

Spring controller

```

@Controller
public class HomeController {
    @RequestMapping("/data")
    public String displayHome(Model m){

        m.addAttribute("data",1234);
        return "hhomepage";
    }
}

```

Above request handler returns view with data

CrudReposiory

JPARepository

MongoRepository

RestController

```

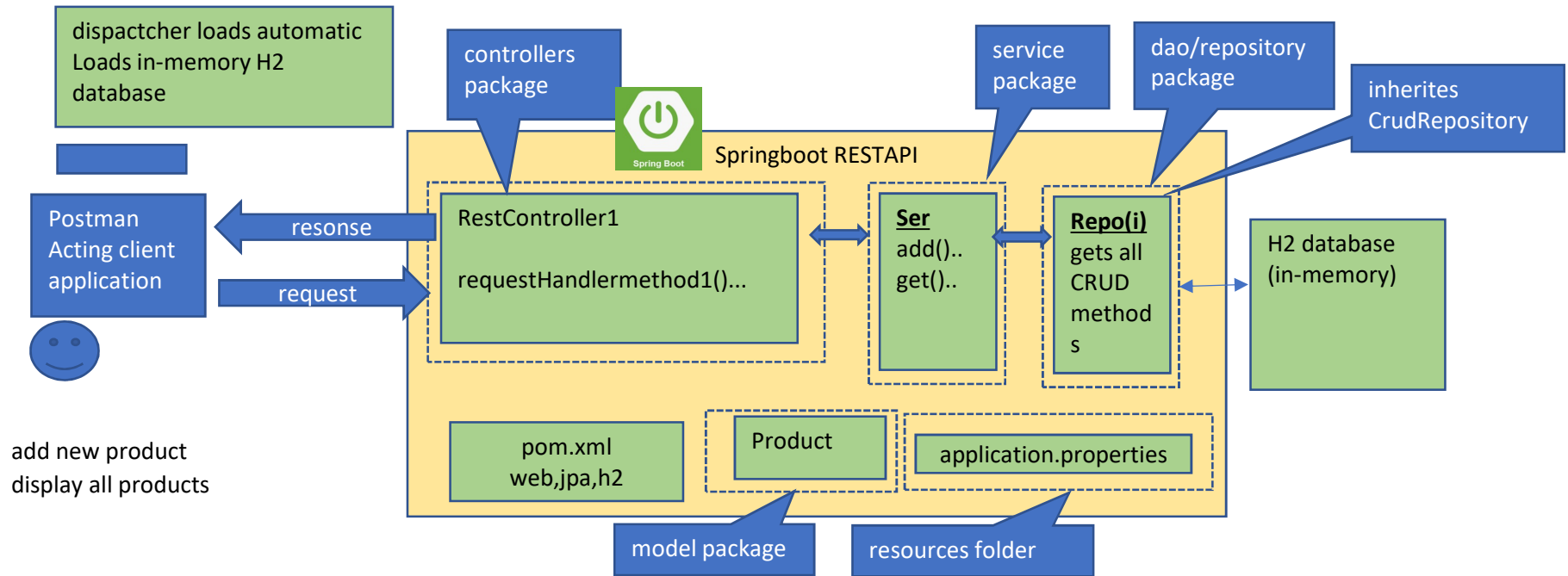
class MyTime{
}

@RestController
public class TimeController{
    @GetMapping("/time")
    public ResponseEntity<?> getTime(){
        MyTime t = new MyTime(1,2,3);
        return new ResponseEntity<>(t,HttpStatus.OK);
    }
}

```

ResponseEntity holds data + status

Demo
SpringBoot application
RESTful API



add new product
display all products

Step 1 Create springboot application by adding required dependencies

web

data

h2



Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M5) ☐ 2.7.5 (SNAPSHOT) ☒ 2.7.4
☐ 2.6.13 (SNAPSHOT) ☐ 2.6.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

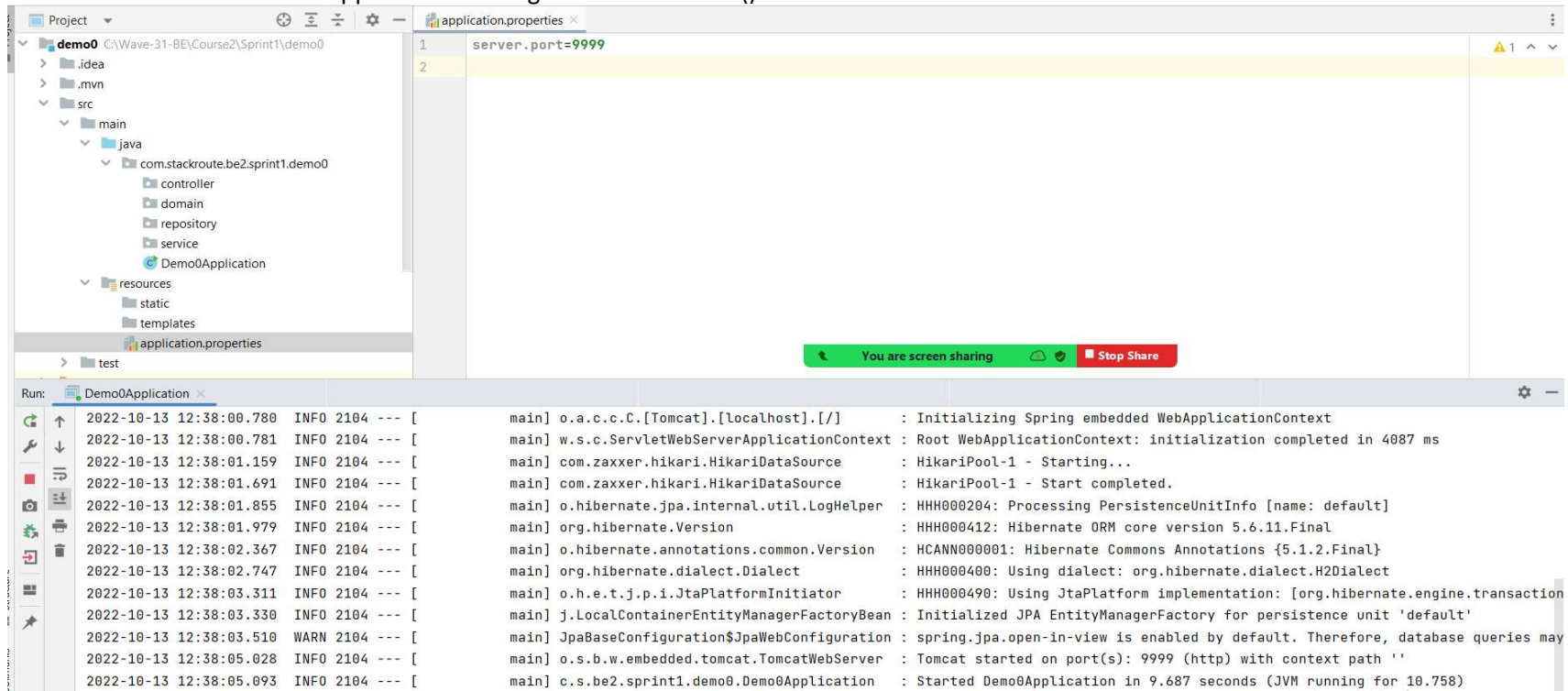
H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

You are screen sharing Stop Share

download the created project

- Step 2 Extract downloaded project
 open in intellij
 create required packages
 Makesure application running with default main()



- Step 3 Define model class

```
6     @Entity
7     public class Product {
8         4 usages
9         @Id
10         private String productId;
11         4 usages
12         private String category,name;
13         4 usages
14         private double price, rating;
15         public Product() {
```

- Step 4 Create repository interface by providing domain entity and identifier type in domain entity
 model class, primary key datatype in model class
 model class/domain entity Product
 identifier type in domain entity String

```
1 package com.stackroute.be2.sprint1.demo0.repository;
2
3 import com.stackroute.be2.sprint1.demo0.domain.Product;
4 import org.springframework.data.repository.CrudRepository;
5
6 // @Repository (not required)
7 public interface ProductRepository extends CrudRepository <Product, String>{
8     /*
9         findAll()
10        findById()
11        save();
12        delete();
13        ..
14        ..
15        .
16        .
17    */
18 }
19
20 // Spring IOC creates an IMPL class in spring container for above
21 // repository with all definitions for abstract methods
22 /*
23 class ProductRepositoryImpl implements ProductRepository{
24
25     findAll() { ... get all products ... }
26     save(product) { .. to save product... }
27 }
28 */
```

insert/save/add()
return object

- Step 5 Define service layer with two methods
 addProduct()
 getProducts()

 declare interface
 define impl class

```

7      public interface ProductService {
8          public abstract Product addProduct(Product product);
9          public abstract List<Product> getProducts();
10     }

10     @Service
11     public class ProductServiceImpl implements ProductService {
12
13         2 usages
14         @Autowired
15         private ProductRepository productRepository;
16
17         @Override
18         public Product addProduct(Product product) {
19             return productRepository.save(product);
20         }
21
22         @Override
23         public List<Product> getProducts() {
24             return (List<Product>)productRepository.findAll();
25         }
26     }

```

Step 6 Define controller

```

14     @RequestMapping("/productapp/v1")
15     @RestController
16     public class ProductController {
17
18         1 usage
19         @Autowired
20         private ProductService productService;
21
22         @GetMapping("/product")
23         public ResponseEntity<?> getProductdata(){
24             // return RE object with productdata+status
25             List<Product> products = productService.getProducts();
26             return new ResponseEntity<>(products, HttpStatus.OK);
27         }
28     }

```

- Step 7 enable inmemory h2
- application.properties
 - spring.h2.console.enabled=true

```
1 server.port=9999
2 spring.h2.console.enabled=true
```

How to login to in memory h2 database

```
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 3175 ms
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed
main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:bb650b1c-9eae-4cac-bb0e-05afae7baa57'
main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.11.Final
main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. E
```

if port number is xxxx
then H2 URL is

<http://localhost:xxxx/h2-console>

JDBC URL

changes for every execution

localhost:9999/h2-console/login.jsp?sessionId=30072e995895d2802d44ab56f1f1c4a1

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:bb650b1c-9eae-4cac-bb0e-05afae7baa57

User Name: sa

Password:

Connect Test Connection

insert few records

```
insert into product values('PRD0001','Electronics','LED TV',12345,4.1);
insert into product values('PRD0002','Stationary','Notebook',75,4.2);
insert into product values('PRD0003','Furniture','Chair',1500,3.5);
insert into product values('PRD0004','Toy','Thomas Train - Diecast',657,4.9);
insert into product values('PRD0005','Grocery','Surf',56,4.7);
```

Run	Run Selected	Auto complete	Clear	SQL statement:
SELECT * FROM PRODUCT				
SELECT * FROM PRODUCT;				
PRODUCT_ID	CATEGORY	NAME	PRICE	RATING
PRD0001	Electronics	LED TV	12345.0	4.1
PRD0002	Stationary	Notebook	75.0	4.2
PRD0003	Furniture	Chair	1500.0	3.5
PRD0004	Toy	Thomas Train - Diecast	657.0	4.9
PRD0005	Grocery	Surf	56.0	4.7

(5 rows, 2 ms)

How to check application

make sure inmemory h2 running and few records inserted

Open Postman

The screenshot shows a Postman interface with a GET request to `http://localhost:9999/productapp/v1/product`. The response is a JSON array of product objects. Callouts highlight the following elements:

- request type**: GET
- request URL**: `http://localhost:9999/productapp/v1/product`
- request header**: The Headers tab is active, showing an empty list.
- Response status**: Status: 200 OK
- Response data**: The JSON response body, which is an array of 5 product objects.

```
[{"productId": "PRD0001", "category": "Electronics", "name": "LED TV", "price": 12345.0, "rating": 4.1}, {"productId": "PRD0002", "category": "Stationary", "name": "Notebook", "price": 75.0, "rating": 4.2}, {"productId": "PRD0003", "category": "Furniture", "name": "Chair", "price": 1500.0, "rating": 3.5}, {"productId": "PRD0004", "category": "Toy", "name": "Thomas Train - Diecast", "price": 657.0, "rating": 4.9}, {"productId": "PRD0005", "category": "Grocery", "name": "Surf", "price": 56.0, "rating": 4.7}]
```

How to add record

define request handler in controller

```
26 // http://localhost:9999/productapp/v1/product [POST]
27 @PostMapping("/product")
28 public ResponseEntity<?> addProduct(@RequestBody Product prd){
29     Product result = productService.addProduct(prd);
30     return new ResponseEntity<>(result,HttpStatus.CREATED);
31 }
```

test in postman

The screenshot shows the Postman interface with a POST request to `http://localhost:9999/productapp/v1/product`. The request body is a JSON object: `{ "productId": "PRD0006", "category": "Electronics", "name": "Iron box", "price": 1100, "rating": 4.3 }`. The response status is `201 Created` with a time of `507 ms` and size of `263 B`. The response body is a JSON object: `{ "productId": "PRD0006", "category": "Electronics", "name": "Iron box", "price": 1100.0, "rating": 4.3 }`. Blue callout boxes with arrows point to the following elements:

- request type**: Points to the `POST` method dropdown.
- request URL**: Points to the URL `http://localhost:9999/productapp/v1/product`.
- request body**: Points to the JSON body of the request.
- response data**: Points to the JSON body of the response.
- response status**: Points to the `Status: 201 Created` text.

What client application DO

UI

- send request (post/get/put/...)
- send request with data
- receive response

Postman is doing same without UI

BASE_URL

<http://localhost:9999/productapp/v1>

to get all product records

GET <http://localhost:9999/productapp/v1/product>

// <http://localhost:9999/productapp/v1/product> [GET]

`@GetMapping("/product")`

`public ResponseEntity<?> getProductdata(){`

`// return RE object with productdata+status`

`List<Product> products = productService.getProducts();`

`return new ResponseEntity<>(products, HttpStatus.OK);`

`}`

`@Override`

`public List<Product> getProducts() {`

`return (List<Product>)productRepository.findAll();`

`}`

7

8

`public interface ProductRepository extends CrudRepository <Product, String> {`

`}`

to add new product

POST <http://localhost:9999/productapp/v1/product>

// <http://localhost:9999/productapp/v1/product> [POST]

`@PostMapping("/product")`

`public ResponseEntity<?> addProduct(@RequestBody Product prd){`

`Product result = productService.addProduct(prd);`

`return new ResponseEntity<>(result, HttpStatus.CREATED);`

`}`

`@Override`

`public Product addProduct(Product product) {`

`return productRepository.save(product);`

`}`

7

8

`public interface ProductRepository extends CrudRepository <Product, String> {`

`}`

to update product

PUT <http://localhost:9999/productapp/v1/product>

// <http://localhost:9999/productapp/v1/product> [PUT]

`@PutMapping("/product")`

`public ResponseEntity<?> updateProduct(@RequestBody Product prd){`

`Product result = productService.updateProduct(prd);`

`return new ResponseEntity<>(result, HttpStatus.OK);`

`}`

`@Override`

`public Product updateProduct(Product p) {`

`return productRepository.save(p);`

`}`

to delete product

DELETE

<http://localhost:9999/productapp/v1/product/PRD0001>

Template variable
/ Path variable

```
// http://localhost:9999/productapp/v1/product/PRD0001 [DELETE]
```

```
@DeleteMapping("/product/{productId}")
```

```
public ResponseEntity<?> deleteProduct(@PathVariable String productId){  
    productService.deleteProduct(productId);  
    return new ResponseEntity<>( body: "deleted",HttpStatus.OK );  
}
```

```
@Override
```

```
public void deleteProduct(String pid) {  
    productRepository.deleteById(pid);  
}
```

to get product by id

GET

<http://localhost:9999/productapp/v1/get-product-by-id/PRD0001>

Template variable
/ Path variable

```
// http://localhost:9999/productapp/v1/get-product-by-id/PRD0001 [GET]
```

```
@GetMapping("/get-product-by-id/{productId}")
```

```
public ResponseEntity<?> getProductById(@PathVariable String productId){  
    Product result = productService.getProductById(productId);  
    return new ResponseEntity<>(result,HttpStatus.OK );  
}
```

```
@Override
```

```
public Product getProductById(String pid) {  
    if(productRepository.findById(pid).isPresent())  
    {  
        return productRepository.findById(pid).get();  
    }  
    else {  
        return null;  
    }  
}
```

Iterable<>	findAll()	gives all records
Object	save()	saves / updates record
void	deleteById(id)	deletes record
Optional	findById(id)	