

#22-Project Using Spring, Loading data in H2

Since there is currently no product data in the database, we must deal with the controller.

```
1 package com.example.ecom_proj.repo;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7
8
9 @Repository
10 public interface ProductRepo extends JpaRepository<Product, Integer> {
11
12 }
13
14
15
16
17 @Autowired
18 ProductService service;
19
20 @RequestMapping("/")
21 public String greet() {
22     return "Hello Aliens";
23 }
24
25 @GetMapping("/products")
26 public List<Product> getAllProducts() {
27     return service.getAllProducts();
28 }
29 }
30
```

Let's create a method called `getAllProducts()` in the `ProductController` controller class. It will return a list of products that have been annotated with `@GetMapping` and have an endpoint of `"/products."` However, the implementation logic will be in the Service layer class, which we will access through the Service object.

`ProductService` is a service class that is declared with the `@Service` annotation, and it

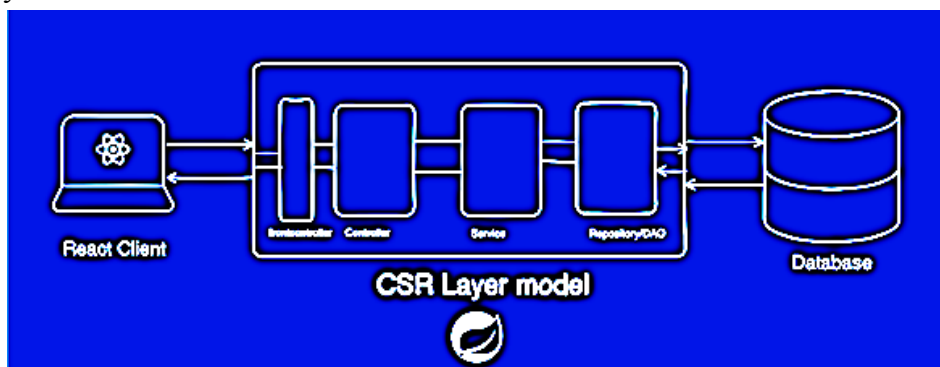
```
1 package com.example.ecom_proj.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.example.ecom_proj.model.Product;
9 import com.example.ecom_proj.repo.ProductRepo;
10
11 @Service
12 public class ProductService {
13
14     @Autowired
15     ProductRepo repo;
16
17     public List<Product> getAllProducts() {
18         return repo.findAll();
19     }
20 }
```

provides the implementation for `getAllProducts()`. However, we know that we will be getting the actual implementation method from the repository/dao layer, so we will be creating an interface `ProductRepo` in the Repository layer that is annotated with `@Repository`. This will extend the JPA repository interface, and it requires two type arguments: one is an integer, which is our primary key (id), and the other is a class.

using field injection in the controller for the service class `ProductService` that is annotated with `@Autowired` and in the service class for the reference of the `ProductRepo` interface. Constructor is the injection that comes highly recommended; you can select any other injection.

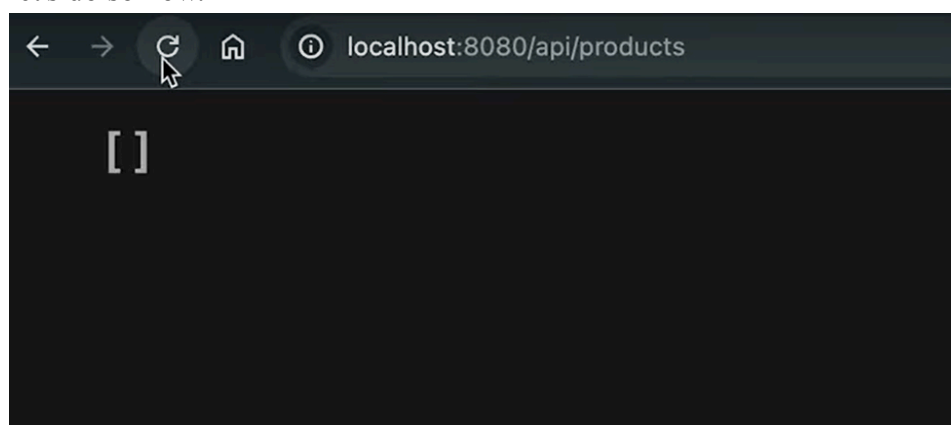
The CSR layer model operates as you can see by looking at how the layers are related to one another and how each layer depends on the others.

CSR Layer Picture



Now let's run the application to see how it functions and what kind of output we can expect.

As you can see, the output is an empty data array. Since we haven't added any data to the database, let's do so now.

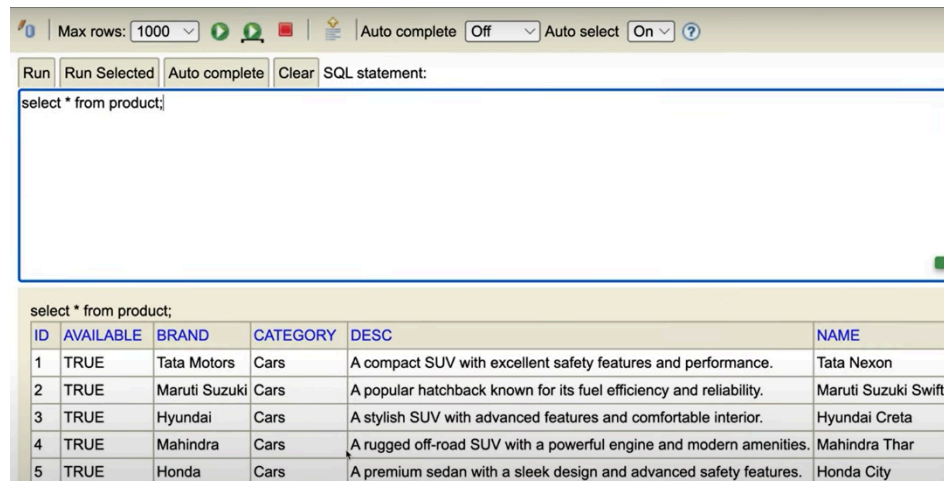


A sample SQL data set has been created and is intended to be inserted into the H2 database; however, the database is in memory, meaning that the data disappears from the database upon application reloading. We'll search for it so that we can find a solution.

Here, we need to add an annotation to the model class.

@GeneratedValue(strategy=GenerationType.IDENTITY) to make the insert query we added not null and auto-generated. Then, after firing the insert query in the h2-console, fire the select query to confirm that the query data was successfully added to the database.

H2 console O/P:

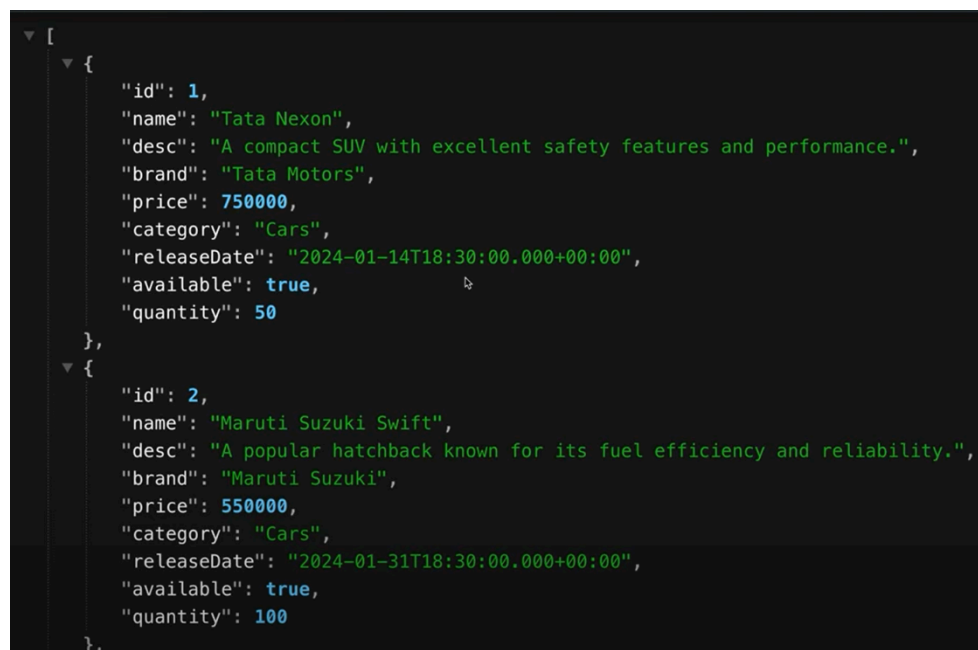


The screenshot shows the H2 console interface. At the top, there are controls for 'Max rows' (set to 1000), 'Auto complete' (set to Off), and 'Auto select' (set to On). Below these are buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear'. The 'SQL statement:' field contains the query 'select * from product;'. Below the query field, the results are displayed in a table format.

ID	AVAILABLE	BRAND	CATEGORY	DESC	NAME
1	TRUE	Tata Motors	Cars	A compact SUV with excellent safety features and performance.	Tata Nexon
2	TRUE	Maruti Suzuki	Cars	A popular hatchback known for its fuel efficiency and reliability.	Maruti Suzuki Swift
3	TRUE	Hyundai	Cars	A stylish SUV with advanced features and comfortable interior.	Hyundai Creta
4	TRUE	Mahindra	Cars	A rugged off-road SUV with a powerful engine and modern amenities.	Mahindra Thar
5	TRUE	Honda	Cars	A premium sedan with a sleek design and advanced safety features.	Honda City

Browser window O/p:

Let's restart the application. However, before doing so, we need to address an issue with our database, which can be temperamental. To ensure that our data is consistently available, we



The screenshot shows a JSON array of product objects. Each object contains fields for id, name, desc, brand, price, category, releaseDate, available, and quantity.

```
[
  {
    "id": 1,
    "name": "Tata Nexon",
    "desc": "A compact SUV with excellent safety features and performance.",
    "brand": "Tata Motors",
    "price": 750000,
    "category": "Cars",
    "releaseDate": "2024-01-14T18:30:00.000+00:00",
    "available": true,
    "quantity": 50
  },
  {
    "id": 2,
    "name": "Maruti Suzuki Swift",
    "desc": "A popular hatchback known for its fuel efficiency and reliability.",
    "brand": "Maruti Suzuki",
    "price": 550000,
    "category": "Cars",
    "releaseDate": "2024-01-31T18:30:00.000+00:00",
    "available": true,
    "quantity": 100
  }
]
```

should add a file called data.sql under the resources folder. This file will contain the SQL queries for data insertion, and every time we reload the application, the data will be populated from this file.

spring.jpa.defer-datasource-initialization=true

This property will defer the data source initialization until the data from the data.sql file has been inserted.

Since JPA tends to search for data before it is created and inserted, we need to delay this searching process. To accomplish this, we must ensure that the data is properly inserted before any search is performed. To achieve this, add a new property in the application.properties file:

spring.jpa.defer-datasource-initialization=true

This property will defer the data source initialization until the data from the data.sql file has been inserted.

If this property is not added, a mapping error will occur, meaning that the system will look for query data before the creation table of the necessary data. We made use of this property, but the issue arose after the Spring Boot version 2.7.

Let's now restart or reload the application to see that we are getting the product list on the browser side and the h2 side without having to manually add queries because it is added through our sql file.

But we are not getting output on our React application side. What could be the issue, though, and let's decode it in our upcoming chapter.

Until then, happy coding and keep learning 😊 🐾