**Spring-Data-Masterclass**

**What is Spring-Data?**

Spring-Data a part of Spring family whose purpose is to easily access both relational database systems and NoSQL data stores.



**Brief Description**

In the above diagram, I represented an architecture diagram of spring-data connecting with respective databases.

* First layer is the base interface layer consists of **CrudRepository**, **PagingAndSortingRepository**.
* Second layer is that layer used for respective databases like **JpaRepository** used for **RDBMS**, **MongoRepository** used for **MongoDB**, **GraphRepository** used for **Neo4j DB**.
* All the above derived Repository Interface extends the Base Repository Interface as **JpaRepository** extends **CrudRepository**, **MongoRepository** extends **PagingAndSortingRepository** and **GraphRepsoitory** extends **PagingAndSortingRepository**.
* Now the final step is to define the Drivers and Dialect by which, from code it will connect to the respective databases.

**Why Spring Data?**

Introduction to Spring Data reduces a lot of the boilerplate code which we used to write earlier to make a JDBC Connection.

Earlier we used to write a DAO layer which will holds the business logic interacting with the Model and the Model connects with the database. In this DAO Layer we used to write everything like loading the Driver class as required, opening a connection, write a query, based on the query action either it inserts or updates or deletes or fetches the data from DB, converts the data from Model to an object which will be best fitted in the application and last of all to close the connection.

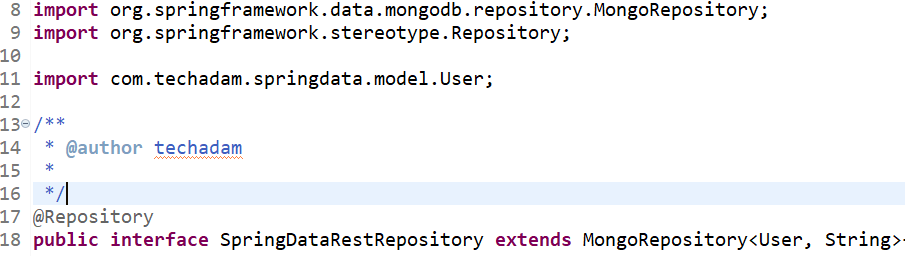
All these above activities are now taken care of Spring-Data if you use it in your application or project.

**Reasons to use Spring-Data**

There are 3 most important and valid reasons of using Spring-Data over the legacy implementation to connect to database.

1. **No-code Repositories** – One of the most popular persistence-related patterns is the **Repository Pattern**. It helps the developer to only focus on the implementation of business logic and hides the data store specific implementation details. It also helps not to write the repetitive standard CRUD operations for each entity.

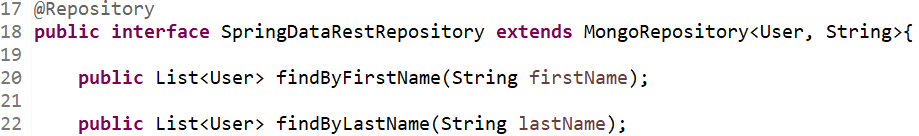
It provides a set of repository interfaces from where you will extend to define a specific repository for one of your entities.



Here, you can see that I have used **MongoRepository** to perform the CRUD operations on the entity – **User**. Because in this example, I have used MongoDB as the underlying database. **MongoRepository** extends **PagingAndSortingRepository**. By this implementation you will get the default implementations of methods like:

* To insert, update and delete one or more User
* To find one or more User by their primary keys
* To count, get and delete all Users
* To check if a User with a given primary key already exists or not

1. **Reduce boilerplate code** – Spring-Data provides a default implementation for each method like read or write operations. That means you don’t have to write the basic implementation logic for read and write operations excluding the business logic.
2. **Generated Queries** – Spring-Data generates the database queries based on their method names. You just need to define a method in your repository interface with a name that starts with **findBy**. Spring parse the method names and create a JPQL query, sets the provided method parameters as bind parameter values, execute the query, returns the result.



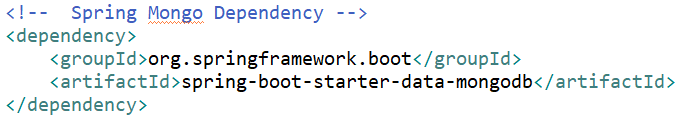
Here you can see that this Repository Interface extends MongoRepository and I have attached the User entity in the concept of Generics as the 1st argument and in the 2nd argument you provide the data type of the Primary Key as defined in your entity. In my case the **data type** of the **primary key** used in **User** entity is **String**.

**Using Spring Data with Spring Boot and use MongoDB as database**

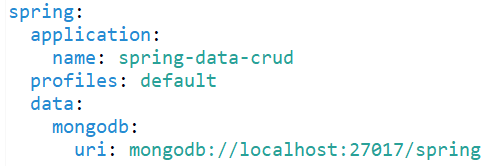
We can say now that the introduction of Spring-Data makes the implementation of persistent layer much easier to a developer. For this let’s make our hands dirty by creating a project. I have created a Demo project named as **spring-data-masterclass** uploaded in my personal GitHub account [spring-data-masterclass](https://github.com/c86amik/spring-data-masterclass).

One can clone the project and can test locally in their system. All the steps that are required to download the project from GitHub and running it locally are mentioned in the **README.md** file of **spring-data-masterclass** repository.

1. **Brief Description** – I have developed a project named as **spring-data-masterclass** using Spring Boot, Maven, Java and Mongo as underlying database. In this project I mainly focus on implementing the Spring-data with Spring-REST service. It consists of a Controller annotated with @RestController annotation, a Repository extending the MongoRepository and the CRUD operations to the Mongo Database. To test the Services, I have used **Postman**.
2. **Software Used** – Software required to develop this project.
   1. **Spring Tool Suite-4.7.0-RELEASE** – If latest version is available then download that one
   2. **Apache Maven 3.6.3** – If latest version is available then download that one
   3. **Java 8** – Not less than Java8
   4. **Git 2.27.0** – Latest version as available
   5. **Mongo** **4.2.8** – Mongo DB Server
   6. **Robo 3T 1.3.1** – Mongo DB Client to see the data as persisted in DB. You can download any other client as available in the market
   7. **Postman v8.3.0** – To test the REST Service
3. **Project Components** – The project that I develop in support to the spring-data with Mongo concept is a Maven project. And I have used one dependency as **spring-boot-starter-data-mongodb**. The reason behind this as I have used **Mongo** as the underlying database.

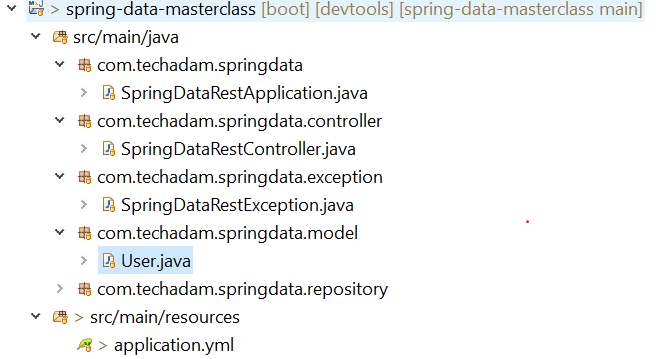


In the next step I have configured the database connection in **application.yml** file. You can use either **application.properties** or **application.yml** file. A wonderful concept using Spring Boot and Spring Data JPA is that they handle the default configuration for you, you only need to override the parameters you want to change.



If you use application.properties then the name of the property is **spring.data.mongodb.uri=mongodb://localhost:27017/spring**. I have used application.yml file so it follows the indentation level.

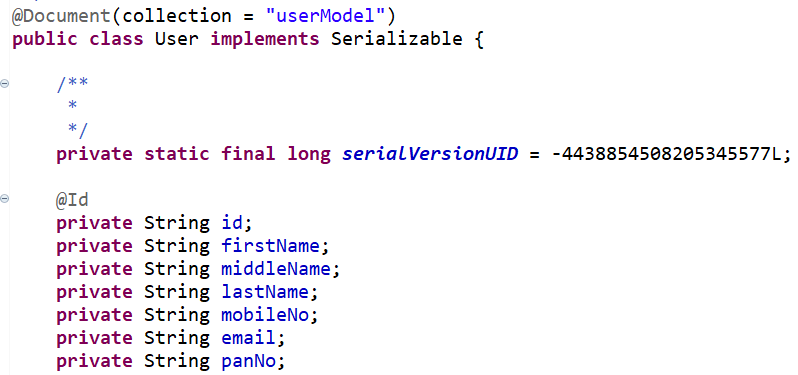
1. **Structure of the Project** – To use Spring Data JPA and its repositories in your project you need to structure it in a right way.



Spring Boot expects that all repositories are located in a sub-packages of the class annotated with [**@SpringBootApplication**](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/autoconfigure/SpringBootApplication.html).

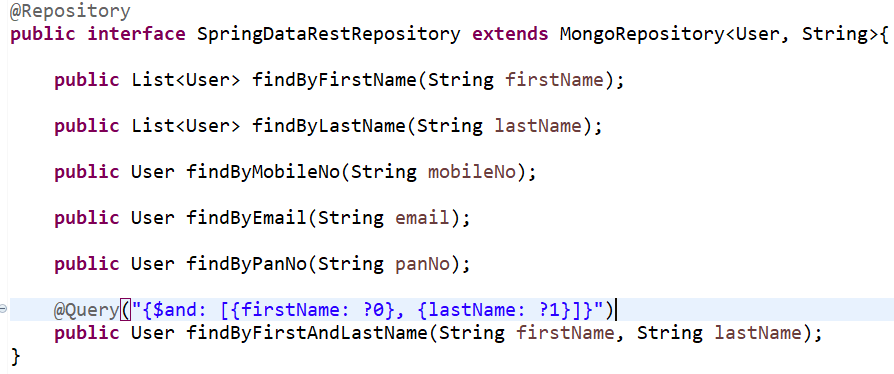
1. **Spring-Data Implementation** – To implement Spring-Data I have used one of its repositories named as spring-boot-starter-data-mongodb. For this you need to define one Model class which will interact with the database and a Repository interface which you will use for the CRUD operations with the database.

Let’s look to the **Model** class first



Here you can see that the Model class name is **User** annotated with [**@Document**](https://docs.spring.io/spring-data/data-mongodb/docs/current/api/org/springframework/data/mongodb/core/mapping/Document.html)**(collection=”userModel”)**. It means that whenever you persist the **User** model data the name of the collection in Mongo will be **userModel**. In Mongo, a table is referred to as a collection. And for this collection the primary key is **id** as the property is annotated with **@Id** annotation.

Now we will investigate the **Repository** interface



The name of the Repository interface is **SpringDataRestRepository** extending the [**MongoRepsoitory**](https://docs.spring.io/spring-data/data-mongodb/docs/current/api/org/springframework/data/mongodb/repository/MongoRepository.html) interface of **org.springframework.data.mopngodb.repository** package. And I have passed the model class **User** and the data type of the primary key used in the **User** class as **String**.

On extending this interface the advantage what I got is it provides me the default methods as **findById(id)**, **save()**, **findAll()**, **delete()** methods as

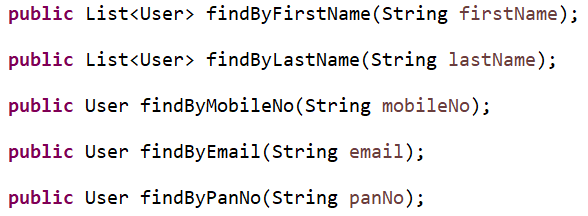
findById(userId) - 

save() - 

findall() – 

delete() - 

Above all this, it provides some other methods which you just need to define in the interface. Like



Here you can see that the methods are defined by following a naming convention as **findBy** then the name of the **property** as defined in the **User** model. And the best thing is that you don’t need to provide any implementation for this. As, spring-data itself provides that.

But if you defined a method as provided below:



After that when you run the application you will get the following error as:



It means that you have defined a method without following the spring data naming convention.

Here I have also used one custom query example to find the record based on firstName and lastName. I have used one annotation as [**@Query**](https://docs.spring.io/spring-data/data-mongodb/docs/current/api/org/springframework/data/mongodb/repository/Query.html)**("{$and: [{firstName: ?0}, {lastName: ?1}]}")**.

In this project I have used [**@RestController**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) and use the Spring-Rest annotations as [**@GetMapping**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/GetMapping.html), [**@PostMapping**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html), [**@PutMapping**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PutMapping.html) and [**@DeleteMapping**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/DeleteMapping.html).This project has no UI implementation. If anyone try to integrate UI then one can clone the repository and add the UI components there.

1. **Testing the application** – In this application as there is no UI component, I have used **Postman** to test the REST endpoint as defined in the Controller class.
   * 1. You clone the application from GitHub and set up the application locally in any one of the IDEs like **Spring Tool Suite (STS)** or **Eclipse**.
     2. Right click on the application,
     3. Click the Option **Run As**
     4. Select the option **Spring Boot App**.
     5. It will start the application in the port no **7111**.
     6. Now in Postman you can test the endpoints.

Below I have provided the list of endpoints as available in this application.

* 1. **Get all Users** – [localhost:7111/allUsers](http://localhost:7111/allUsers)
  2. **Save an User** – [localhost:7111/saveUser](http://localhost:7111/saveUser), in the Body provide the JSON Object. A dummy JSON object is provided.

{

"firstName" : <<firstName>>,

"middleName" : <<middleName>>,

"lastName" : <<lastName>>,

"mobileNo" : <<mobileNo>>,

"email" : <<email>>,

"panNo" : <<panNo>>

}

* 1. **Update an User** – [localhost:7111/updateUser/{id}](http://localhost:7111/updateUser/%7bid%7d), in the Body again provides the JSON data as

{

“id” : <<Mongo\_ID>>,

"firstName" : <<firstName>>,

"middleName" : <<middleName>>,

"lastName" : <<lastName>>,

"mobileNo" : <<mobileNo>>,

"email" : <<email>>,

"panNo" : <<panNo>>

}

* 1. **Delete an User** – [localhost:7111/deleteUser/{id}](http://localhost:7111/deleteUser/%7bid%7d)
  2. **Get User By First Name** – [localhost:7111/getUserByFirstName/{firstName}](http://localhost:7111/getUserByFirstName/%7bfirstName%7d)
  3. **Get User By Last Name** – [localhost:7111/getUserByLastName/{lastName}](http://localhost:7111/getUserByLastName/Chakraborty)
  4. **Get User By Mobile No** – [localhost:7111/getUserByMobileNo/{mobileNo}](http://localhost:7111/getUserByMobileNo/%7bmobileNo%7d)
  5. **Get User By Email** – [localhost:7111/getUserByEmail/{email}](http://localhost:7111/getUserByEmail/%7bemail%7d)
  6. **Get User By Pan No** – [localhost:7111/getUserByPan/{panNo}](http://localhost:7111/getUserByPan/%7bpanNo%7d)
  7. **Get User By Name** - [localhost:7111/getUserByName/{firstName}/{lastName}](http://localhost:7111/getUserByName/%7bfirstName%7d/%7blastName%7d)