade and having a basic understanding of the native mobile architecture is indispensable for every junior software engineer. Justify how Android architecture facilitates *portability*, *re-usability*, and *security* properties of a component.

Practical Exercise

In Exercise 3, we learned how to query MongoDB collection using the Spring Data MongoDB Repository. Alternatively, there are other techniques to query MongoDB from Spring, which leverage the features of the MongoDB Query engine. In this exercise, you will learn how to query the database using an `@Query` annotator. Additionally, to customize the query operation, we will create our database interactions by leveraging `MongoTemplate` and implementing the Mongo Repository.

Exercise 2: Query MongoDB Database using @Query

The `@Query` annotator enables us to query MongoDB database using the Mongo query language, which also supports projecting fields in a document. This is particularly helpful when we want to hide the value of specific fields in the returned object.

To demonstrate this capability, let us create a new attribute named in the `Project` collection: *startingYear* with type integer. Add the *startingYear* in the constructor of the `Project`, and create the getter and setter methods for it. Afterward, create new projects with their starting year using the `CommandLineRunner`, or add the year value into the existing projects in your database.

In the `ProjectRepository`, add the following query to extract projects which start before a given year:

```
    public List<Project> findByStartingYearLessThanEqual(int year, Sort sort);
```

If you call this query as below, you should receive a list of projects in your system that start in or before the year 2015, sorted in ascending order.

```
    projectRepository.findByStartingYearLessThanEqual(
        2015,
        Sort.by(Sort.Direction.ASC, "startingYear")
    );
```

However, if you print the `project` in the query result, you will see all of its properties, including the `_id`. Suppose we want to hide this `_id` value from the world, then the following `@Query` instruction can help us remove the `_id` field from the result:

```
    @Query(fields = "{ '_id' : 0 }")
    public List<Project> findByStartingYearLessThanEqual(int year, Sort sort);
```

Now, suppose that you want to filter a project search by name and starting year. The Spring Repository function name for this query is:

```
    public Project findByNameAndStartingYearLessThanEqual(String projectName, int year);
```

Using `@Query`, an equivalent query is written as:

```
    @Query(value = "{ 'name' : ?0, 'startingYear': { $lte: ?1 } }")
    public Project findProjectByNameAndYear(String name, int year);
```

Exercise 3: Query Sub-documents Using @Query and @Aggregation

So far, we only query attributes of a root document, but how can we query sub-documents? For example, if we model students as sub-documents of a class document, how can we find which classes are attended by a student with a given name? Let us find out in this exercise.

Assume we want to model investors who funded our projects. How could we find which projects are invested by an investor with a given name? To simplify this example, we will not create a separate investor collection.

Create an `Investor` class which has two properties *name* of type String and *fund* of type double. In the Project class, create an *investors* attribute as a List of `Investor` objects.

```
    public class Investor {
        String name;
        double funding;

        public Investor() {}

        public Investor(String name, double funding) {
            this.name = name;
            this.funding = funding;
        }
        // getters and setters
    }

    //*********************************

    @Document(collection = "projects")
    public class Project {
        @MongoId
        private String id;

        @Indexed(unique = true)
        @NonNull
        private String name;

        @NonNull
        private int startingYear;

        private List<Investor> investors;

        protected Project() {}

        public Project(String name, int startingYear, List<Investor> investors) {
            this.name = name;
            this.startingYear = startingYear;
            this.investors = investors;
        }
        // getters and setters
    }
```

Now, add investors and their funds into our projects such that an investor appears in at least two projects, and that project has at least two investors. For instance, in the below example, *Banana Corp.* invests in two projects: ASE Smartphone and ASE Smart Farm. ASE Smartphone is also funded by *Ruby Group*:

```
    projectRepository.save(
        new Project("ASE Smartphone",
                        2015,
                        Arrays.asList(
                                new Investor("Ruby Group", 3000000),
                                new Investor("Banana Corp.", 2000000)
                        )
        )
    );
    projectRepository.save(
        new Project("ASE Smart Farm",
                        2010,
                        Arrays.asList(new Investor("Banana Corp.", 4999999))
        )
    );
    // Create more projects
```

To find an investor that matches a given name, you may think of the following Mongo Repository functions:

```
    public List<Project> findByInvestors_name(String investorName);
```

This function will search for any property inside the `Project` that matches the "investors" pattern. The "_" sign is a delimiter to separate between an object and its attribute. In this case, we tell `Spring Repository` to search the property name of each element in the `investors` list, and find if any `investors.name` matches the given `investorName` search string.

Although this function returns the right project(s) that contains the investor, it also includes all other investors in that project(s). This means if you search for projects invested by Ruby Group, you will see ASE Smartphone. When you print out the list of investors in the `ASE Smartphone` project, you will see both Ruby Group and Banana Corp. However, if we only want the project information and the attribute of the investor that matches our search result, i.e, showing only Ruby Group and information about the ASE Project in the result, we need to hide other investors in the project. This can be done using the `@Query` Annotator like below:

```
    @Query(value = "{ 'investors.name': ?0 }",
            fields = "{ 'id': 1, 'name': 1, 'startingYear': 1, 'investors.$': 1 }")
    public List<Project> findByInvestors_name(String investorName);
```

In the fields, stating 'investors.$' includes all attributes of the investor that matches the query (investors.name). Therefore, only `Ruby Group` is displayed in the list of ASE Smartphone project along with the `id`, `name`, and `startingYear` attributes of the project. Nevertheless, suppose our `Project` has more than 20 fields, it is more optimal if we do not have to specify all 20 attributes in our `@Query` string. Mongo `@Aggregation`[1] can come to the rescue in this case.

MongoDB *Aggregation*[2] groups data records and computes the results using various operations (count, filter, map-reduce, sum, etc.). In Spring Boot, Aggregation can be called using the `Aggregation` annotator or a `Aggregation` and `Operation beans` such as `MatchOperation` or `ProjectOperation`. In the next step, you will use `Aggregation` to **remove investors of which name does not match the search**

[1] Aggregation Repository Methods: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongodb.repositories.queries.aggregation

[2] MongoDB Aggregation: https://docs.mongodb.com/manual/aggregation/

string out of the result. This means your result should contain only the `Project` name, startingYear, id, the name and funds of the investor that matches a given name **without hard coding these attributes in your aggregation pipeline**.

You can check the following materials to find clues in solving this problem:

1. Aggregation Pipeline: https://docs.mongodb.com/manual/aggregation/#aggregation-pipeline
2. Calling @Aggregation in Spring: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongodb.repositories.queries.aggregation
3. $filter Operation: https://docs.mongodb.com/manual/reference/operator/aggregation/filter/#mongodb-expression-exp.-filter

There are many solutions to handle this task, so please feel free to explore other possibilities. In case you are stuck, and only when you are really stuck, you can check the last hint on the solution at https://www.tutorialspoint.com/mongodb-query-to-find-and-return-subdocument-with-criteria

Exercise 4: Sending Email using Google Mail Service

One of the key features in the ASE Delivery system is sending emails to notify customers about the status of their orders. In this exercise, we will learn how to send email in Spring using a Google Mail Service. We do not restrict you to use Google Mail in your project, so feel free to use another approach for sending emails, as long as it is ethical, legal, and secure.

First, if you do not have a Gmail account already, sign up for an account at gmail.com. Afterward, Enable 2-step verification in the account. Then register a password for our application to use the Gmail service as follow:

1. Click on your avatar on the top right corner, select **Manage your Google Account**.
2. In the *Security* tab, navigate to **Signing in to Google** and click **App passwords**.
3. Type your Gmail Account password to enter the App passwords interface. Create a new password by clicking on **Select app**, choose **Other (custom name)**, and type the project name, e.g. Project.
4. Click **Generate** and you should see your App password under "Your app password for your device". Save this password in a place before you close the window.

Next, in the Project application, add the following dependency into your pom.xml file

```
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
```

Configure your email service with your email, password, and google mail service properties in the `application.yml` file:

```
spring:
    mail:
        protocol: smtp
        host: smtp.gmail.com
        port: 587
        username: your_email@gmail.com
        password: your_app_password
        properties:
          mail:
            smtp:
              auth: true
              starttls:
                enable: true
```

Next, create an email service to send a simple email with a recipient(s), subject, and content. You can see how to send an email using `SimpleMailMessage` at https://www.baeldung.com/spring-email#sending-email. Finally, the task of this exercise is to send an email when a project is created.