# Xây dựng ứng dụng web với Java

## Module: Building web applications with Java

&  18

**Biên soạn: Vũ Duy Linh**

vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

# Content

Chap 1. Java Spring web technology overview

Chap 2. Spring Core 6.x

Chap 3. Spring Data JPA, Spring Data Rest 3.x

Chap 4. Spring Boot 3.x (Security) MVC with Thymeleaf 3.x

Chap 5. Backend RESTful API (Security) with Spring Boot

Chap 6. Frontend using React.Js 18.x

# Building web applications with Java

## Chap 1. Java Spring web technology overview

**Biên soạn: Vũ Duy Linh**

vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

- JDK 17


- Spring Tool Suite 4.x (included Java Eclipse and M2E Maven)


- MySQL/SQL Server/MongoDB compass


- Node.js v18.x

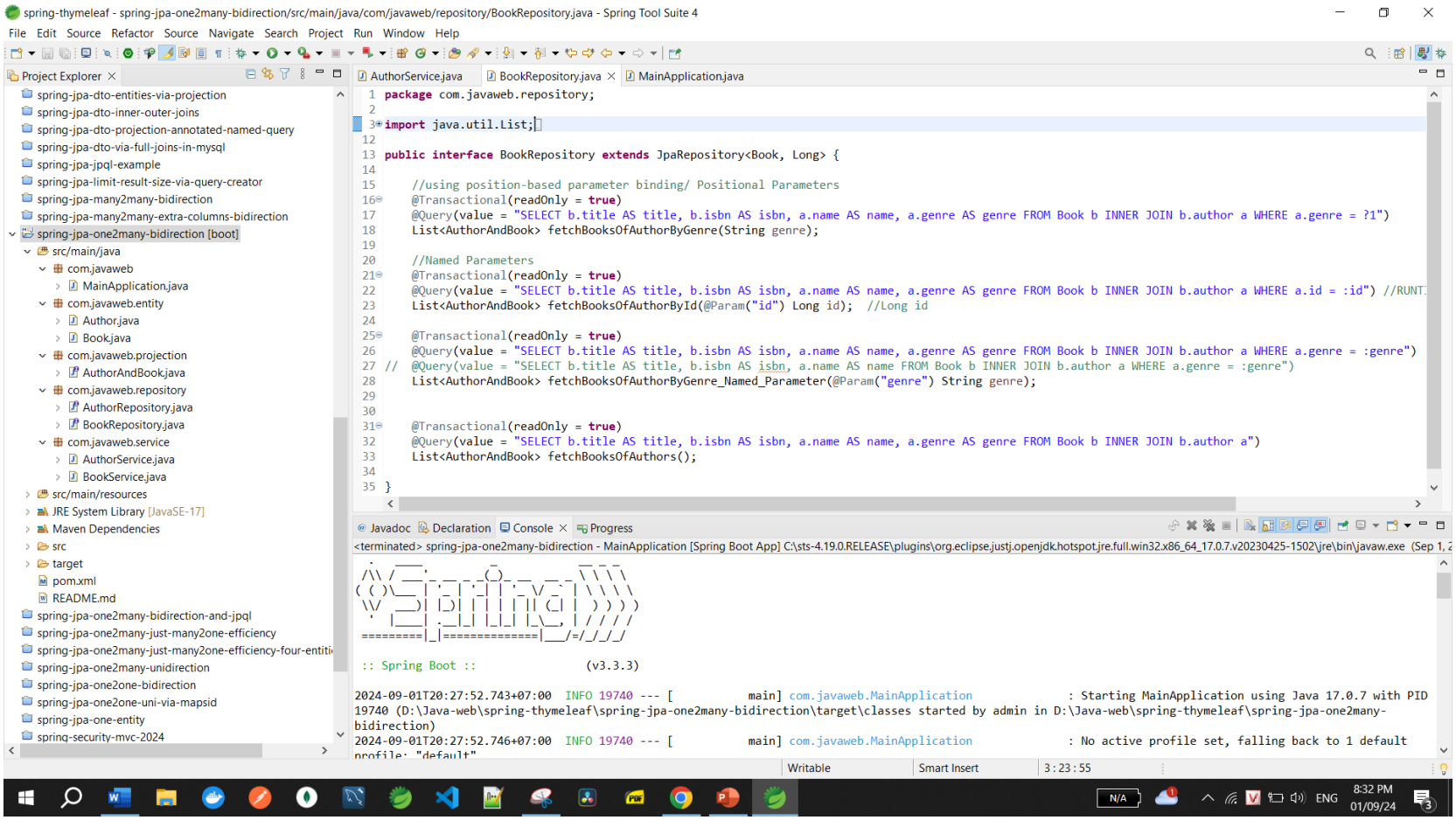

- ReactJS 18.x


- Visual code

- Postman / SpringDoc OpenAPI (Swagger 3)/Curl


# Java Spring Web Development Environment ◆ Chap 1

- https://spring.io/tools
- Spring web development environment is ready to study



- Overview
  - Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
  - We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.
  - If you're looking for information about a specific version, or instructions about how to upgrade from an earlier release, check out the project release notes section on our wiki.
  - https://docs.spring.io/spring-boot/docs/current/reference/html/

- Features
  - Create stand-alone Spring applications
  - Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
  - Provide opinionated 'starter' dependencies to simplify your build configuration
  - Automatically configure Spring and 3rd party libraries whenever possible
  - Provide production-ready features such as metrics, health checks, and externalized configuration
  - Absolutely no code generation and no requirement for XML configuration

- https://spring.io/projects/spring-boot

# Spring Boot 3.3.3

OVERVIEW | LEARN | SUPPORT | SAMPLES

| Branch | Initial Release | End of Support | End Enterprise Support * |
|---|---|---|---|
| 3.4.x | 2024-11-21 | 2025-11-21 | 2027-02-21 |
| 3.3.x | 2024-05-23 | 2025-05-23 | 2026-08-23 |
| 3.2.x | 2023-11-23 | 2024-11-23 | 2026-02-23 |
| 3.1.x | 2023-05-18 | 2024-05-18 | 2025-08-18 |
| 3.0.x | 2022-11-24 | 2023-11-24 | 2025-02-24 |
| 2.7.x | 2022-05-19 | 2023-11-24 | 2025-08-24 |

- Overview
  - The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.
  - A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.
- Support Policy and Migration
  - For information about minimum requirements, guidance on upgrading from earlier versions and support policies, please check out the official Spring Framework wiki page
- Features
  - Core technologies: dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.
  - Testing: mock objects, TestContext framework, Spring MVC Test, WebTestClient.
  - Data Access: transactions, DAO support, JDBC, ORM, Marshalling XML.
  - Spring MVC and Spring WebFlux web frameworks.
  - Integration: remoting, JMS, JCA, JMX, email, tasks, scheduling, cache and observability.
  - Languages: Kotlin, Groovy, dynamic languages.

# Spring Framework 6.1.12

OVERVIEW | LEARN | SUPPORT

| Branch | Initial Release | End of Support | End Enterprise Support * |
|---|---|---|---|
| 6.2.x | 2024-11-14 | 2026-08-31 | 2027-12-31 |
| 6.1.x | 2023-11-16 | 2025-08-31 | 2026-12-31 |
| 6.0.x | 2022-11-16 | 2024-08-31 | 2025-12-31 |
| 5.3.x | 2020-10-27 | 2024-08-31 | 2026-12-31 |
| 5.2.x | 2019-09-30 | 2021-12-31 | 2023-12-31 |

More

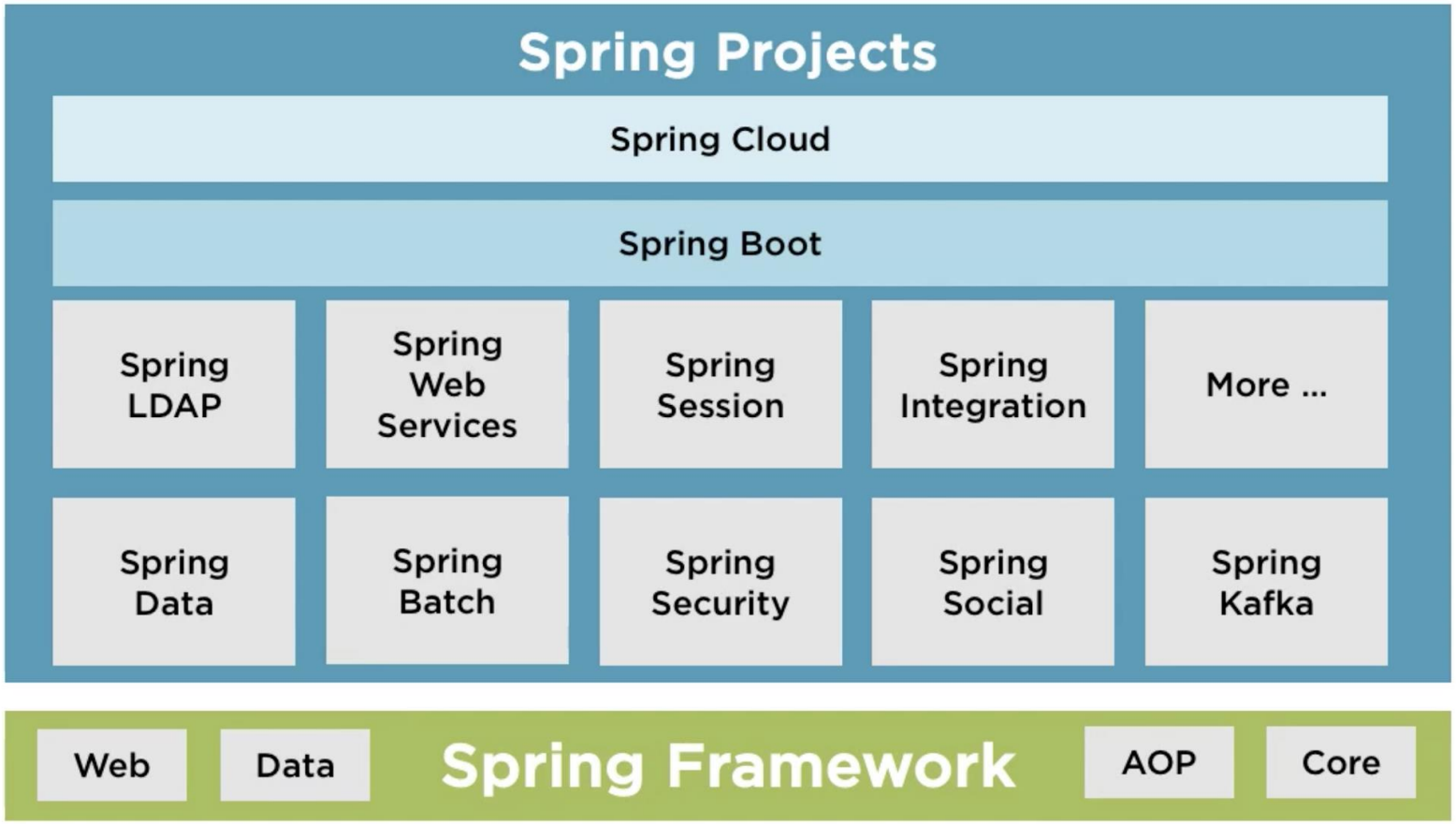
## JDK Version Range

- Spring Framework 6.2.x: JDK 17-25 (expected)
- Spring Framework 6.1.x: JDK 17-23
- Spring Framework 6.0.x: JDK 17-21
- Spring Framework 5.3.x: JDK 8-21 (as of 5.3.26)

## Java/Jakarta EE Versions

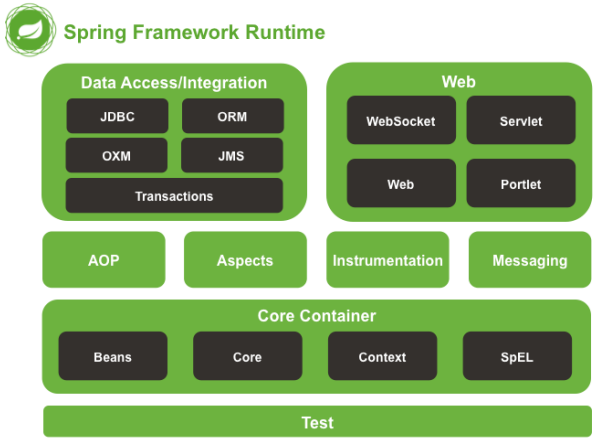
- Spring Framework 6.2.x: Jakarta EE 9-11 (jakarta namespace)
- Spring Framework 6.1.x: Jakarta EE 9-10 (jakarta namespace)
- Spring Framework 6.0.x: Jakarta EE 9-10 (jakarta namespace)
- Spring Framework 5.3.x: Java EE 7-8 (javax namespace)

- More details: https://spring.io/why-spring



# Building web applications with Java

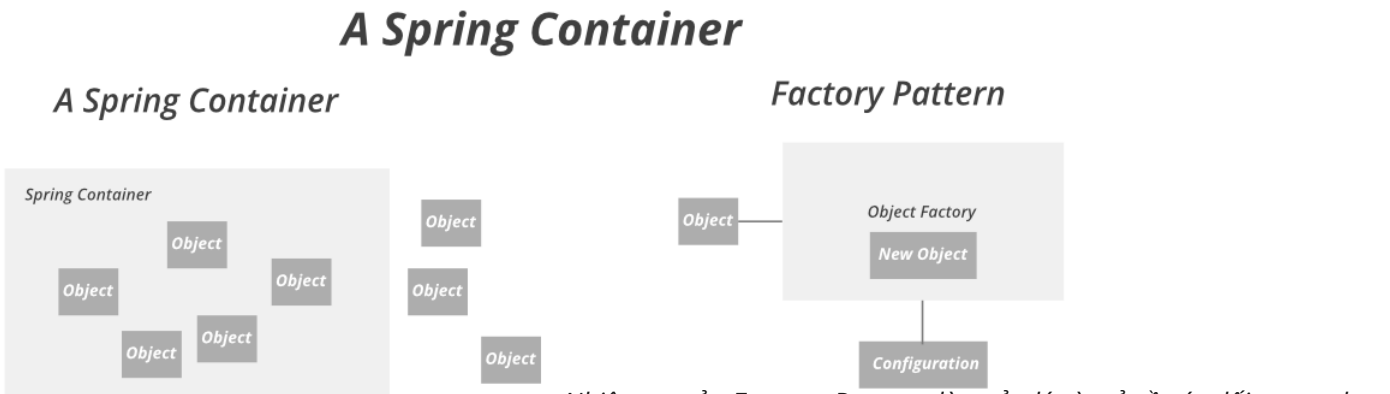
## Chap 2. Spring core 6.x




**Biên soạn: Vũ Duy Linh**

vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

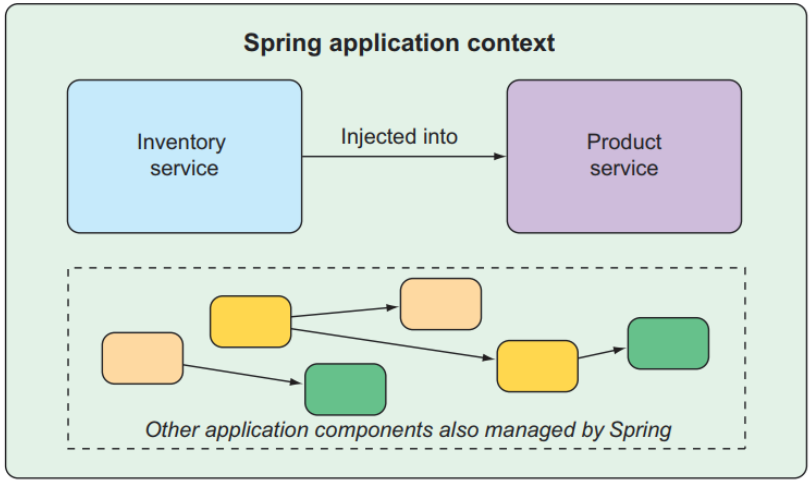
- **Container:** org.springframework.context.ApplicationContext interface represents the **Spring IoC (Inversion of Control) container** and is responsible for instantiating, configuring, and assembling the beans.
- In Spring, **the objects** that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.




*Nhiệm vụ của **Factory Pattern** là quản lý và trả về các đối tượng theo yêu cầu, giúp cho việc khởi tạo đối tượng một cách linh hoạt hơn.*

- The IoC container is used to manage and configure Plain Old Java Objects (POJOs). IoC is also known as *dependency injection* (DI)
- Core Technologies: https://docs.spring.io/spring-framework/reference/core.html , https://docs.spring.io/spring-framework/reference/

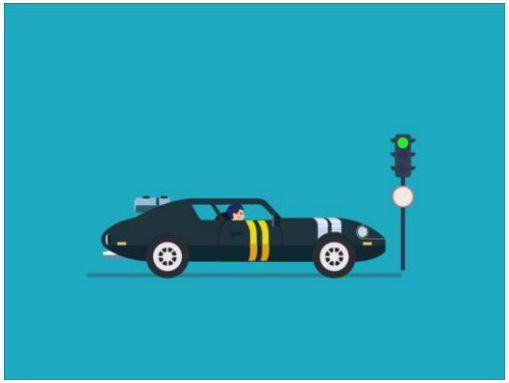
- Application components are managed and injected into each other by the Spring application context

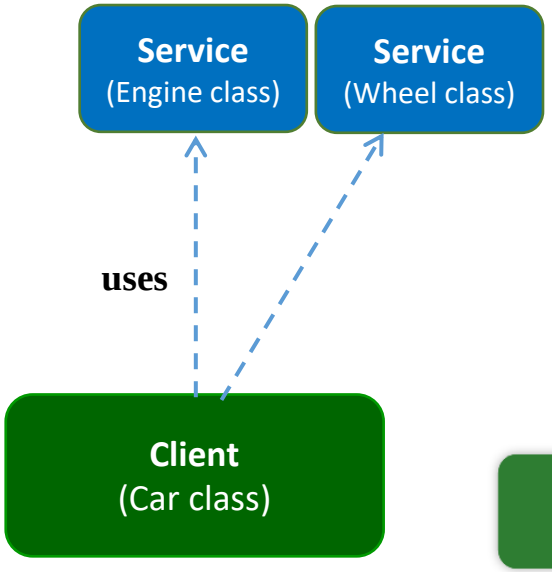



- DI is a software development technique where we can create objects that depend on other objects. DI helps the interaction between classes, but at the same time keeps the classes independent.
  - It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.
  - The container then injects those dependencies when it creates the bean

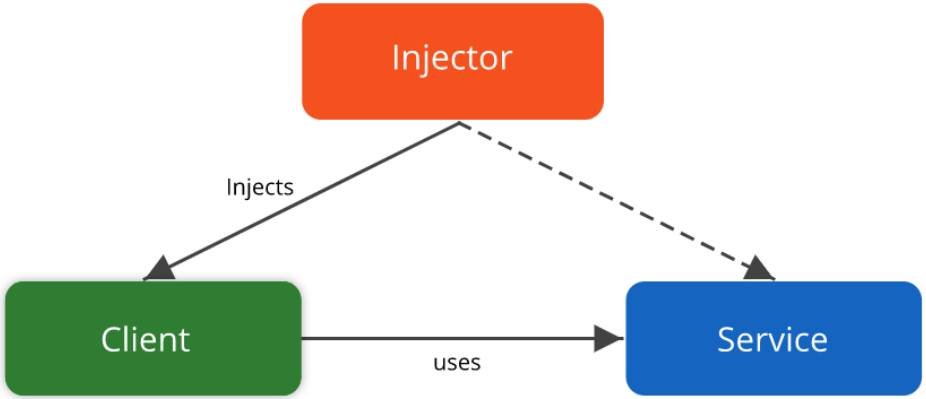
- There are three types of classes in DI, with scenarios:
  - A **Service** (or dependency) is a class that can be used.
  - The **Client** is a class that uses dependency.
  - The **Injector** passes the dependency (**Service**) to the dependent class (**Client**).

  ⇒DI in Spring ensures **loose-coupling** between the classes.
  - Ex: class **Car** has a dependency on **Engine** and **Wheel** classes






Dependency **in OOP**.
Ex: Car uses Engine and Wheel




Dependency Injection **in Spring**.
Ex: Inject Engine and Wheel into Car

- The three types of classes in DI are shown in the following diagram:
  - **Service** classes

```java
public class Engine {
    private Integer number;
    //four-stroke or two-stroke
    private String stroke;
    private String manufacturer;

    public Engine() {

    //...
```

```java
public class Wheel {
    private Integer size;
    private String manufacturer;

    public Wheel() {
        super();
    }
    //...
```

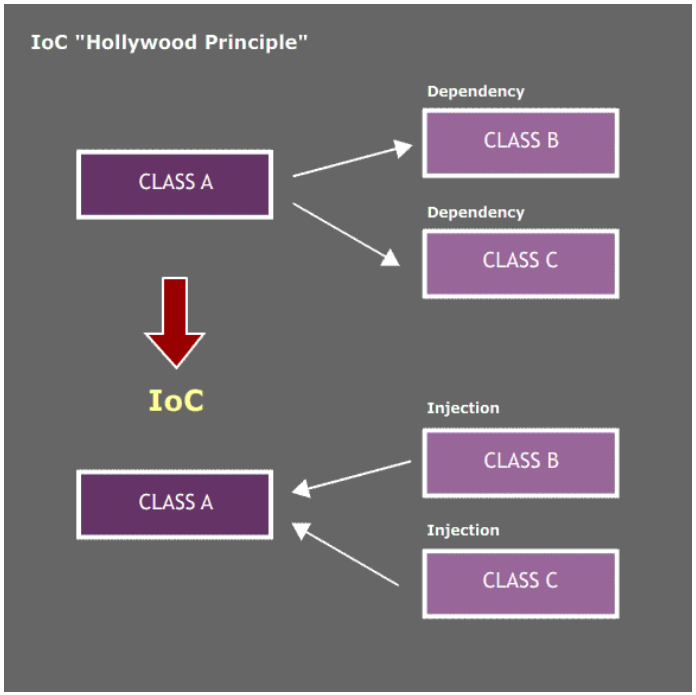
  - **Client** class: Constructor injection, Setter injection, Field injection

```java
public class Car {
    private Engine engine;  //to inject into Car
    private Wheel wheel;    //to inject into Car

    private String model;
    private String manufacturer;

    public Car(Engine engine, Wheel wheel) {
        this.engine = engine;
        this.wheel = wheel;
    }
    //...
```

```java
public class Car {
    private Engine engine;
    private Wheel wheel;

    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    public void setWheel(Wheel wheel) {
        this.wheel = wheel;
    }
    //...
```

```java
public class Car {
    @Autowired
    private Engine engine;

    @Autowired
    private Wheel wheel;

    private String model;
    private String manufacturer;

    //...
```

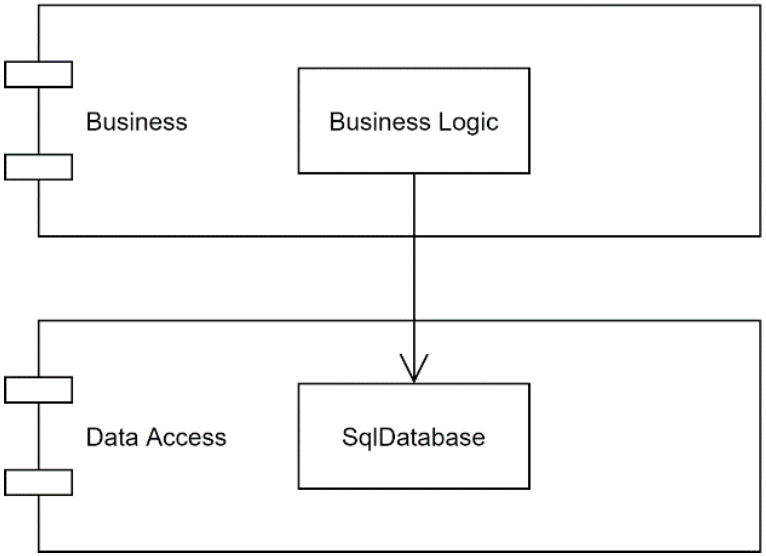
- Inversion of Control
  - IoC is a programming principle which inverts the flow of control as compared to traditional control flow.
  - Spring framework uses dependency injection to form the IoC. , which is a form of inversion control.  IoC is a kind of **Hollywood principle**, don't call us, we'll call you.



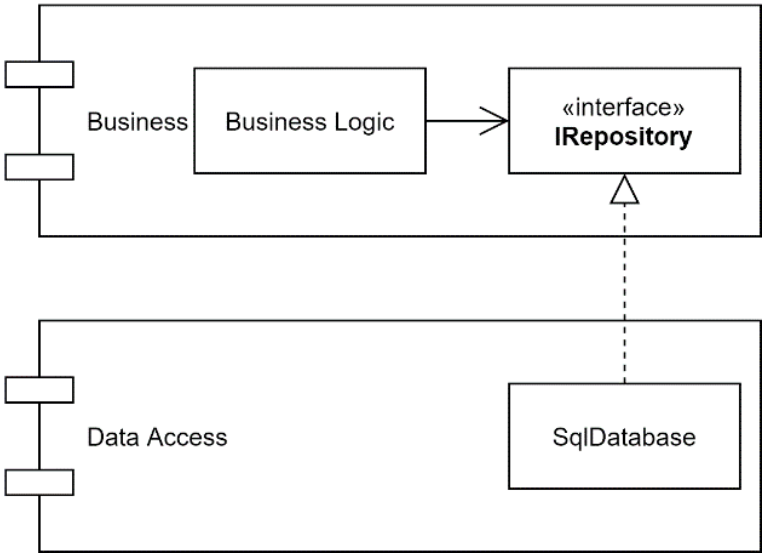

https://medium.com/analytics-vidhya/dependency-injection-concept-of-spring-framework-d9c3688005f8

- Jakarta EE vs Spring Boot

**Without Dependency Inversion**



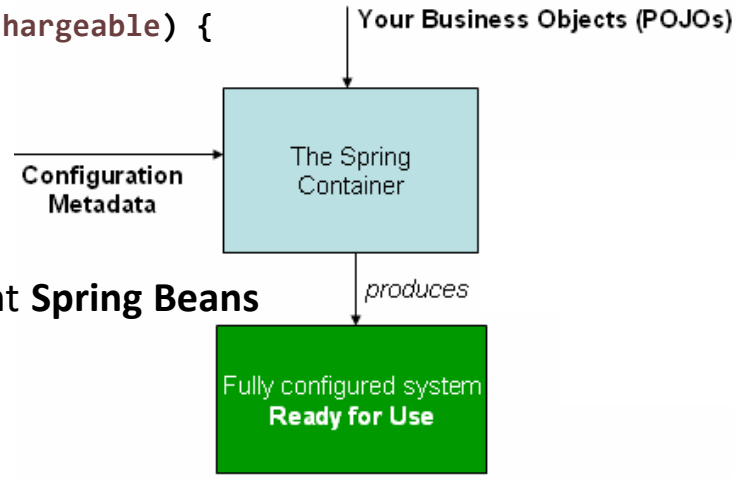

**With Dependency Inversion**




```
@Service
public class LecturerService {
    @Autowired
    private CourseRepository courseRepo;

    @Autowired
    private LecturerRepository lecturerRepo;
    //...
}
```

- Spring Beans and its life cycle
  - Plain Old Java Objects (POJOs): Use to represent entity **in JPA**/Hibernate

```
public class Battery {
    private String name;    private double price;    private boolean rechargeable;
    public Battery(String name, double price) {
        super(name, price);
    }
    public boolean getRechargeable() {
        return rechargeable;
    }
    public void setRechargeable(boolean rechargeable) {
        this.rechargeable = rechargeable;
    }
   //…
}
```




  - Plain Old Java Object (POJOs): Use to represent **Spring Beans**

```
@Component
public class MathComponent {
        public int add(int x, int y) {
                return x + y;
        }

        public int subtract(int x, int y) {
                return x - y;
        }
}
```

- Spring Beans: In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container.
- A Spring IoC container manages one or more beans. These beans are created with the configuration metadata that you supply to the container.

```
@Configuration  //class level
public class ShopConfiguration {
    @Bean  // method level
    public Product aaa() {
        Battery p1 = new Battery("AAA", 2.5);
        p1.setRechargeable(true); return p1;
    }
    @Bean
    public Product cdrw() {
        Disc p2 = new Disc("CD-RW", 1.5);
        p2.setCapacity(700); return p2;
    }
}
```

- Bean naming conventions: The convention is to use the standard Java convention for instance field names when naming beans. That is, bean names start with a lowercase letter, and are camel-cased from then on. Ex: accountManager, loginController,…

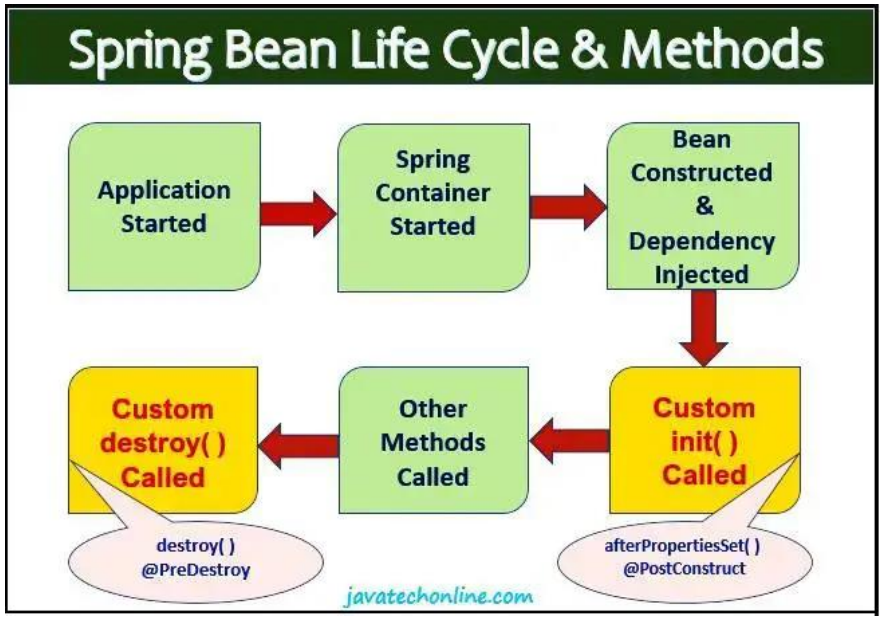
- **Demo:** D:\Java-web\spring-thymeleaf**spring-core-recipe-2dot2**

SpringCoreRecipe2dot2Application.java

```java
package com.recipes;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
//@ComponentScan(basePackages = "com.recipes.component")
public class SpringCoreRecipe2dot2Application {

    public static void main(String[] args) {
        // ApplicationContext represent the Spring IoC container.
        ApplicationContext context = SpringApplication.run(SpringCoreRecipe2dot2Application.class, args);

        Product aaa = context.getBean("aaa", Product.class);
        Product cdrw = context.getBean("cdrw", Product.class);
        System.out.println(aaa); // Product: AAA, price: 2.5
        System.out.println(cdrw); // Product: CD-RW, price: 1.5

        System.out.println("=========== List all Spring Beans =============== ");
        String[] beans = context.getBeanDefinitionNames();
        for (String bean : beans)
            System.out.println(bean);

        MathComponent ms = context.getBean(MathComponent.class);
        System.out.println("Addition of 3 and 4 = " + ms.add(3,4));

        AdminComponent adminComponent = context.getBean("admin", AdminComponent.class);
        adminComponent.login();

//      MathComponent ms = (MathComponent) context.getBean("mc");
//      System.out.println("Addition of 3 and 4 = " + ms.add(3,4));

//      AdminComponent adminComponent2 = context.getBean("adminComponent", AdminComponent.class);
//      adminComponent2.login();
    }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.2)

2023-08-05T21:47:48.332+07:00  INFO 14088 --- [           main] c.r.SpringCoreRecipe2dot2Application
2023-08-05T21:47:48.338+07:00  INFO 14088 --- [           main] c.r.SpringCoreRecipe2dot2Application
2023-08-05T21:47:48.932+07:00  INFO 14088 --- [           main] c.r.SpringCoreRecipe2dot2Application
Product: AAA, price: 2.5
Product: CD-RW, price: 1.5
========== List all Spring Beans ===============
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
springCoreRecipe2dot2Application
org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory
loginComponent
mathComponent
shopConfiguration
admin
aaa
cdrw
org.springframework.boot.autoconfigure.AutoConfigurationPackages
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration
propertySourcesPlaceholderConfigurer
org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration
mbeanExporter
objectNamingStrategy
mbeanServer
org.springframework.boot.context.properties.ConfigurationPropertiesBindingPostProcessor
org.springframework.boot.context.internalConfigurationPropertiesBinder
```

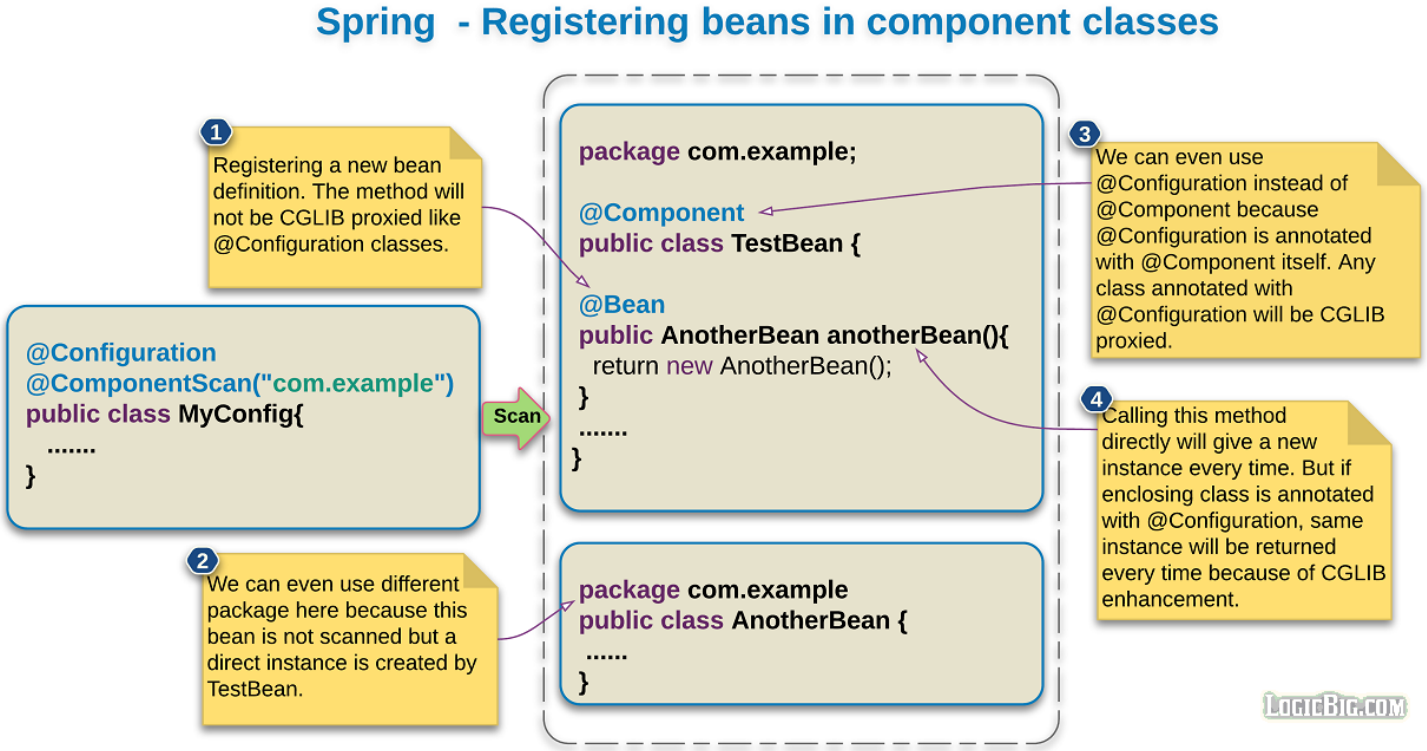
- Spring Bean life cycle




- Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.
- Spring @PostConstruct Annotation
- Spring @PreDestroy Annotation

- Some annotations
  - **@Configuration**: Used to indicate that a class declares one or more @Bean methods. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.
  - **@Bean**: Indicates that a method produces a bean to be managed by the Spring container. This is one of the most used and important spring annotation. @Bean annotation also can be used with parameters like name, initMethod and destroyMethod.
  - Spring 2.5 introduces further stereotype annotations: **@Component**, **@Service**, and **@Controller**. @Component is a generic stereotype for any Spring-managed component.
  - **@Component**: Indicates that an annotated class is a "component". Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
  - **@ComponentScan**: to specify the packages that we want to be scanned. @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages.

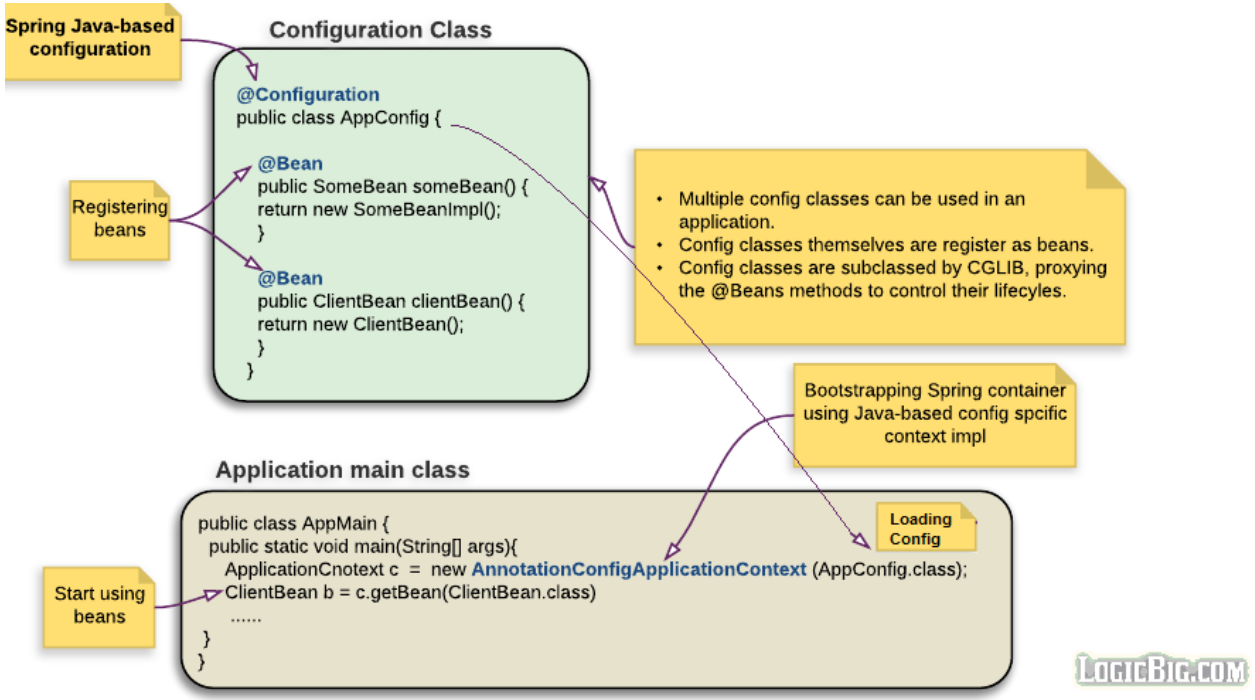
- Registering Spring beans within @Component classes: Classes annotated with any of the stereotype component annotations (@Component, @Service, @Repository etc), can also expose new bean definitions using @Bean annotation on their methods.




⇒**'@Configuration' is missing on a class defining Spring Beans** : Cảnh báo không nên sử dụng.

**Source:** https://www.logicbig.com/tutorials/spring-framework/spring-core/bean-definition-in-components.html

- Registering Spring beans based configuration @Configuration: (Only) using @Bean methods within @Configuration classes is a recommended approach of ensuring that 'full' mode is always used. This will prevent the same @Bean method from accidentally being invoked multiple times and helps to reduce subtle bugs that can be hard to track down when operating in 'lite' mode.

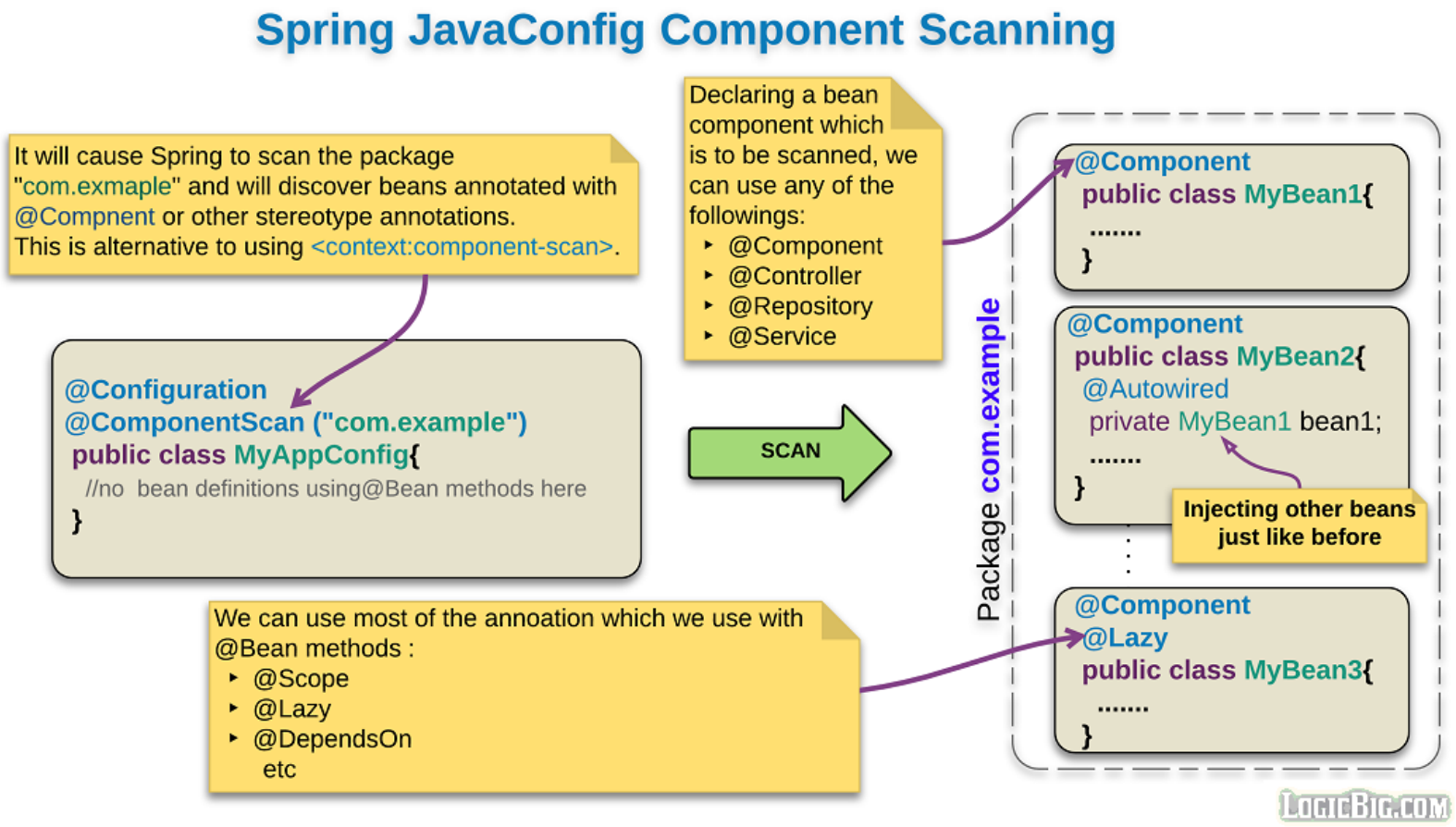



➢ if you want to use third-party classes or jar then use @Bean.

➢ If you are writing your own classes for your application then use @Component.

**Note:** Accessing Beans from the Outside of the Spring Context → using ApplicationContextAware.

- Following diagram summarizes how @ComponentScan works:




**Link:** https://www.logicbig.com/tutorials/spring-framework/spring-core/javaconfig-with-componnet-scan.html

- **@Primary**: Indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency. If exactly one 'primary' bean exists among the candidates, it will be the autowired value.
- **@Order**: The value() is optional and represents an order value as defined in the Ordered interface. Lower values have higher priority. The default value is Ordered.LOWEST_PRECEDENCE, indicating the lowest priority (losing to any other specified order value).
- **@Autowired**: This annotation tells the Spring container where to perform dependency injection.
- **@PostConstruct**: Spring calls the methods annotated with @PostConstruct only once, just after the initialization of bean properties.
- **@PreDestroy**: A method annotated with @PreDestroy runs only once, just before Spring removes our bean from the application context.

- **Demo** (**again**): D:\Java-web\spring-thymeleaf\spring-core-recipe-2dot2

  ========== List all Spring beans ==========

  loginComponent, mathComponent, shopConfiguration,adminComponent

  Aaa, cdrw,

  …

- Read more details: https://docs.spring.io/spring-framework/reference/core/beans/annotation-config.html

- Spring Bean Scopes

  **Link:** https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html

| Scope | Description |
|---|---|
| singleton | (Default) Scopes a single bean definition to a single object instance for each Spring IoC container. |
| prototype | Scopes a single bean definition to any number of object instances. |
| request | Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring `ApplicationContext`. |
| session | Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`. |
| application | Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`. |
| websocket | Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`. |

Ex:
```
@Service @SessionScope @Transactional
public class ShoppingCartServiceImpl implements ShoppingCartService {
    @Autowired private ProductRepository productRepository;
    private Map<Product, Integer> products = new HashMap<>();
    //…
}
```

# Demo 2 ◆ Chap 2

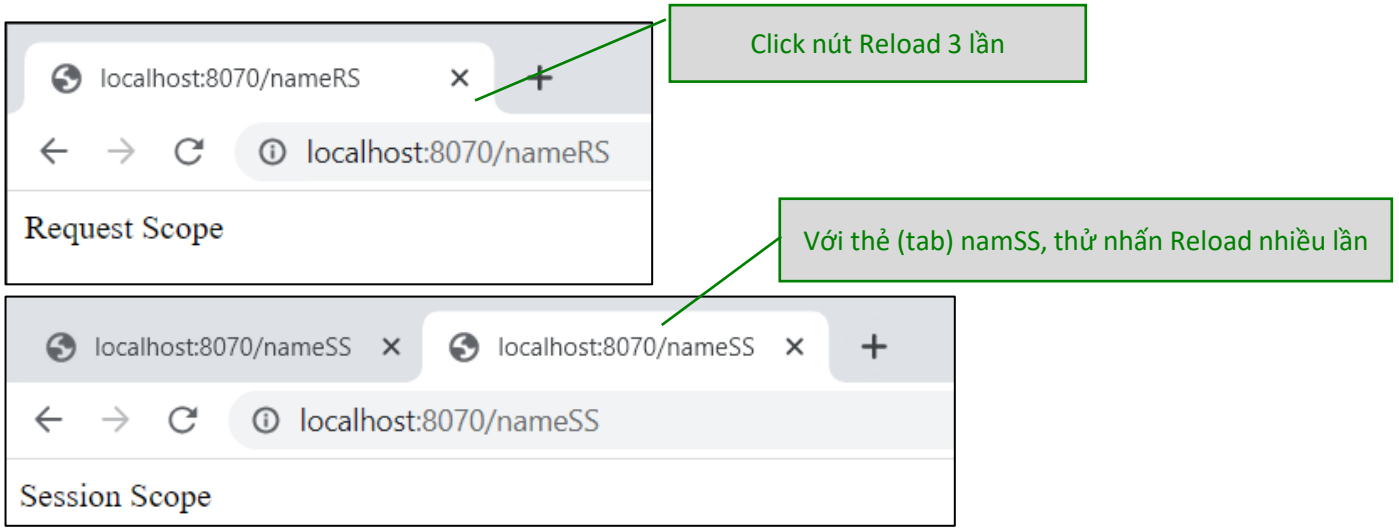
- **Demo 2**: D:\Java-web\spring-thymeleaf**spring-bean-scopes**

```
@Component
@RequestScope
//or @Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class DataRequestScope {
   private String name = "Request Scope";
   public DataRequestScope() {
      System.out.println("DataRequestScope Constructor Called at " +
                                    LocalDateTime.now());
   }
   public String getName() {
      return name;   }
   public void setName(String name) {
      this.name = name;   }
}

@Component
@SessionScope
//or @Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class DataSessionScope {
   private String name = "Session Scope";
   public DataRequestScope() {
      System.out.println("DataSessionScope Constructor Called at " +
                                    LocalDateTime.now());
   }
   public String getName() {
      return name;   }
   public void setName(String name) {
      this.name = name;
   }
}
```

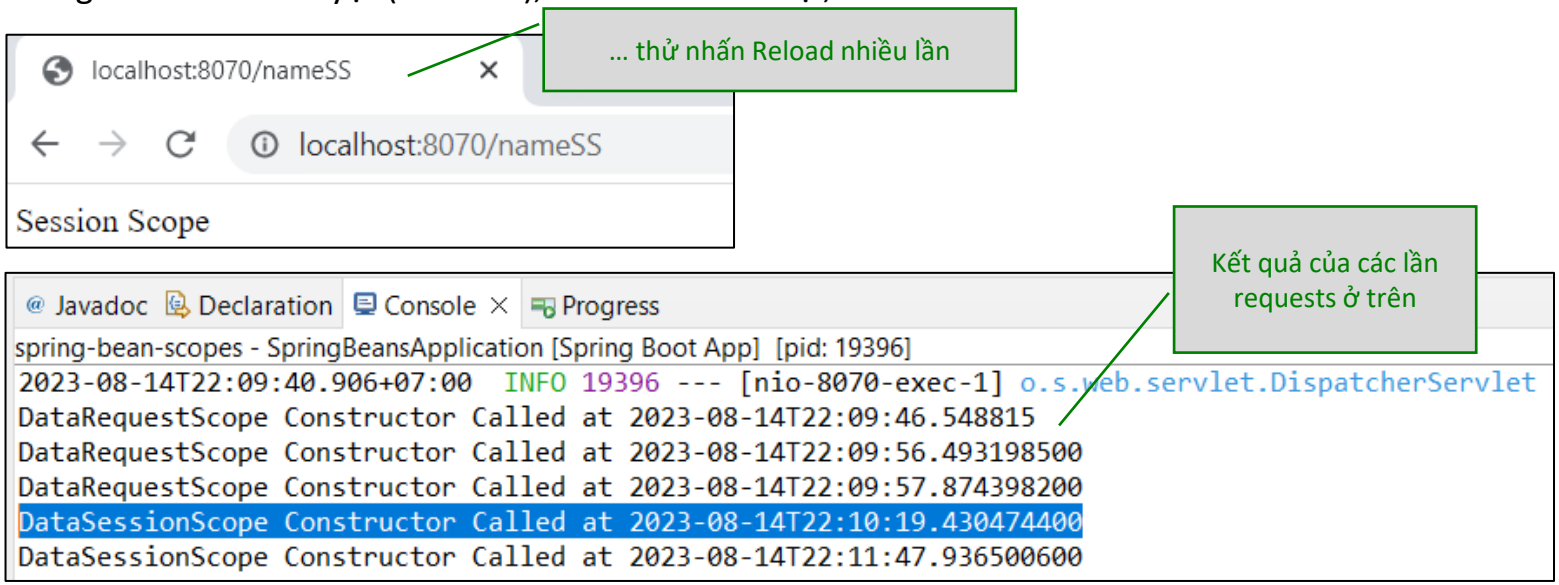
# Demo 2 ◆ Chap 2

```java
@RestController
public class TestBeanScopeController {

   @Autowired
   DataRequestScope dataRequestScope;

   @Autowired
   DataSessionScope dataSessionScope;

   @GetMapping("/nameRS")
   public String helloRS() {
      return dataRequestScope.getName();
   }


   @GetMapping("/nameSS")
   public String helloSS() {
      return dataSessionScope.getName();
   }
}


@SpringBootApplication
public class SpringBeansApplication {
   public static void main(String[] args) {
      SpringApplication.run(SpringBeansApplication.class, args);
   }

}
```

# Demo 2 ◆ Chap 2

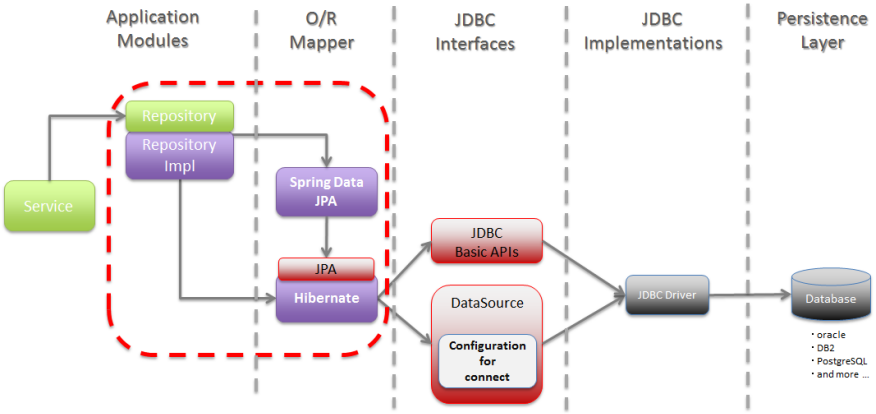



Đóng cửa sổ trình duyệt (Browser), rồi mở nameSS lại, và…



# Building web applications with Java

## Chap 3. Spring Data JPA, Spring Data REST

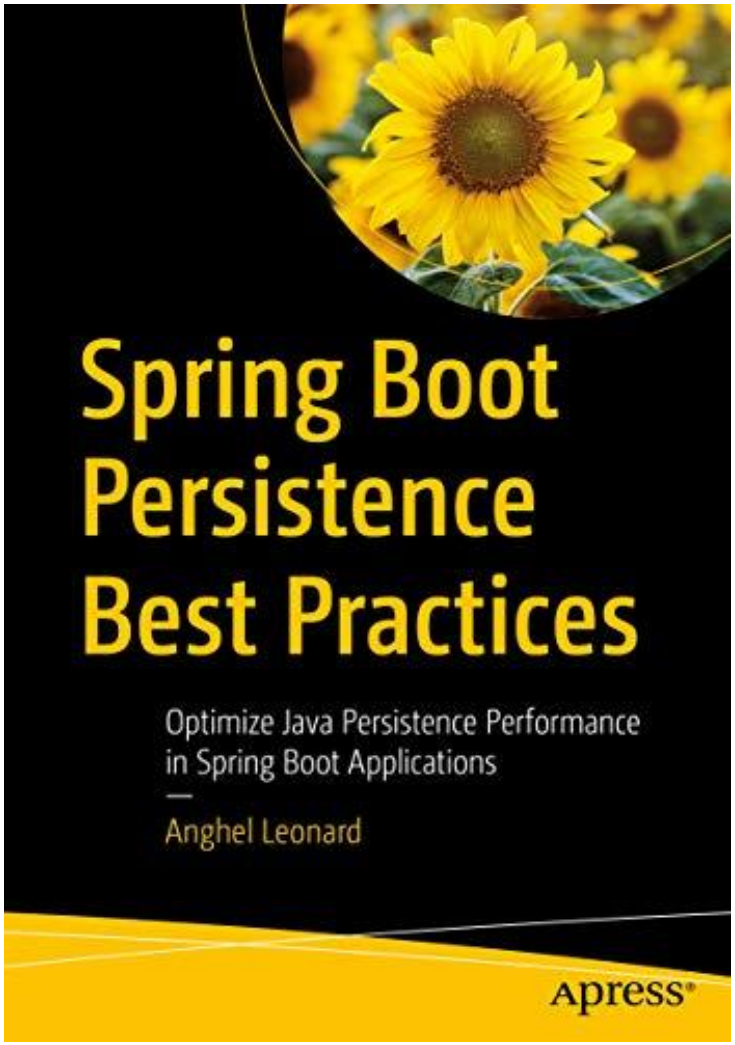



**Biên soạn: Vũ Duy Linh**

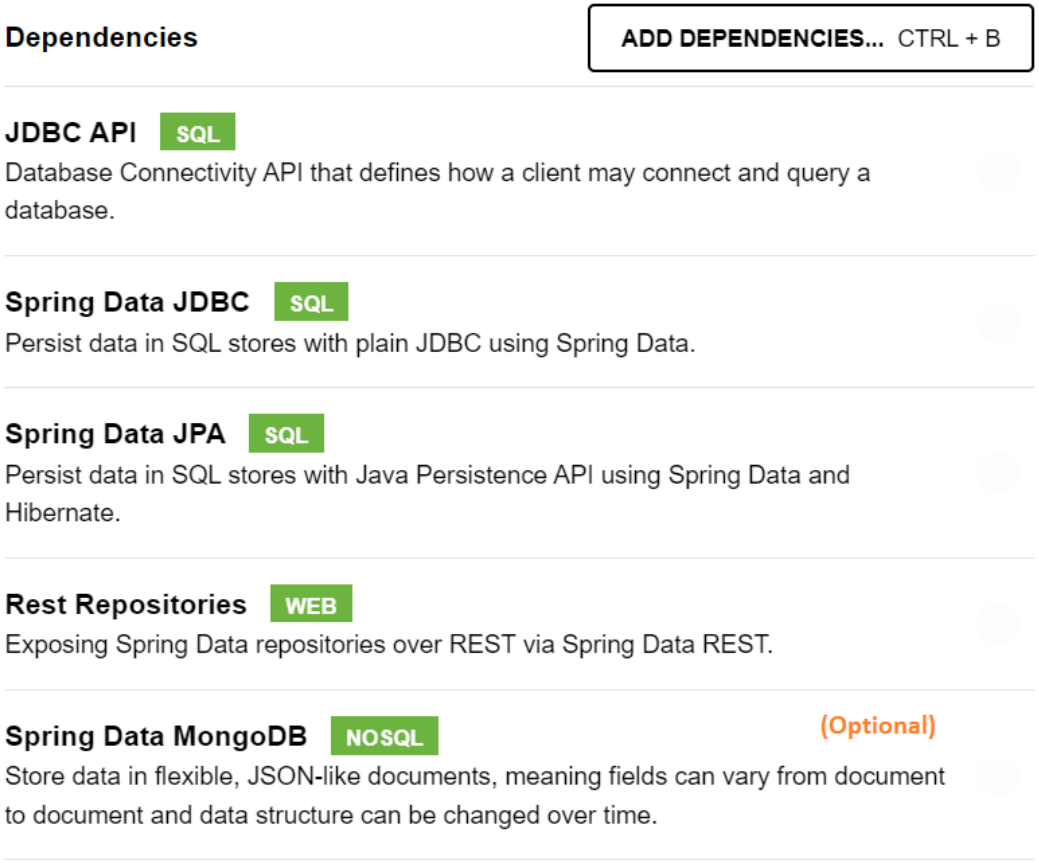
vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

- Advice book:

  My students should read more on this book to apply the high performance recipes in real projects.

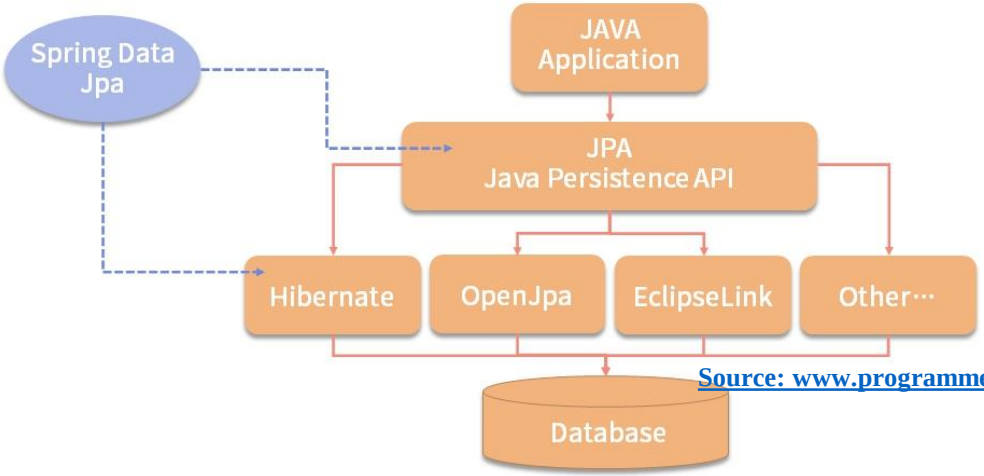


- Spring Data is a project driven by Spring that aims at providing a consistent data access layer for various data stores, right from relational to NoSQL databases.
- There are some popular frameworks for you to apply in your project




- The Spring Data is too broad for the scope of this course. This module mainly focuses on JPA and Spring Data JPA.

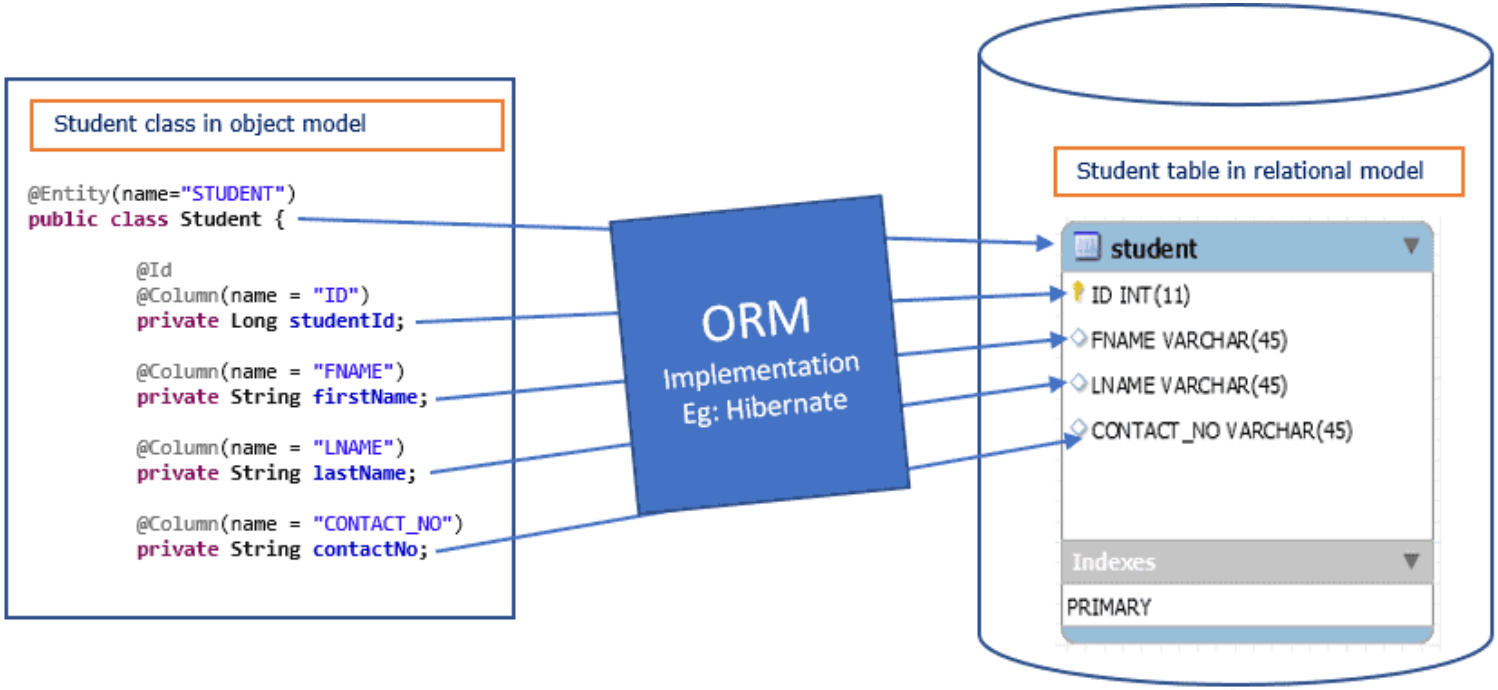
- **Jakarta Persistence/ Java Persistence API (JPA)**
  - Data Persistence is a means for an application to persist and retrieve information from a non-volatile storage system.
  - JPA provides a mechanism for managing persistence and object-relational mapping and functions since the EJB 3.0 specifications.
  - The JPA specification, meaning there is no implementation, defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations.




Source: www.programmerall.com

- **Hibernate ORM (tool/fr**
  - is an object-relational mapping solution for Java environments. Object-relational mapping (ORM) is the programming technique to map application domain model objects to the relational database tables.
  - provides a reference implementation of the JPA that makes it a great choice as an ORM tool with the benefits of loose coupling.

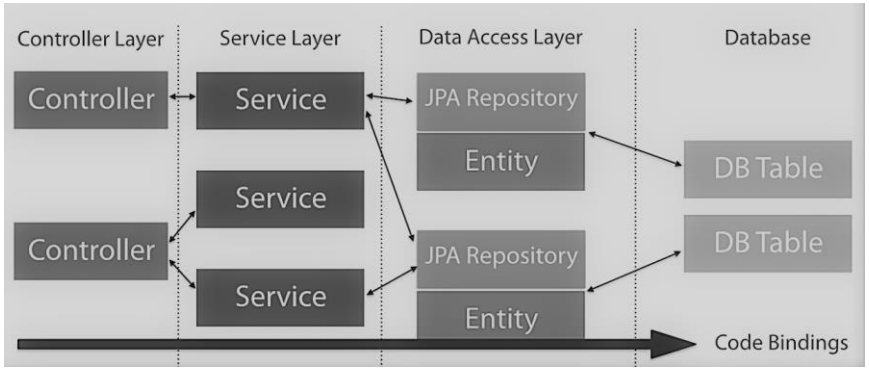
- Popular ORM implementations/frameworks in Java
  - **Hibernate – Open Source**
  - Top Link – By Oracle
  - Eclipse Link – Eclipse Persistence Platform
  - Open JPA – By Apache
  - MyBatis – Open Source – Formerly known as iBATIS



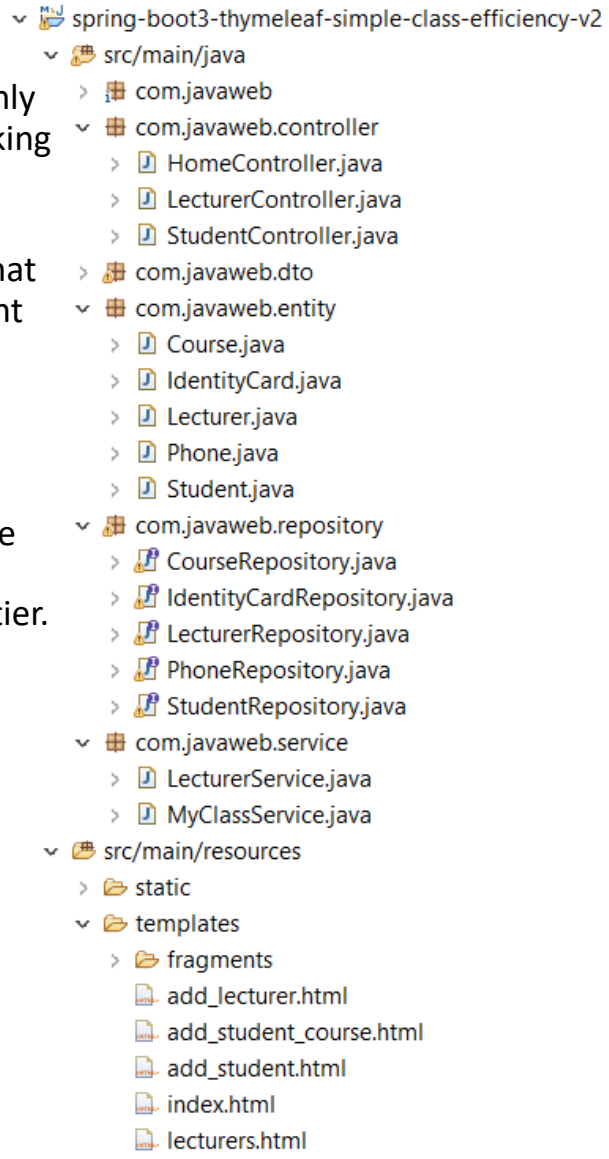

*ORM implements responsibility of mapping the Object to Relational Model.* JavaByDeveloper

- **Spring Data JPA**
  - is part of the larger Spring Data family, it makes easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers.
  - aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed.
  - as a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.
  - Spring Data JPA provides repository support for the Jakarta Persistence API (JPA). It eases development of applications that need to access JPA data sources.
  - Official link: https://spring.io/projects/spring-data-jpa

- Repository Architectural Overview
  - Repositories fit into the data access layer but they aren't the only objects and concepts that you have to keep in mind when working on a server side.
  - At data access layer, we have JPA/Hibernate entities to created/mapped to database tables. By defining an interface that the surface code uses, the data access layer is free to implement the DAO contract anyway.
  - Spring services can then be used to perform logical bundles of work for the application
  - And last, your application will typically have some kind of controller layer that handles request routing coming in from the UI. These controllers can utilize one or more services and are responsible for returning a response to the UI or presentation tier.
  - https://stackabuse.com/guide-to-spring-data-jpa/




A typical Spring application, Source: https://stackabuse.com/guide-to-spring-data-jpa/



- Spring Data JPA provides repository support for the Java Persistence API (JPA). It eases development of applications that need to access JPA data sources.
- Interface **Repository<T, ID>:**
  - Central repository marker interface. Captures the domain type to manage as well as the domain type's id type. General purpose is to hold type information as well as being able to discover interfaces that extend this one during classpath scanning for easy Spring bean creation.
  - Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the same signature as those declared in CrudRepository.
  - Syntax:

org.springframework.data.repository

**Interface Repository<T,ID>**

**Type Parameters:**

`T - the domain type the repository manages`

`ID - the type of the id of the entity the repository manages`

  - https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html

- Interface **CrudRepository<T, ID>:** Interface for generic CRUD operations on a repository for a specific type.
  - Syntax:

```
@NoRepositoryBean
public interface CrudRepository<T,ID>
extends Repository<T,ID>
```

Interface for generic CRUD operations on a repository for a specific type.

  - https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html

  - Method Summary

| Modifier and Type | Method and Description |
|---|---|
| long | **count**() Returns the number of entities available. |
| void | **delete**(**T** entity) Deletes a given entity. |
| void | **deleteAll**() Deletes all entities managed by the repository. |
| void | **deleteAll**(**Iterable**<? extends **T**> entities) Deletes the given entities. |
| void | **deleteAllById**(**Iterable**<? extends **ID**> ids) Deletes all instances of the type T with the given IDs. |
| void | **deleteById**(**ID** id) Deletes the entity with the given id. |

- Method Summary of **CrudRepository<T, ID>:**

| Modifier and Type | Method and Description |
|---|---|
| boolean | **existsById**(**ID** id) Returns whether an entity with the given id exists. |
| **Iterable**<**T**> | **findAll**() Returns all instances of the type. |
| **Iterable**<**T**> | **findAllById**(**Iterable**<**ID**> ids) Returns all instances of the type T with the given IDs. |
| **Optional**<**T**> | **findById**(**ID** id) Retrieves an entity by its id. |
| <S extends **T**> S | **save**(S entity) Saves a given entity. |
| <S extends **T**> **Iterable**<S> | **saveAll**(**Iterable**<S> entities) Saves all given entities. |

- Please read more about **interface Iterable<T>** and **Optional<T>** in Java 8+
  - https://docs.oracle.com/en/java/javase/17/docs/api//java.base/java/util/Optional.html
  - https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html

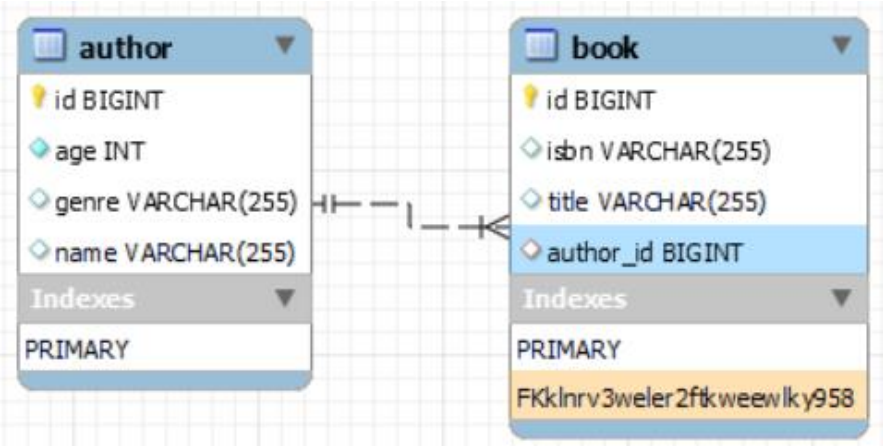
- Interface **JpaRepository<T,ID>:** JPA specific extension of Repository.
  - Syntax:

```
@NoRepositoryBean
public interface JpaRepository<T,ID>
extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>
```

  - Ref: https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html
  - Some useful methods

| Modifier and Type | Method and Description |
|---|---|
| T | getReferenceById(ID id) Returns a reference to the entity with the given identifier. |
| List<T> | findAll() |
| List<T> | findAll(Sort sort) |
| List<T> | findAllById(Iterable<ID> ids) |
| <S extends T> List<S> | saveAll(Iterable<S> entities) |
| <S extends T> List<S> | saveAllAndFlush(Iterable<S> entities) Saves all entities and flushes changes instantly. |
| <S extends T> S | saveAndFlush(S entity) Saves an entity and flushes changes instantly. |
| void | deleteAllInBatch(Iterable<T> entities) Deletes the given entities in a batch which means it will create a single query. |

- **@OneToMany Association**




- The author table has a @OneToMany relationship with the book table
- The mappedBy attribute (in Author) defines a bidirectional relationship. This attribute allows you to refer the associated entities from both sides.
- The author_id column maps this relationship via a foreign key that references the primary key of the author table.
- A book cannot exist without an author, therefore,
  - the author is the parent-side (@OneToMany)
  - while the book is the child-side (@ManyToOne).
- **The @ManyToOne association** is responsible for synchronizing the foreign key column with the Persistence Context

- Demo app: **spring-jpa-one2many-bidirection**

Parent/owner-side

```
@Entity
public class Author implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String genre;
    private int age;

    @OneToMany(cascade = CascadeType.ALL,
            mappedBy = "author", orphanRemoval = true)
    private List<Book> books = new ArrayList<>();

    public void addBook(Book book) {
        this.books.add(book);
        book.setAuthor(this);
    }
```

Child/reverse-side

```
@Entity
public class Book implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String isbn;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id")
    private Author author;

    public Book() {
        super();
    }
```

```
///// Bidirectional association /////
    public Book(Author author) {
        this.author = author;
        author.getBooks().add(this);
    }
```

| id | age | genre | name |
|---|---|---|---|
| 1 | 38 | History | Joana Nimar |
| 2 | 23 | Anthology | Mark Janel |
| 3 | 51 | Anthology | Quartis Young |
| 4 | 38 | Anthology | Alicia Tom |

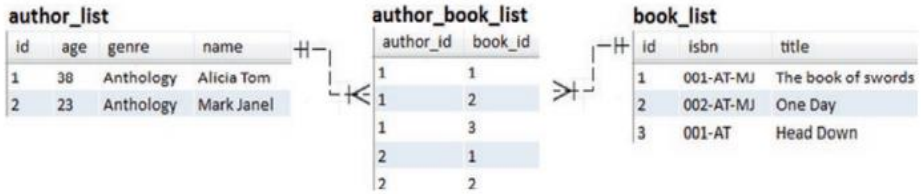
| id | isbn | title | author_id |
|---|---|---|---|
| 1 | 001-JN | A History of Ancient Prague | 1 |
| 2 | A People's History | 002-JN | 1 |
| 3 | The Beatles Anthology | 001-MJ | 2 |
| 4 | 001-AT | The book of swords | 4 |

- **@ManyToMany Association**
  - The bidirectional @ManyToMany association can be navigated from both sides, therefore, both sides can be parents (parent-side).
  - Since both are parents, none of them will hold a foreign key.
  - In this association, there are two foreign keys that are stored in a separate table, known as the junction or join table.
  - The junction table is hidden and it plays the child-side role.


***Figure 1-6.*** *The @ManyToMany table relationship*

- **Demo app**: **spring-jpa-many2many-bidirection**

```java
@Entity
public class Author implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String genre;
    private int age;

    @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable(name = "author_book",
            joinColumns = @JoinColumn(name = "author_id"),
            inverseJoinColumns = @JoinColumn(name = "book_id")
    )
    private Set<Book> books = new HashSet<>();

    public void addBook(Book book) {
        this.books.add(book);
        book.getAuthors().add(this);
    }
```



author_list

| id | age | genre | name |
|---|---|---|---|
| 1 | 38 | Anthology | Alicia Tom |
| 2 | 23 | Anthology | Mark Janel |

author_book_list

| author_id | book_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 1 |
| 2 | 2 |

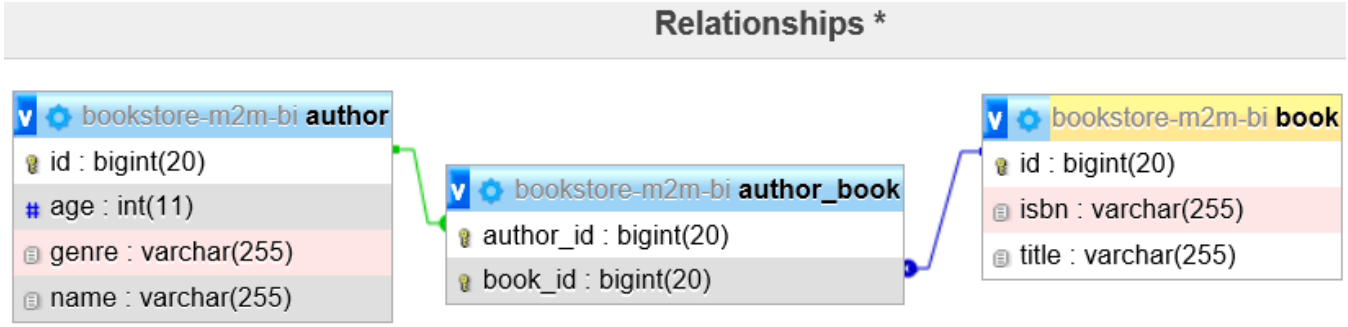
book_list

| id | isbn | title |
|---|---|---|
| 1 | 001-AT-MJ | The book of swords |
| 2 | 002-AT-MJ | One Day |
| 3 | 001-AT | Head Down |

***Figure 1-7.*** *Data snapshot (bidirectional @ManyToMany)*

```java
@Entity
public class Book implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String isbn;

    @ManyToMany(mappedBy = "books")
    private Set<Author> authors = new HashSet<>();

    public Long getId() {
        return id;
    }
```

- **Demo**: **spring-jpa-many2many-bidirection**
- **Note: All code projects are in my Practice Lecture Material.**




atabase: bookstore-m2m-bi » Table: author

SQL | Search | Insert

| id | age | genre | name |
|---|---|---|---|
| 1 | 38 | Anthology | Alicia Tom |
| 2 | 23 | Anthology | Mark Janel |

Table: author_book

| author_id | book_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |

» Database: bookstore-m2m-bi » Table: book

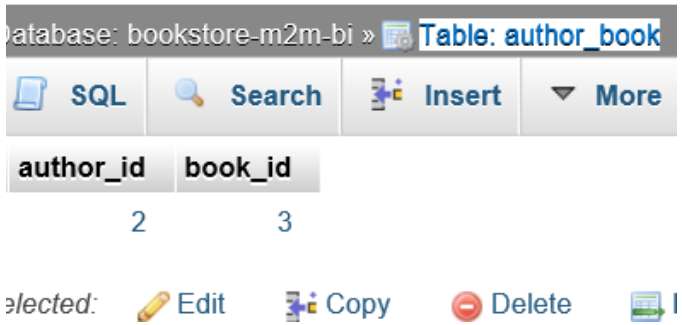
ıre | SQL | Search | Insert | Expo

| id | isbn | title |
|---|---|---|
| 1 | 001-AT-MJ | The book of swords |
| 2 | 001-Java-OOP | Java Object Oriented Programming |
| 3 | 002-AT-MJ | One Day |

Author.java | AuthorRepository.java ×

```
1  package com.bookstore.repository;
2
3⊕ import com.bookstore.entity.Author;
6
7  @Repository
8  public interface AuthorRepository extends JpaRepository<Author, Long> {
9  }
```

- Delete an author with id = 1


```
bookstoreService.removeAuthor2(1L);  //Run again and test deleting...
```

Author.java | Book.java | AuthorRepository.java | BookstoreServi

```
41    public void removeBook(Book book) {
42        this.books.remove(book);
43        book.getAuthors().remove(this);
44    }
45
46    public void removeBooks() {
47        Iterator<Book> iterator = this.books.iterator();
48
49        while (iterator.hasNext()) {
50            Book book = iterator.next();
51
52            book.getAuthors().remove(this);
53            iterator.remove();
54        }
55    }
```

BookstoreService.java | AuthorRepository.java

```
    /* to remove an entity that owns the association, you then need to
     * remove all associations yourself before you can remove the entity.
     */
    @Transactional
    public void removeAuthor1(Long authorId) {
        Author author = authorRepository.getReferenceById(authorId);
        for (Book book : author.getBooks()) {
            author.removeBook(book);
        }
        authorRepository.delete(author);

    }

    @Transactional
    public void removeAuthor2(Long authorId) {
        Author author = authorRepository.getReferenceById(authorId);
        author.removeBooks();
        authorRepository.delete(author);
    }
```

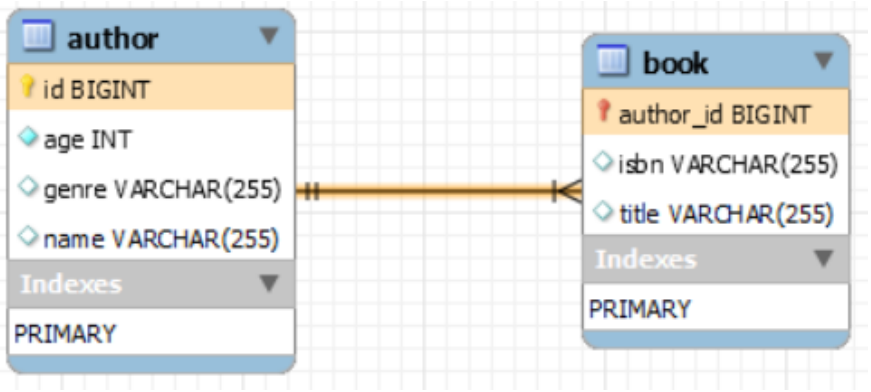
- **Demo: spring-jpa-one2one-mapsid**
  - **@MapsId** and @OneToOne **shared key**

*Author.java × Book.java

```java
package com.bookstore.entity;

import jakarta.persistence.Entity;

@Entity
@Table
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String genre;
    private int age;

    //...
}
```

*Book.java ×

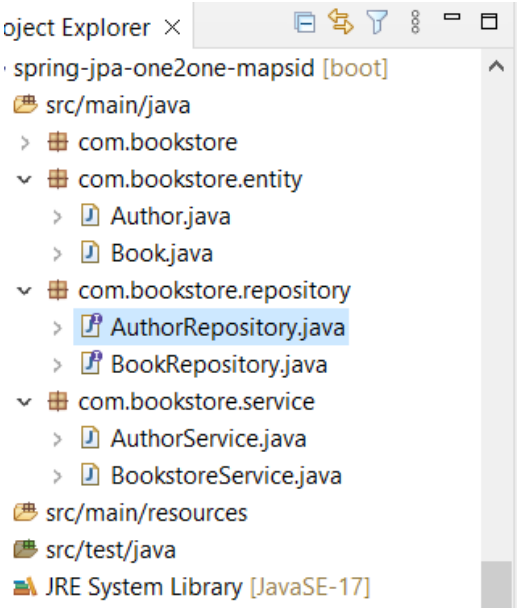
```java
package com.bookstore.entity;

import jakarta.persistence.Entity;
@Entity
@Table
public class Book {
    @Id
    private Long id;
    private String title;
    private String isbn;

    @MapsId
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id")
    private Author author;
    //...
}
```



- **Demo app:**

  **spring-jpa-one2one-mapsid**

```java
@SpringBootApplication
public class SpringJpaOne2oneMapsidApplication {/*implements CommandLineRunner
    @Autowired
    private AuthorService authorService;

    @Autowired
    private BookstoreService bookstoreService;
    public static void main(String[] args) {
        SpringApplication.run(SpringJpaOne2oneMapsidApplication.class, args);
    }

    @Bean
    public ApplicationRunner init() {
        return args -> {
            authorService.newBook();
            bookstoreService.newBookOfAuthor();
            Book book = bookstoreService.fetchBookByAuthorId();
            System.out.println(book);
        };
    }
```




AuthorRepository.java | AuthorService.java

```java
package com.bookstore.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service  //<=> Component --> to create Spring Bean
public class AuthorService {

    @Autowired
    private AuthorRepository authorRepository;

    @Transactional
    public void newBook() {
        authorRepository.save(new Author("Joana Nimar", "History", 34));
    }
}
```

## Query Methods

Spring Data JPA supports a way to create a query from the method name. Query methods are methods which are declared in the repository interface to perform CRUD operations on a database. This section describes the various ways to create a query with Spring Data JPA.

- Query Creation
- Using JPA Named Queries
- Using @Query (JPQL query by using the @Query annotation)
  - Applying a QueryRewriter
  - Using Advanced LIKE Expressions
  - Native Queries
  - Using with special parameter handling
- Using Named Parameters
- Using SpEL Expressions

Official reference: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

## Query Creation (in Query Methods)

- The query builder mechanism built into the Spring Data repository infrastructure is useful for building constraining queries over entities of the repository.
- Structure of Derived (dẫn xuất) Query methods: Typically, a Derived Query method has two elements: **subject keywords (the action),** and **predicate keywords and modifiers (the conditions)**
  - **Subject keywords** : is the introducing clause (find…By, exists…By, …), it may contain further expressions (between find/exists/count and By) for result-limiting keywords such as Distinct or Top/First.
  - **Predicate keywords and modifiers (vị từ)**: is placed after the subject. It can be entity properties (concatenating with And/Or) followed by one or more keywords (StartingWith, EndingWith, Containing, IgnoreCase…).
- Ex: Query creation from method names

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

- The supported **keywords** inside **method names** as below table

| Keyword | Sample | JPQL snippet |
|---|---|---|
| Distinct | findDistinctByLastnameAndFirstname | select distinct … where x.lastname = ?1 and x.firstname = ?2 |
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |

# Query Methods - JpaRepository ◆ Spring Data JPA 3.x ◆ Chap 3

| Keyword | Sample | JPQL snippet |
|---|---|---|
| Is, Equals | findByFirstname,findByFirstnameIs, findByFirstnameEquals | … where x.firstname = ?1 |
| Between | findByStartDateBetween | … where x.startDate between ?1 and ?2 |
| LessThan | findByAgeLessThan | … where x.age < ?1 |
| LessThanEqual | findByAgeLessThanEqual | … where x.age <= ?1 |
| GreaterThan | findByAgeGreaterThan | … where x.age > ?1 |
| GreaterThanEqual | findByAgeGreaterThanEqual | … where x.age >= ?1 |
| After | findByStartDateAfter | … where x.startDate > ?1 |
| Before | findByStartDateBefore | … where x.startDate < ?1 |
| IsNull, Null | findByAge(Is)Null | … where x.age is null |
| IsNotNull, NotNull | findByAge(Is)NotNull | … where x.age not null |
| Like | findByFirstnameLike | … where x.firstname like ?1 |
| NotLike | findByFirstnameNotLike | … where x.firstname not like ?1 |

| Keyword | Sample | JPQL snippet |
|---|---|---|
| StartingWith | findByFirstnameStartingWith | … where x.firstname like ?1 (parameter bound with appended %) |
| EndingWith | findByFirstnameEndingWith | … where x.firstname like ?1 (parameter bound with prepended %) |
| Containing | findByFirstnameContaining | … where x.firstname like ?1 (parameter bound wrapped in %) |
| OrderBy | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by x.lastname desc |
| Not | findByLastnameNot | … where x.lastname <> ?1 |
| In | findByAgeIn(Collection<Age> ages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collection<Age> ages) | … where x.age not in ?1 |
| True | findByActiveTrue() | … where x.active = true |
| False | findByActiveFalse() | … where x.active = false |
| IgnoreCase | findByFirstnameIgnoreCase | … where UPPER(x.firstname) = UPPER(?1) |

- **Demo:** D:\Java-web\...**spring-jpa-limit-result-size-via-query-creator**

# Query Methods - JpaRepository ◆ Spring Data JPA 3.x ◆ Chap 3

AuthorRepository.java

```
10 @Repository
11 @Transactional(readOnly = true)
12 public interface AuthorRepository extends JpaRepository<Author, Long> {
13
14     List<Author> findFirst5By();
15
16     List<Author> findFirst5ByAge(int age);
17
18     List<Author> findFirst5ByAgeGreaterThanEqual(int age);
19
20     List<Author> findFirst5ByAgeLessThan(int age);
21
22     List<Author> findFirst5ByAgeOrderByNameDesc(int age);
23
24     List<Author> findFirst5ByGenreOrderByAgeAsc(String genre);
25
26     List<Author> findFirst5ByAgeGreaterThanEqualOrderByNameAsc(int age);
27
28     List<Author> findFirst5ByGenreAndAgeLessThanOrderByNameDesc(String genre, int age);
29
30     List<AuthorDto> findFirst5ByOrderByAgeAsc();
31
32 }
```

| id | age | genre | name |
|---|---|---|---|
| 1 | 56 | Anthology | Mark Janel |
| 2 | 43 | Horror | Olivia Goy |
| 3 | 51 | Anthology | Quartis Young |
| 4 | 34 | History | Joana Nimar |
| 5 | 38 | Anthology | Alicia Tom |
| 6 | 56 | Anthology | Anatoly Quentin |
| 7 | 56 | Anthology | Katy Loin |
| 8 | 30 | Horror | Ulm Nair |
| 9 | 37 | Horror | Ilan Tastenir |
| 10 | 57 | History | Biar Gul |
| 11 | 56 | History | Rona Fullos |
| 12 | 56 | Anthology | Joana Vaser |
| 13 | 44 | History | Vitali Hill |
| 14 | 56 | Horror | Carmen Corra |
| 15 | 56 | Anthology | Dalia Rene |

**bookstoreService.fetchFirst5ByAge(56)**

```
Fetch the first 5 authors by age equal to 56:
[Author{id=1, age=56, name=Mark Janel, genre=Anthology},
Author{id=6, age=56, name=Anatoly Quentin, genre=Anthology},
Author{id=7, age=56, name=Katy Loin, genre=Anthology},
Author{id=11, age=56, name=Rona Fullos, genre=History},
Author{id=12, age=56, name=Joana Vaser, genre=Anthology}]
```

**bookstoreService.fetchFirst5ByGenreAndAgeLessThanOrderByNameDesc("Horror", 50)**

```
Fetch the first 5 authors by genre Horror and age less than 50 ordered descending by name:
[Author{id=8, age=30, name=Ulm Nair, genre=Horror},
Author{id=2, age=43, name=Olivia Goy, genre=Horror},
Author{id=9, age=37, name=Ilan Tastenir, genre=Horror}]
```

- **Using JPA Named Queries and Spring Projection interface**
  - **Named query** is a statically defined query with a predefined unchangeable query string. They're validated when the session factory is created, thus making the application to fail fast in case of an error.

**Demo: spring-jpa-dto-projection-annotated-named-query**

Author.java

```
10 @NamedQueries({ //demonstrate for @NamedQueries
11     @NamedQuery(
12         name = "Author.fetchName",
13         query = "SELECT a.name FROM Author a"
14     ),
15
16     @NamedQuery(
17         name = "Author.fetchNameAndAge",
18         query = "SELECT a.age AS age, a.name AS name FROM Author a"
19     )
20 })
21 @Entity
22 public class Author {
23     @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Long id;
25     private int age;
26     private String name;
27     private String genre;
```

  - Using named queries , there are some advantages listed below:
    - one consist in the fact that named query annotations are compiled and validated at startup time
    - the annotations will be processed only once, then executed at runtime in query form
    - the application avoids continuously parsing JPQL and generating SQL
    - named queries are easier to maintain than string literals embedded in your code

- **Data Transfer Object (DTO):**

  - Martin Folwer defines a DTO as “an object that carries data between processes in order to reduce the number of method calls”.

  - Most of the time, DTOs contain only a subset of entity attributes and this way you avoid fetching more data (columns) than needed.

- **Spring Projection interface and DTO**

  - Spring projections have essentially the same purpose.

  - DTO relies on classes with constructor and getters/setters, while Spring projections rely on interfaces and automatically generated proxies.

*UserRegistrationDto.java

```
11 public class UserRegistrationDto {
17     private String password;
18
24     private String username;
25
26     @Size(min=1, message = "Please provide your first name")
27     private String firstName;
28
29     @Size(min=1, message = "Please provide your last name")
30     private String lastName;
31
37     private String email;
38
40     private Set<String> roles;
41
42     public UserRegistrationDto() {
44
45     public String getPassword() {
46         return password;
47     }
48
49     public void setPassword(String password) {
50         this.password = password;
51     }
52
```

Author.java

```
21 @Entity
22 public class Author {
23     @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Long id;
25     private int age;
26     private String name;
27     private String genre;
```

Spring Projection Interface

AuthorNameAge.java

```
1 package com.bookstore.dto;
2
3 public interface AuthorNameAge {
4
5     public String getName();
6     public int getAge();
7 }
```

- JPA Repository uses @Named Query defined above

Author.java | AuthorRepository.java

```java
package com.bookstore.repository;

import java.util.List;

@Repository
@Transactional(readOnly = true)
public interface AuthorRepository extends JpaRepository<Author, Long> {

    // Scalar Mapping
    List<String> fetchName();

    // Spring projection
    List<AuthorNameAge> fetchNameAndAge();
}
```

```java
@Service
public class BookstoreService {

    private final AuthorRepository authorRepository;

    public BookstoreService(AuthorRepository authorRepository) {
        this.authorRepository = authorRepository;
    }

    public List<AuthorNameAge> fetchAuthorsNamesAndAges() {

        return authorRepository.fetchNameAndAge();
    }

    public List<String> fetchAuthorsNames() {

        return authorRepository.fetchName();
    }
}
```

MainApplication.java

```
13 @SpringBootApplication
14 public class MainApplication {
15     @Autowired
16     private BookstoreService bookstoreService;
17
18     public static void main(String[] args) {
19         SpringApplication.run(MainApplication.class, args);
20     }
21
22     @Bean
23     public ApplicationRunner init() {
24         return args -> {
25             List<AuthorNameAge> authors = bookstoreService.fetchAuthorsNamesAndAges();
26             System.out.println("Spring projection on Name and Age of Author:");
27
28             System.out.format("%-20s %s%n","Name","Age");
29
30             for (AuthorNameAge author : authors) {
31                 System.out.printf("%-20s %d\n", author.getName(), author.getAge());
32             }
```

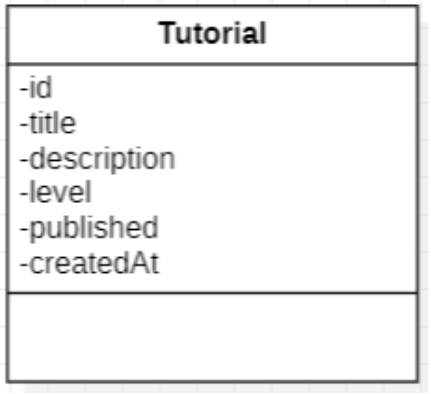
| | id | age | genre | name |
|---|---|---|---|---|
| ▶ | 1 | 23 | Anthology | Mark Janel |
| | 2 | 43 | Horror | Olivia Goy |
| | 3 | 51 | Anthology | Quartis Young |
| | 4 | 34 | History | Joana Nimar |
| | 5 | 38 | Anthology | Alicia Tom |
| | 6 | 56 | Anthology | Katy Loin |
| * | NULL | NULL | NULL | NULL |

**Spring projection on Name and Age of Author:**

| Name | Age |
|---|---|
| Mark Janel | 23 |
| Olivia Goy | 43 |
| Quartis Young | 51 |
| Joana Nimar | 34 |
| Alicia Tom | 38 |
| Katy Loin | 56 |
| Number of authors:6 | |

## Java/Jakarta Persistence query language (JPQL)

- JPQL, Jakarta Persistence Query Language defined in JPA specification, is a powerful query language that allows you to define database queries based on your entity model. Its structure and syntax are very similar to SQL.
- JPQL uses **the entity object model** instead of database tables to define a query. That makes it very comfortable for us Java developers.
- JPA/Hibernate has to transform the JPQL query into SQL.

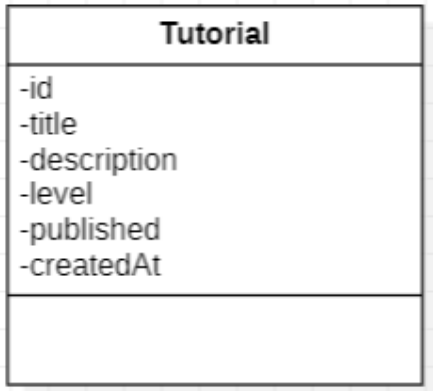



- Query Structure

```
SELECT ... FROM ... [WHERE ...] [GROUP BY ... [HAVING ...]] [ORDER BY ...]

DELETE FROM ... [WHERE ...]

UPDATE ... SET ... [WHERE ...]
```

  - Distinct query
  - Grouping – The GROUP BY and HAVING clause
  - Ordering – The ORDER BY clause

- **Demo: spring-jpa-jpql-example**
- Tutorial Entity:




- **JPA/Hibernate** creates a tutorial table:

| | id | level | published | created_at | description | title |
|---|---|---|---|---|---|---|
| ▶ | 1 | 3 | 1 | 2022-03-11 00:00:00.000000 | Tut#1 Description | Spring Data |
| | 2 | 1 | 0 | 2022-03-11 00:00:00.000000 | Tut#2 Description | Java Spring |
| | 3 | 3 | 1 | 2022-04-26 00:00:00.000000 | Tut#3 Description | Hibernate |
| | 4 | 2 | 0 | 2022-04-26 00:00:00.000000 | Tut#4 Description | Spring Boot |
| | 5 | 3 | 1 | 2022-05-19 00:00:00.000000 | Tut#5 Description | Spring Data JPA |
| | 6 | 4 | 0 | 2022-05-19 00:00:00.000000 | Tut#6 Description | Spring Batch |
| | 7 | 5 | 0 | 2022-05-19 00:00:00.000000 | Tut#7 Description | Spring Security |
| * | NULL | NULL | NULL | NULL | NULL | NULL |

TutorialRepository.java

```java
package com.javaweb.repository;
import java.util.Date;

public interface TutorialRepository extends JpaRepository<Tutorial, Long> {

    @Query("SELECT t FROM Tutorial t WHERE t.title LIKE %?1%")
    List<Tutorial> findByTitleLike(String title);                                   //1

    @Query("SELECT t FROM Tutorial t WHERE t.createdAt BETWEEN ?1 AND ?2")          //2
    List<Tutorial> findByDateBetween(Date start, Date end);

    @Query("SELECT t FROM Tutorial t WHERE t.published=:isPublished AND t.level BETWEEN :start AND :end")
    List<Tutorial> findByLevelBetween(@Param("start") int start, @Param("end") int end,
                                      @Param("isPublished") boolean isPublished);   //3

    @Query("SELECT t FROM Tutorial t WHERE t.published=?1")
    List<Tutorial> findByPublishedAndSort(boolean isPublished, Sort sort);          //4

    @Query("SELECT t FROM Tutorial t WHERE t.published=true ORDER BY t.createdAt DESC")
    List<Tutorial> findAllPublishedOrderByCreatedDesc();                            //5

    @Query("SELECT t FROM Tutorial t")
    Page<Tutorial> findAllWithPagination(Pageable pageable);                        //6

    /*
     * JPA Query Update: Spring Data JPA Query to update an entity using @Query
     * along with @Transactional and @Modifying.
     */
    @Transactional
    @Modifying
    @Query("UPDATE Tutorial t SET t.published=true WHERE t.id=?1")
    int updatePublishTutorial(Long id);                                             //7
}
```

```
@Service
public class TutorialService {
   @Autowired TutorialRepository tutorialRepository;

   //…Call some methods (from 1->7) of tutorialRepository as bellow

}
```

- `tutorials = tutorialRepository.findByTitleLike("Data");` //1

```
Tutorial [id=1, title=Spring Data, description=Tut#1 Description, level=3,
published=true, createdAt=2022-03-11 00:00:00.0]
Tutorial [id=5, title=Spring Data JPA, description=Tut#5 Description, level=3,
published=true, createdAt=2022-05-19 00:00:00.0]
```

```
Date myDate1 = new SimpleDateFormat("yyyy-MM-dd").parse("2022-04-11");          //2
Date myDate2 = new SimpleDateFormat("yyyy-MM-dd").parse("2022-05-11");
```

- `tutorials = tutorialRepository.findByDateBetween(myDate1, myDate2);`

```
Tutorial [id=3, title=Hibernate, description=Tut#3 Description, level=3, published=true,
createdAt=2022-04-26 00:00:00.0]
Tutorial [id=4, title=Spring Boot, description=Tut#4 Description, level=2,
published=false, createdAt=2022-04-26 00:00:00.0]
```

- `tutorials = tutorialRepository.findByLevelBetween(3, 5, true);` //3

```
Tutorial [id=1, title=Spring Data, description=Tut#1 Description, level=3,
published=true, createdAt=2022-03-11 00:00:00.0]
Tutorial [id=3, title=Hibernate, description=Tut#3 Description, level=3, published=true,
createdAt=2022-04-26 00:00:00.0]
Tutorial [id=5, title=Spring Data JPA, description=Tut#5 Description, level=3,
published=true, createdAt=2022-05-19 00:00:00.0]
```

```
tutorials = tutorialRepository.findAllPublishedOrderByCreatedDesc();     //4
Tutorial [id=5, title=Spring Data JPA, description=Tut#5 Description, level=3,
published=true, createdAt=2022-05-19 00:00:00.0]
Tutorial [id=3, title=Hibernate, description=Tut#3 Description, level=3, published=true,
createdAt=2022-04-26 00:00:00.0]
Tutorial [id=1, title=Spring Data, description=Tut#1 Description, level=3,
published=true, createdAt=2022-03-11 00:00:00.0]
```

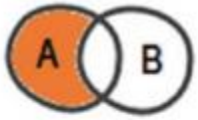
```
tutorials = tutorialRepository.findByPublishedAndSort(true,
                                        Sort.by("level").descending());     //5
Tutorial [id=1, title=Spring Data, description=Tut#1 Description, level=3,
published=true, createdAt=2022-03-11 00:00:00.0]
Tutorial [id=3, title=Hibernate, description=Tut#3 Description, level=3, published=true,
createdAt=2022-04-26 00:00:00.0]
Tutorial [id=5, title=Spring Data JPA, description=Tut#5 Description, level=3,
published=true, createdAt=2022-05-19 00:00:00.0]
```

```
int page = 0; int size = 3;                                                 //6
Pageable pageable = PageRequest.of(page, size, Sort.by("level").descending());
Page<Tutorial> turorialPages =
                    tutorialRepository.findAllWithPagination(pageable);
tutorials = turorialPages.getContent();
int totalPages = turorialPages.getTotalPages(); //totalPages= 3
Tutorial [id=7, title=Spring Security, description=Tut#7 Description, level=5,
published=false, createdAt=2022-05-19 00:00:00.0]
Tutorial [id=6, title=Spring Batch, description=Tut#6 Description, level=4,
published=false, createdAt=2022-05-19 00:00:00.0]
Tutorial [id=1, title=Spring Data, description=Tut#1 Description, level=3,
published=true, createdAt=2022-03-11 00:00:00.0]
```

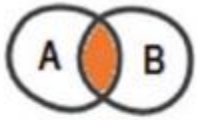
```
tutorialRepository.updatePublishTutorial(tutorials.get(7).getId());     //7
```

- **Query Structure (JPQL) for Joining multiple entities**
  - INNER JOIN
  - OUTER JOIN: LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN

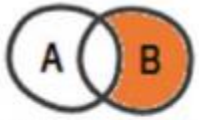

```
SELECT select_list
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



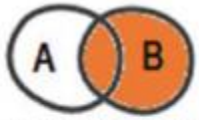

```
SELECT select_list
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



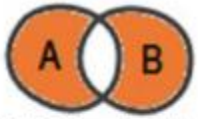

```
SELECT select_list
FROM TableA A
INNER JOIN TableB
ON A.Key = B.Key
```




```
SELECT select_list
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT select_list
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

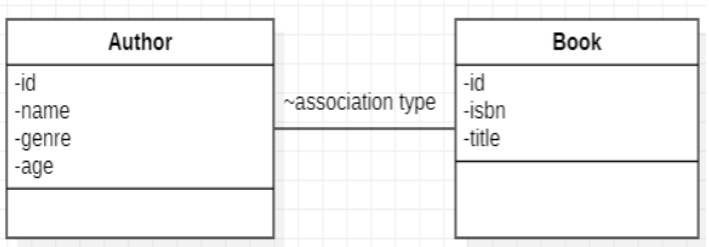


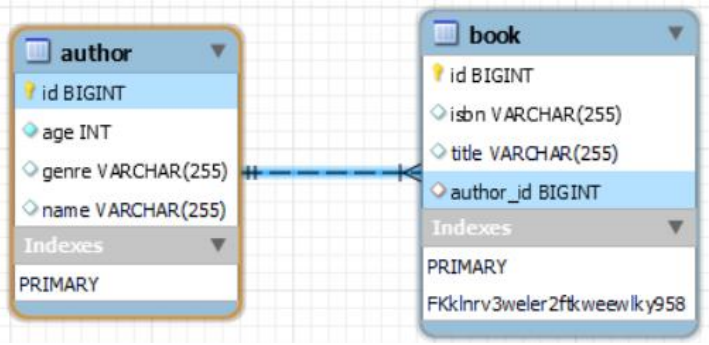

```
SELECT select_list
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```


```
SELECT select_list
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

- Spring Projection Interface on Two Entities via Inner/Outer Joins:
  - Consider the **Author** and **Book** entities involved in a **bidirectional lazy @OneToMany** association => has Materialized Association
  - **Demo**: **spring-thymeleaf\spring-jpa-dto-inner-outer-joins**

```
@Entity
public class Author {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String genre;
    private int age;

    @OneToMany(cascade = CascadeType.ALL,
            mappedBy = "author", orphanRemoval = true)
    private List<Book> books = new ArrayList<>();
@Entity
public class Book {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String isbn;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id")
    private Author author;
```




```
Ex: @Query("SELECT a AS author, b.title AS title FROM Author a JOIN a.books b") //INNER JOIN
    List<BookstoreDto> fetchAll();

    @Query(value = "SELECT b.title AS title, a.name AS name FROM Author a INNER JOIN a.books b")
    List<AuthorNameBookTitle> findAuthorsAndBooksJpql();
```

- **Find all authors have at least one book:**

```
@Query("SELECT a FROM Author a JOIN FETCH a.books")
List<Author> findByJoinFetchJpql();
```

- **Fetch authors and books filtering by author's genre and book's price (JPQL):**

```
@Query(value = "SELECT b.title AS title, a.name AS name "
       + "FROM Author a INNER JOIN a.books b WHERE a.genre = ?1 AND b.price < ?2")
List<AuthorNameBookTitle> findAuthorsAndBooksByGenreAndPriceJpql(String genre,
                                                                 int price);
```

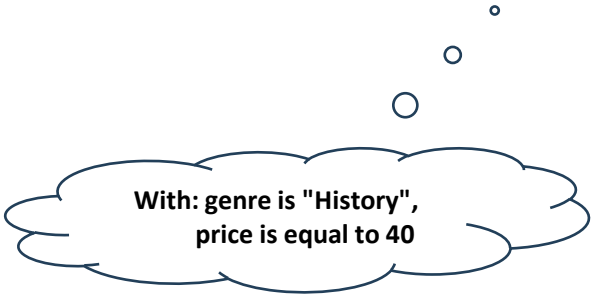
- **Fetch books and authors filtering by author's genre and book's price (JPQL):**

```
@Query(value = "SELECT b.title AS title, a.name AS name "
        + "FROM Book b INNER JOIN b.author a WHERE a.genre = ?1 AND b.price < ?2")
List<AuthorNameBookTitle> findBooksAndAuthorsByGenreAndPriceJpql(String genre,
                                                                 int price);
```

```
findByJoinFetchJpql():
Joana Nimar | History
Mark Janel | Anthology
Olivia Goy | Horror
Larisa Tomay | History
```

```
findAuthorsAndBooksByGenreAndPriceJpql:
Joana Nimar | A History of Ancient Prague
Larisa Tomay | Sapiens
```

```
findBooksAndAuthorsByGenreAndPriceJpql:
Joana Nimar | A History of Ancient Prague
Larisa Tomay | Sapiens
```



- **Fetch authors… (where Author.id NOT null, Book.id is not care)**

```
@Query(value =
        "SELECT b.title AS title, a.name AS name FROM Author a LEFT JOIN a.books b")
List<AuthorNameBookTitle> findAuthorsAndBooksJpql_Left_Joins();
```

- **Fetch books…**

```
@Query(value =
        "SELECT b.title AS title, a.name AS name FROM Book b LEFT JOIN b.author a")
List<AuthorNameBookTitle> findBooksAndAuthorsJpql_Left_Joins();
```

```
findAuthorsAndBooksJpql_Left_Joins:
Mark Janel | The Beatles Anthology
Olivia Goy | Carrie
Quartis Young | null
Joana Nimar | A History of Ancient Prague
Joana Nimar | A People's History
Larisa Tomay | Ghost Soldiers
Larisa Tomay | Sapiens
```

```
findBooksAndAuthorsJpql_Left_Joins:
Joana Nimar | A History of Ancient Prague
Joana Nimar | A People's History
Mark Janel | The Beatles Anthology
Olivia Goy | Carrie
Larisa Tomay | Ghost Soldiers
Larisa Tomay | Sapiens
```

- **Fetch books…**

```
@Query(value =
       "SELECT b.title AS title, a.name AS name FROM Author a RIGHT JOIN a.books b")
List<AuthorNameBookTitle> findAuthorsAndBooksJpql_Right_Joins();
```

- Fetch **authors**…

```
@Query(value =
      "SELECT b.title AS title, a.name AS name FROM Book b RIGHT JOIN b.author a")
List<AuthorNameBookTitle> findBooksAndAuthorsJpql_Right_Joins();
```

➔ Right_Joins's results are similar Left_Joins type.

- Spring Projection Interface on Two Entities via Full Join using native SQL:
  - **Demo**: **spring-jpa-dto-via-full-joins-in-mysql**

AuthorRepository.java

```java
package com.bookstore.repository;

import java.util.List;

//@Repository
@Transactional(readOnly = true)
public interface AuthorRepository extends JpaRepository<Author, Long> {
/* NOTES:
 *  1. MySQL doesn't provide support for FULL JOINs, but there are a few ways you can
 *     simulate FULL JOINs. The best approach relies on UNION or UNION ALL.
 *
 *  2. JPA doesn't support the UNION clause; therefore, you need to use native SQL.
 * */
    // Fetch all authors and books (SQL)
    @Query(value = "(SELECT b.title AS title, a.name AS name FROM author a "
            + "LEFT JOIN book b ON a.id = b.author_id) "
            + "UNION " //  will remove duplicates (use UNION ALL to keep duplicates)
            + "(SELECT b.title AS title, a.name AS name FROM author a "
            + "RIGHT JOIN book b ON a.id = b.author_id "
            + "WHERE a.id IS NULL)",
            nativeQuery = true)
    List<AuthorNameBookTitle> findAuthorsAndBooksSql();
}
```

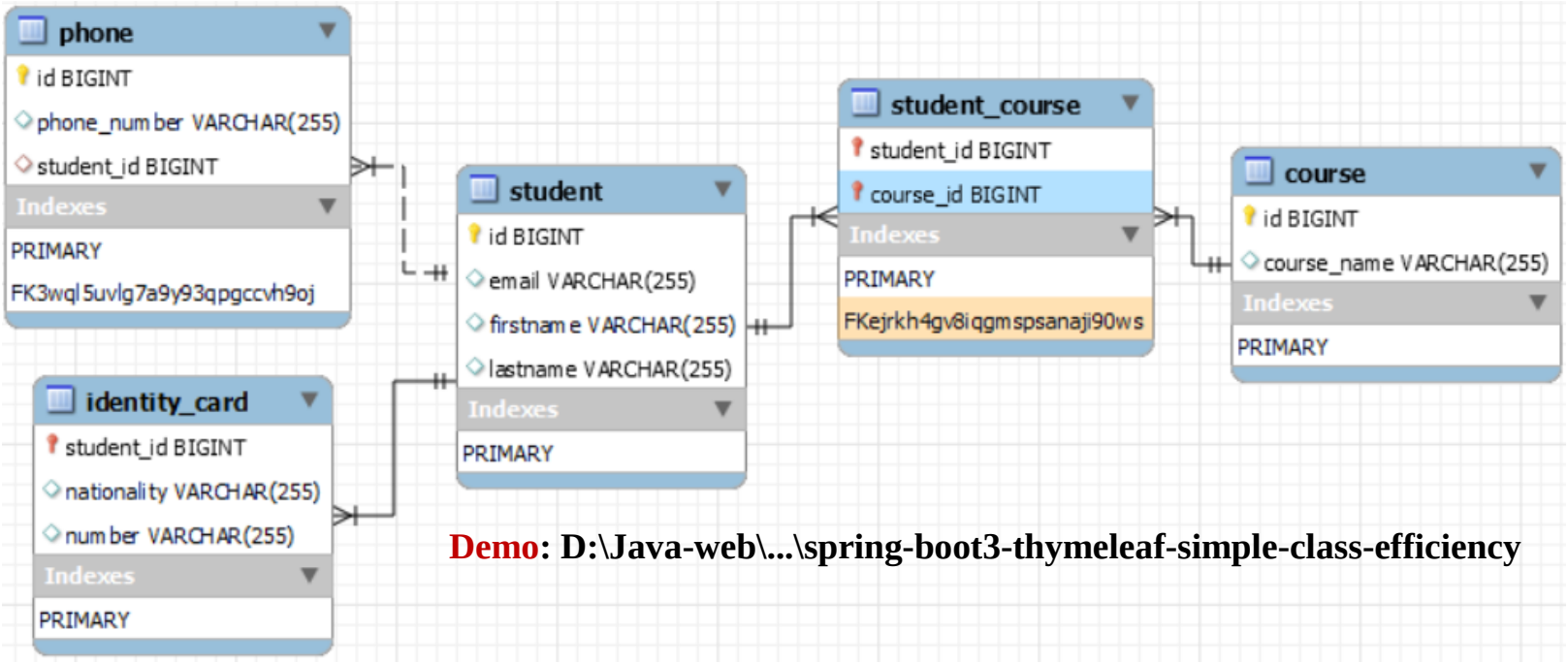
```java
@Service
public class BookstoreService {
   private @Autowired AuthorRepository authorRepository;
    // Fetch all authors and books (SQL)
   public List<AuthorNameBookTitle> fetchAuthorsAndBooksSql() {
      return authorRepository.findAuthorsAndBooksSql();
   }
}

@SpringBootApplication
public class MainApplication {
   private @Autowired BookstoreService bookstoreService;
   public static void main(String[] args) {
     SpringApplication.run(MainApplication.class, args);
   }
   @Bean
   ApplicationRunner init() {
       return args -> { System.out.format("%-20s %s%n","Author.Name","Book.Title");
                        bookstoreService.fetchAuthorsAndBooksSql().forEach(item ->
            System.out.printf("%-20s %-20s\n", item.getName(), item.getTitle())); };
  }
}
```

Javadoc | Declaration | Console × | Progress

spring-jpa-dto-via-full-joins - MainApplication [Spring Boot App] C:\sts-4.19.0.

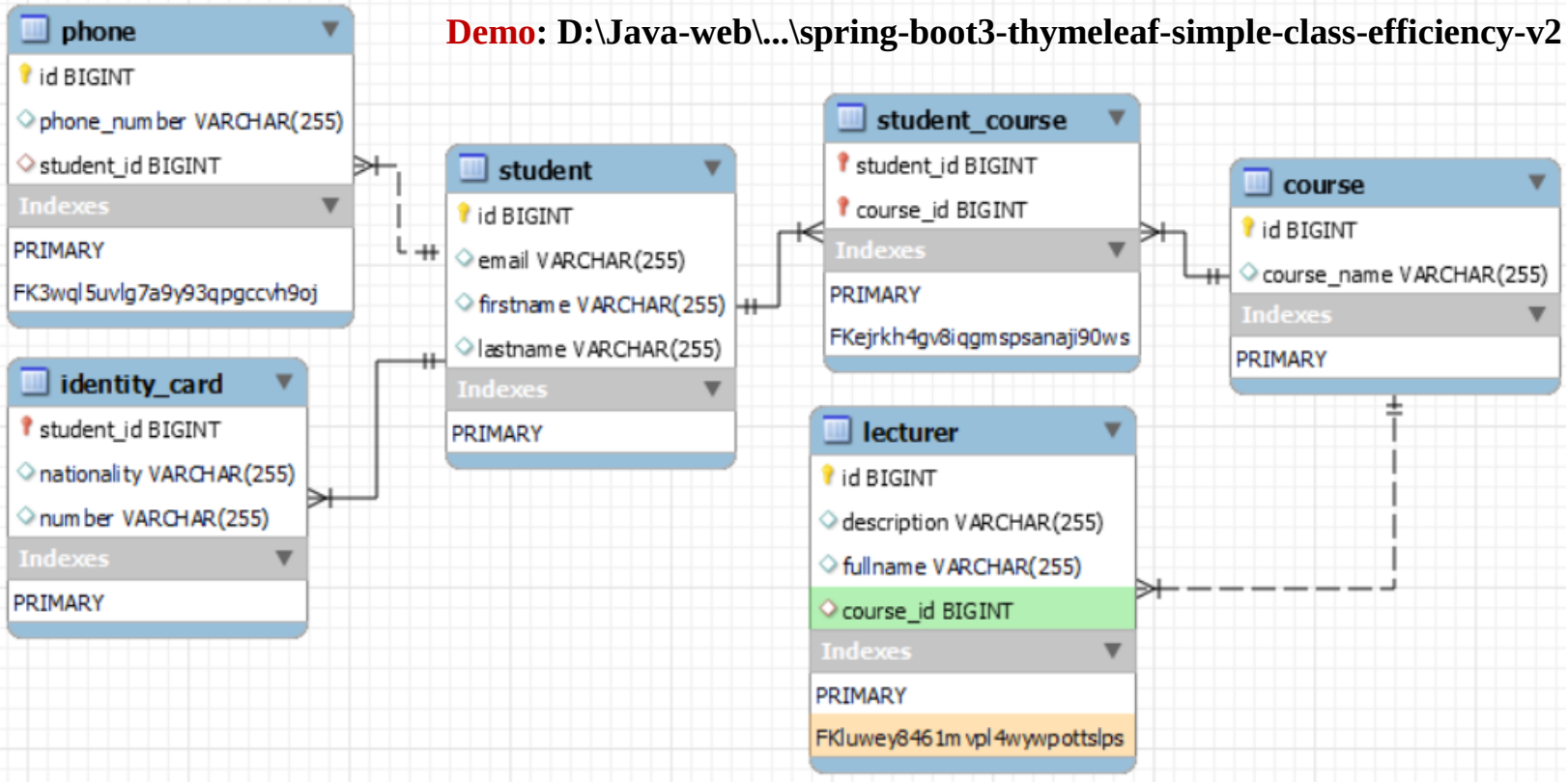
```
2023-08-15T10:32:39.759+07:00  INFO 11624 --- [
seconds (process running for 3.672)
Author.Name          Book.Title
Mark Janel           null
Olivia Goy           Carrie
Quartis Young        null
Joana Nimar          Waste Of History
Joana Nimar          A History of Ancient Prague
Larisa Tomay         null
null                 Lost book
```

- **Project1:** A simple class app with efficient associations (ver1)
  - Using Spring boot 3.1.x, JPA/Hibernate, students do this homework as below
  - Entity associations: Phone vs Student : m2o, IdentityCard vs Student : o2o via MapsId, Student vs Course: m2m bi-association
  - EER Diagram as below




**Demo: D:\Java-web\...\spring-boot3-thymeleaf-simple-class-efficiency**

- **Project2:** A simple class app with efficient associations (ver2)
  - Entity associations: Phone vs Student : m2o, IdentityCard vs Student : o2o via MapsId, Student vs Course: m2m bi-association, Lecturer vs Course: m2o association
  - EER Diagram as below



Student.java

```java
package com.javaweb.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "student")
public class Student  {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstname;
    private String lastname;
    private String email;

    @ManyToMany(cascade = CascadeType.MERGE, fetch = FetchType.LAZY)
    @JoinTable(name = "student_course",
            joinColumns = {@JoinColumn(name = "student_id")},
            inverseJoinColumns = {@JoinColumn(name = "course_id")})
    private Set<Course> courses = new HashSet<>();

    public Student(){}

    public String getFirstname() {
        return firstname;
    }

    public Long getId() {
        return id;
    }
```

Phone.java

```java
package com.javaweb.entity;


import jakarta.persistence.Entity;
@Entity
public class Phone {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String phoneNumber;

    @ManyToOne(fetch = FetchType.LAZY)
    private Student student;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }
```

IdentityCard.java

```java
package com.javaweb.entity;

import jakarta.persistence.Entity;

@Entity
public class IdentityCard {
    @Id
    private Long id;

    private String number;
    private String nationality;

    @MapsId
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "student_id")
    private Student student;

    public IdentityCard() {
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }
```

Course.java

```java
package com.javaweb.entity;
import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
@Table(name = "course")
public class Course{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String courseName;

    @ManyToMany(fetch = FetchType.LAZY, mappedBy = "courses")
    @JsonIgnore
    private Set<Student> students = new HashSet<>();

    public Course(){}

    public Course(String courseName){
        this.courseName = courseName;
    }

    public String getCourseName(){
        return courseName;
    }
```

Lecturer.java

```java
package com.javaweb.entity;

import jakarta.persistence.Entity;
@Entity
public class Lecturer {
    @Id @GeneratedValue(
            strategy=GenerationType.IDENTITY)
    private Long id;

    private String fullname;
    private String technologySpec; //Specializations;

  /* NOTE: Each Lecturer teaches only ONE Course */
    @ManyToOne(fetch = FetchType.LAZY)
    private Course course;

    public Long getId() {
        return id;
    }
```

SimpleClassVer2Application.java

```java
package com.javaweb;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import com.javaweb.service.LecturerService;
import com.javaweb.service.MyClassService;

@SpringBootApplication
public class SimpleClassVer2Application {
    @Autowired MyClassService myClassService;
    @Autowired LecturerService lecturerService;

    public static void main(String[] args) {
        SpringApplication.run(SimpleClassVer2Application.class, args);
    }

    @Bean   //Just run one time.
    ApplicationRunner init() {
        return args -> {
            myClassService.createCourses();
            myClassService.insertStudentsWithCourses();
            myClassService.insertIdentiyCardsForStudents();
            myClassService.insertPhoneForStudents();

            lecturerService.insertLecturersCourses();
        };
    }
}
```

**course table tupes**

| id | course_name |
|---|---|
| 1 | Not yet |
| 2 | Fundamental Java |
| 3 | Advanced Java |
| 4 | Master Java |
| 5 | Fundamental Golang |
| 6 | Advanced Golang |
| 7 | Master Golang |
| 8 | Fundamental JavaScript |
| 9 | Advanced JavaScript |
| 10 | Master JavaScript |
| 11 | Fundamental Python |
| 12 | Advanced Python |
| 13 | Fundamental C/C++ |
| 14 | Advanced C/C++ |
| 15 | Master C/C++ |
| NULL | NULL |

| | | | |
|---|---|---|---|
| 24 | Nguyen Nhi Gia Vinh-6b | C/C++ | 13 |
| 25 | Pham Nguyen Khang-6c | C/C++ | 14 |
| 26 | Phan Thuong Cang-6d | C/C++ | 14 |
| 27 | Pham Nguyen Khang-6e | C/C++ | 15 |
| 28 | Vu Duy Linh-6f | C/C++ | 15 |
| NULL | NULL | NULL | NULL |

**lecturer table tupes**

| id | fullname | technology_spec | course_id |
|---|---|---|---|
| 1 | Vu Duy Linh-2a | Java | 2 |
| 2 | Phan Huy Cuong-2b | Java | 2 |
| 3 | Doan Hien-2c | Java | 3 |
| 4 | Vu Duy Linh-2d | Java | 3 |
| 5 | Phan Huy Cuong-2e | Java | 3 |
| 6 | Vu Duy Linh-2f | Java | 4 |
| 7 | Phan Huy Cuong-3a | Golang | 5 |
| 8 | Nguyen Nhi Gia Vinh-3b | Golang | 1 |
| 9 | Phan Huy Cuong-3c | Golang | 12 |
| 10 | Nguyen Duc Khoa-3d | Golang | 6 |
| 11 | Vu Duy Linh-3e | Golang | 7 |
| 12 | han Huy Cuong-3f | Golang | 7 |
| 13 | Nguyen Duc Khoa-4a | JavaScript | 8 |
| 14 | Nguyen Duc Khoa-4b | JavaScript | 1 |
| 15 | Doan Hien-4c | JavaScript | 1 |
| 16 | Le van Quan-4e | JavaScript | 9 |
| 17 | Vu Duy Linh-4e | JavaScript | 10 |
| 18 | Le Van Quan-4f | JavaScript | 1 |
| 19 | Vu Duy Linh-5a | Python | 11 |
| 20 | Tran Ngan Binh-5b | Python | 11 |
| 21 | Tran Ngan Binh-5c | Python | 12 |
| 22 | Phan Thuong Cang-5d | Python | 12 |
| 23 | Pham Nguyen Khang-6a | C/C++ | 13 |

**Note: Every lecturer teaches only one course.**

**student table**

| id | email | firstname | lastname |
|---|---|---|---|
| 1 | vdthuc@gmail.com | Duy Thức | Vũ |
| 2 | vhty@gmail.com | Hồng Thiên Ý | Vũ |
| 5 | namcuong@gmail.com | Nam Cường | Nguyễn |
| 6 | pngiau@hotmil.com | Ngọc Giàu | Phạm |
| NULL | NULL | NULL | NULL |

**phone table**

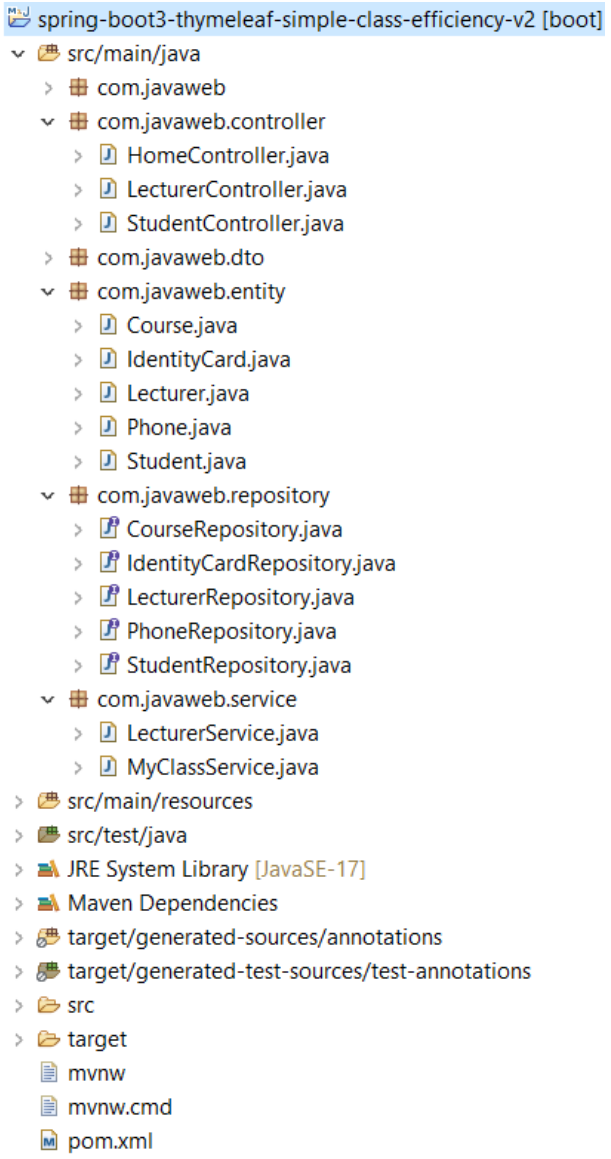
| id | phone_number | student_id |
|---|---|---|
| 1 | 0907856789 | 1 |
| 2 | 0907222789 | 1 |
| 3 | 0907856777 | 2 |
| 4 | 0907222778 | 2 |

**identity_card table**

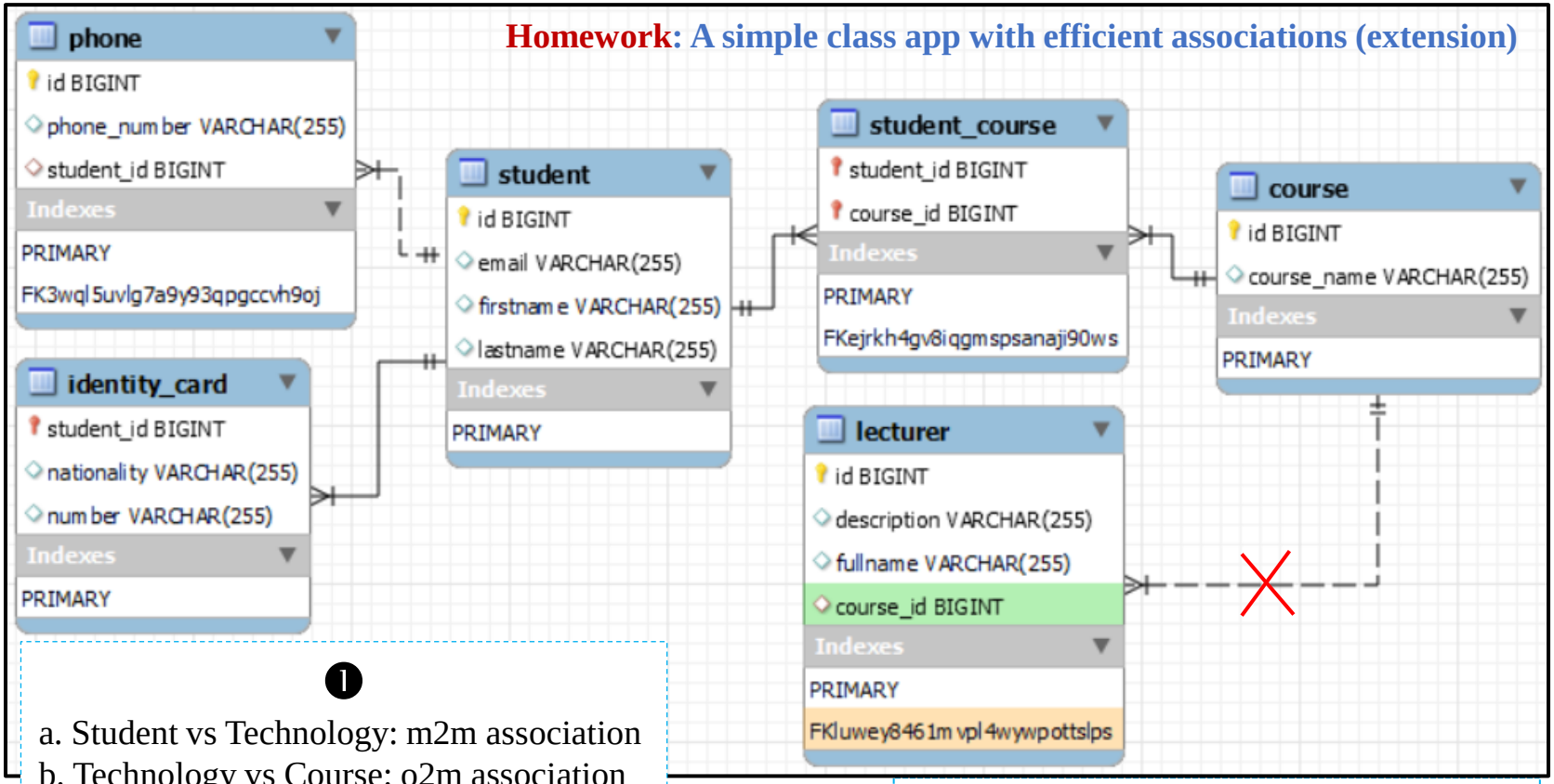
| student_id | nationality | number |
|---|---|---|
| 1 | VN | 189123456789 |
| 2 | VN | 179123456789 |

**student_course**

| student_id | course_id |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 2 | 3 |
| 1 | 4 |
| 2 | 4 |
| 1 | 5 |
| NULL | NULL |



**Homework: A simple class app with efficient associations (extension)**




❶

a. Student vs Technology: m2m association
b. Technology vs Course: o2m association
⇔ Course vs Technology: m2o association
c. Technology vs Lecturer: o2m association
[Technology vs Lecturer: m2m association]

❷

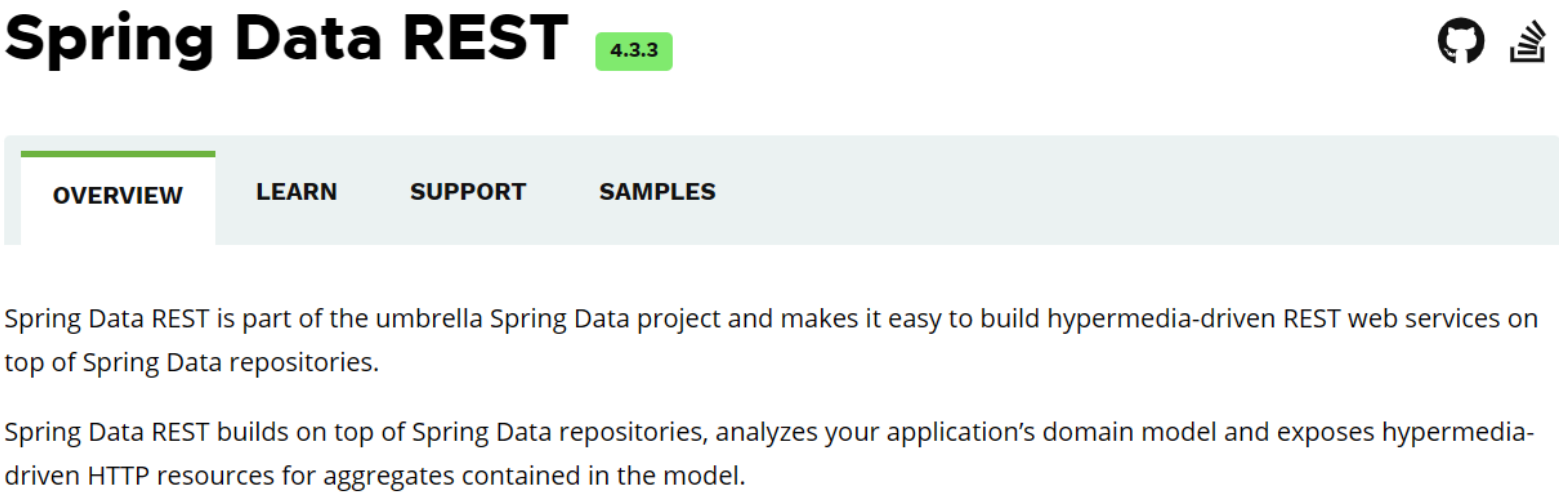
a. Student vs Course: m2m bi-association
b. Course vs Technology: m2o association
c. Technology vs Lecturer: o2m association
[Technology vs Lecturer: m2m association]

❸

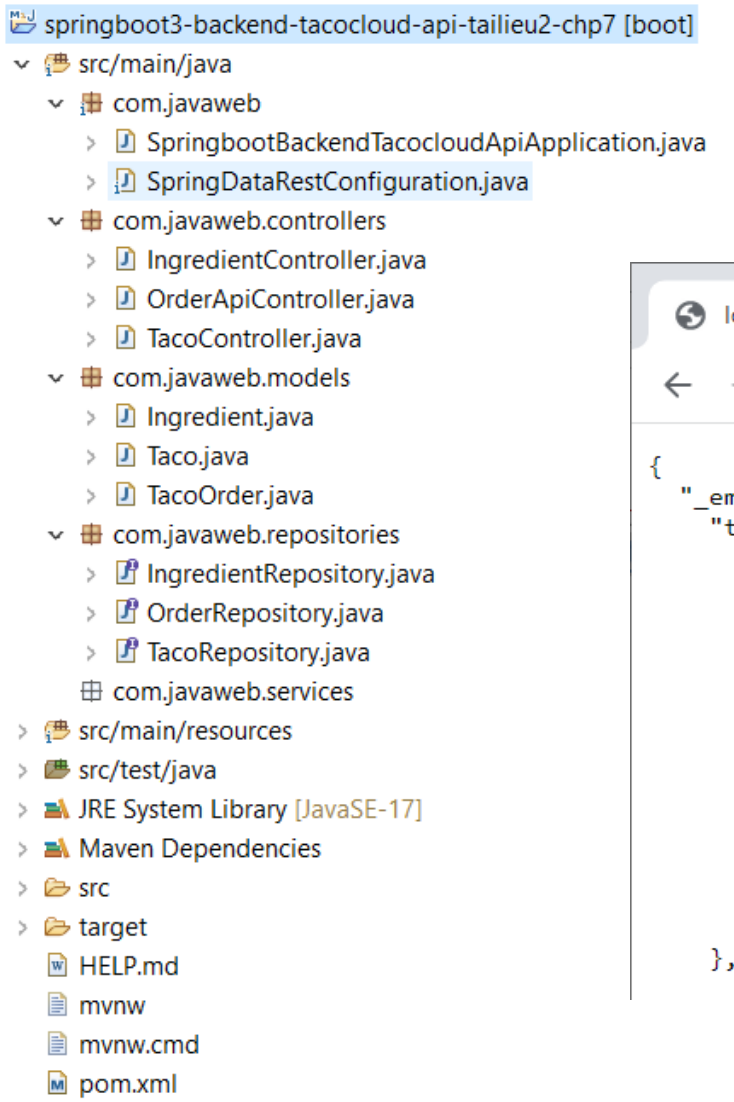
a. Types of associations ???
b. JPA and Hibernate m2m association with extra columns

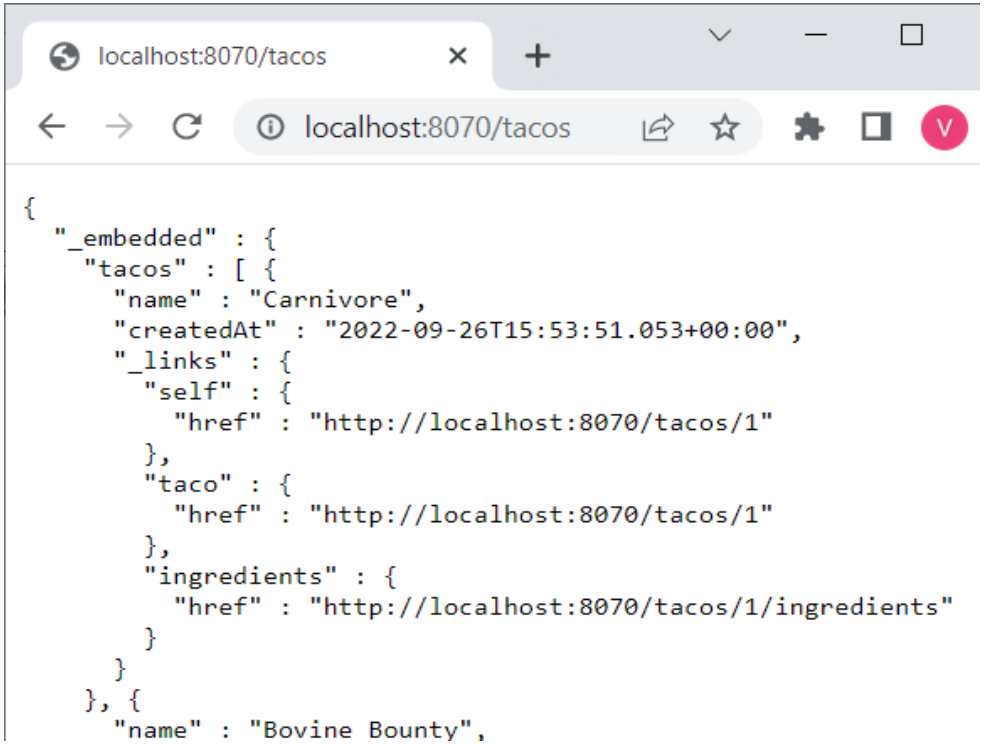
- **REST web services**
  - have become the number one means for application integration on the web.
  - In its core, REST defines that a system that consists of resources (implemented in a hypermedia-driven way) with which clients interact.

  ➔ This topic will be learned in chapter 5.
- **Spring Data REST**
  - builds on top of the Spring Data repositories and automatically exports those as REST resources.
  - leverages hypermedia to let clients automatically find functionality exposed by the repositories and integrate these resources into related hypermedia-based functionality.
  - **Official link:** https://docs.spring.io/spring-data/rest/reference/



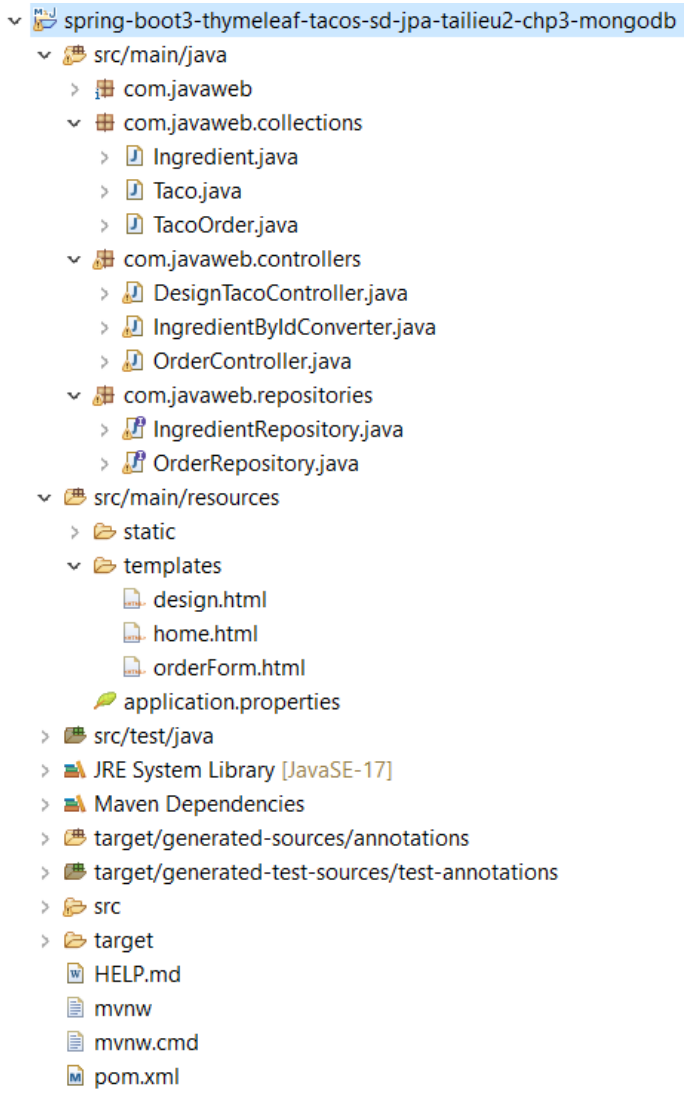
- **springboot3-backend-tacocloud-api-tailieu2-chp7** (more detailed in Chapter 5)



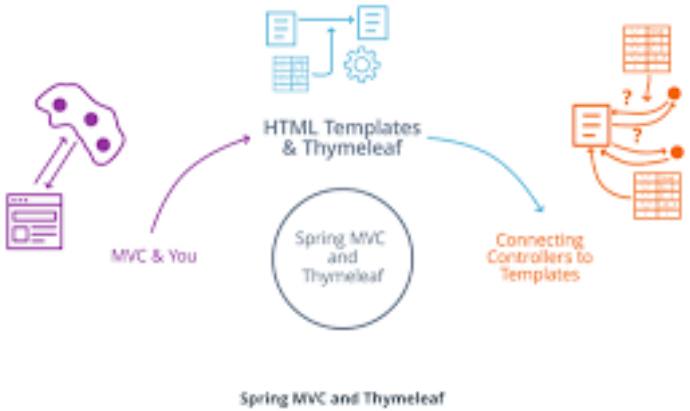
- **Demo: spring-boot3-thymeleaf-tacos-sd-jpa-tailieu2-chp3-mongodb**
  - MongoRepository
  - Demo some Spring web MVC/Rest API projects
  - This is an **optional section**


```
_id: ObjectId('64babbe716f9a60080ed2f8d')
placedAt: 2023-07-21T17:07:08.739+00:00
deliveryName: "Vu Duy Thong"
deliveryStreet: "BB2"
deliveryCity: "Cantho"
deliveryState: "Cantho"
deliveryZip: "9000"
ccNumber: "5105105105105100"
ccExpiration: "12/25"
ccCVV: "123"
tacos: Array (1)
  0: Object
    name: "Vu Duy Linh-CTU"
    createdAt: 2023-07-21T17:07:26.713+00:00
    ingredients: Array (5)
      0: Object
        _id: "FLTO"
        name: "Flour Tortilla"
        type: "WRAP"
      1: Object
      2: Object
      3: Object
      4: Object
_class: "com.javaweb.collections.TacoOrder"
```

# Building web applications with Java

## Chap 4. Spring Web MVC (Security) with Thymeleaf 3.0.15

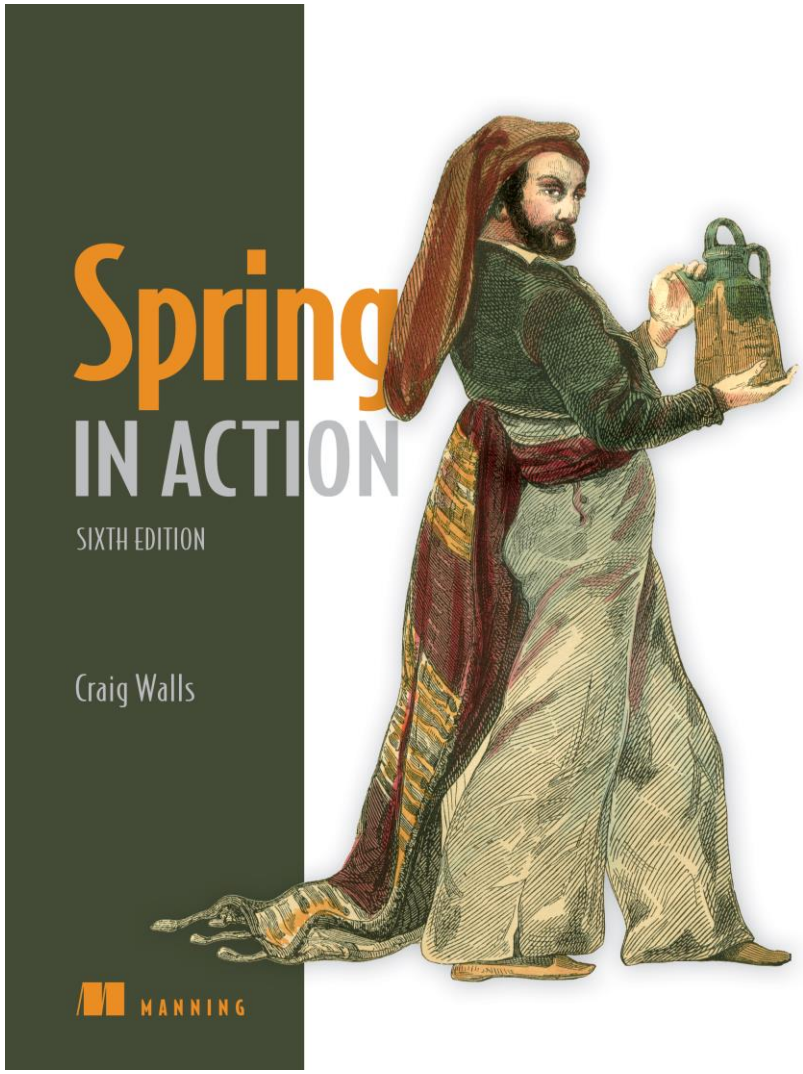



Spring MVC and Thymeleaf

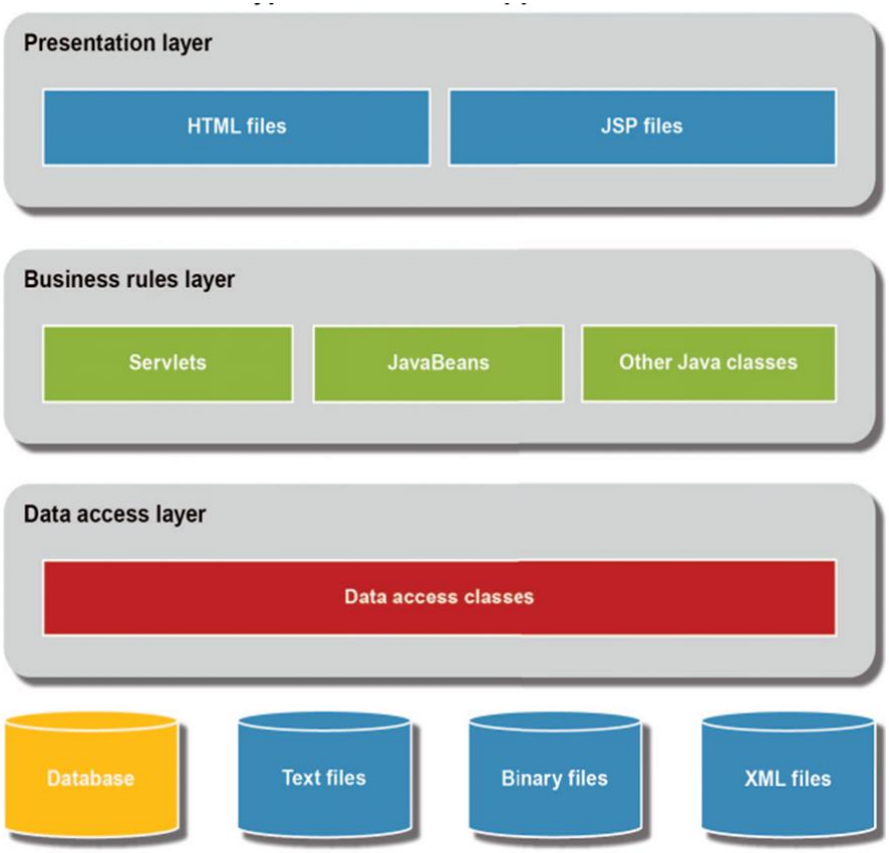
**Biên soạn: Vũ Duy Linh**

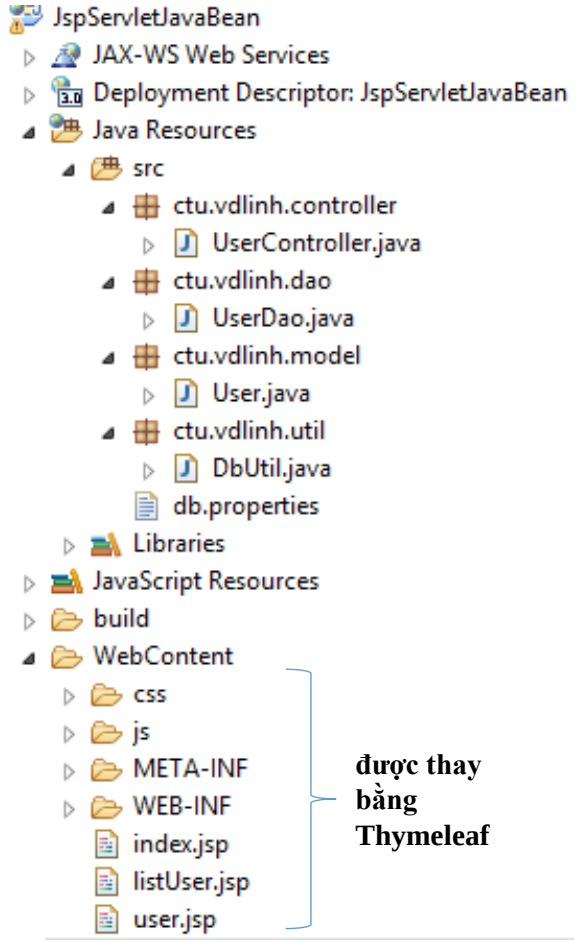
vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

**Advice book:**

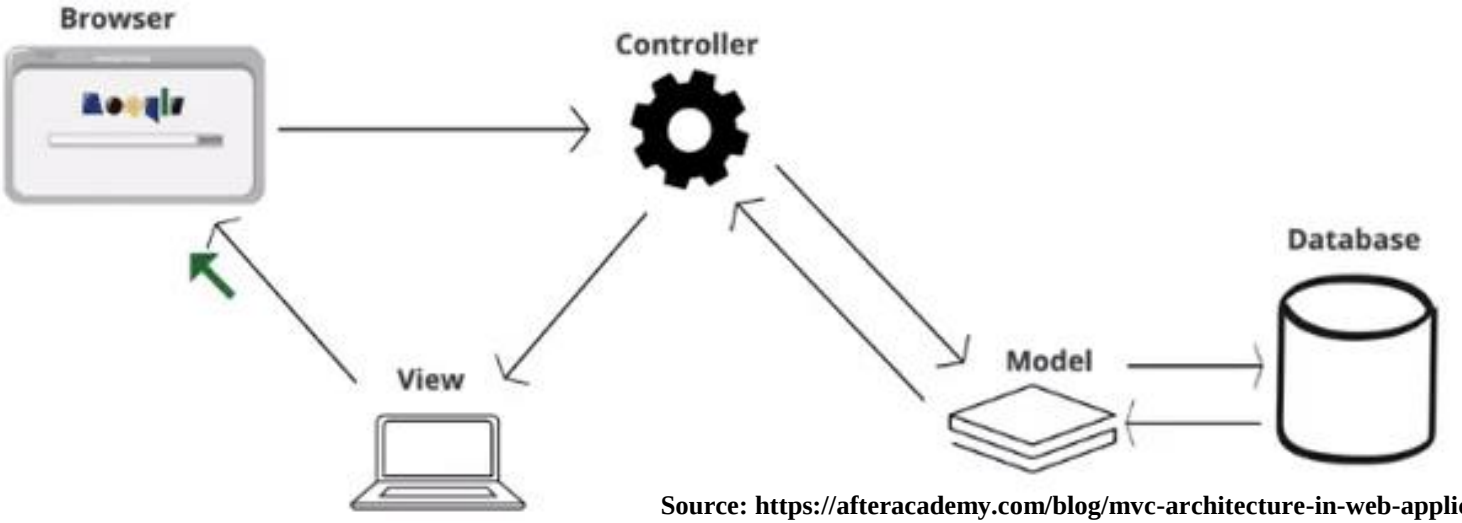


- **Three layers architecture for Servlet and JSP/Thymeleaf**
  - **Presentation Layer**: JSP/Thymeleaf + HTML
  - **Business/Service Layer**: Java/Jakarta Servlet, Beans /Spring Beans
  - **Data Access Layer**: JPA/Hibernate, Spring Data



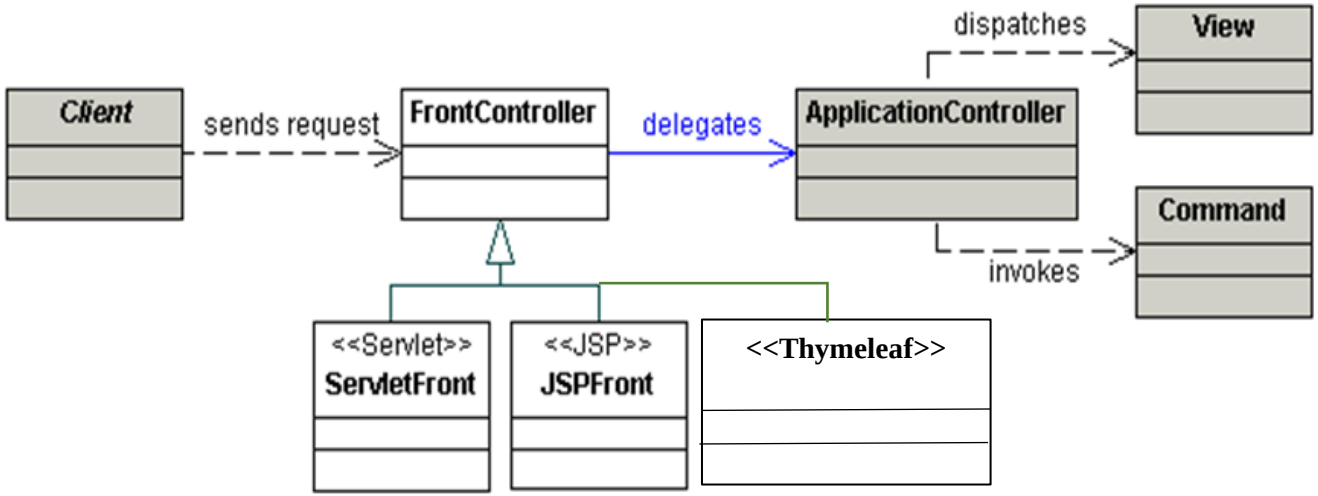



- **MVC Pattern:** is a design pattern that separates the business logic, presentation logic and data.
  - **Controller:** Defines application behavior. It dispatches user requests and selects views for presentation.
    - It interprets user inputs and maps them into actions to be performed by the model.
    - it selects the next view to display based on the user interactions and the outcome of the model operations.
  - **Model:** The classes which are used to store and manipulate state of the application i.e. data. It can also have business logic (xử lý nghiệp vụ).
  - **View:** represents the presentation i.e. UI (User Interface).



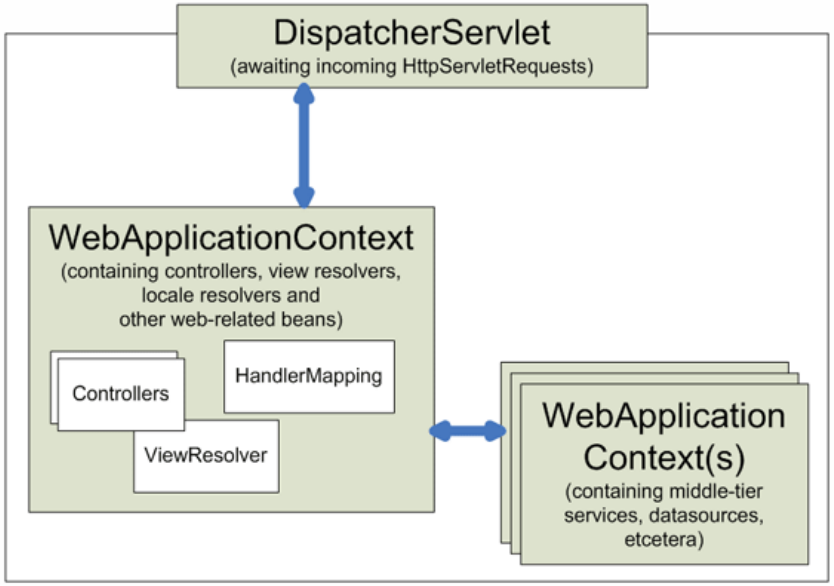

Source: https://afteracademy.com/blog/mvc-architecture-in-web-applications/

- **Mẫu thiết kế Front Controller**
  - là mẫu thiết kế được áp dụng trong bối cảnh phát triển các ứng dụng web
  - tất cả các yêu cầu (request) đến tài nguyên trong ứng dụng đều được tiếp nhận xử lý đầu tiên bởi một bộ phận trung tâm duy nhất (được gọi là Front Controller).
  - Front Controller tập trung các logic điều khiển (có thể bị trùng lặp) và quản lý các hoạt động xử lý yêu cầu chính (the key request handling).



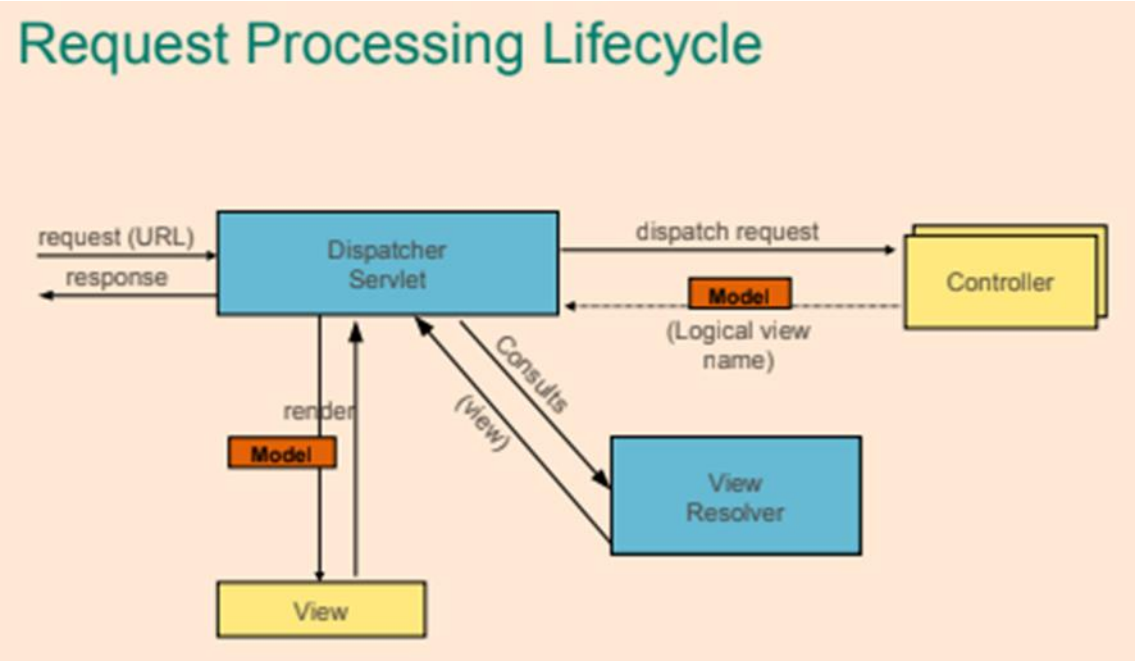

Source: http://www.corej2eepatterns.com/

- **Mô hình Spring MVC (Spring web MVC framework)**
  - Spring MVC is based on Model-View-Controller architecture
  - là một framework có kiến trúc hướng yêu cầu (Request-Driven architecture), được thiết kế xung quanh một Servlet trung tâm, đó chính là DispatcherServlet (hay là FrontController), để gửi các request tới Controllers và các chức năng khác tạo thuận lợi cho việc phát trển các ứng dụng web.




  - Với **HanderMapping: "Trình xử lý cấu hình các URL"** để ánh xạ các yêu cầu web tới các trình xử lý (Handler methods/Controllers) thích hợp. ViewResolver: Trình phân giải (xử lý) View, trong bài giảng sử dụng Thymeleaf view template engine.

- **Luồng làm việc của Spring MVC**




Về mặt cấu hình trong Spring MVC, một **Request URL** có dạng như sau:

**Scheme://Host:Port/Context path/Servlet path/Path info?Parameter(s)**

- Chú ý: Đối với chương trình web được viết theo **Spring MVC Framework** (không sử dụng Spring Boot), thì lập trình viên phải viết cấu hình cho phần Front Controller để web hoạt động được.

Với các thành phần được hiểu như sau:

- **Context path** được gọi là đường dẫn ngữ cảnh, nó là đường dẫn mà toàn bộ ứng dụng web được ánh xạ tới. Từ phiên bản Java Servlet 4.0, cũng như hiện nay là Jakarta Servlet 6.0, Context path bắt đầu bằng ký tự "/" và ghép với tên của dự án web (Web application name), ví dụ như: "**/webstore**".

- **Servlet path** là ánh xạ servlet (servlet-mapping, còn được gọi là **Servlet mapping**) bằng **url-pattern** để định nghĩa một mẫu đường dẫn URL, chẳng hạn như "/*", để hướng tất cả các yêu cầu HTTP đến một servlet trung tâm (Front Controller) thông qua **servlet-class** là **org.springframework.web.servlet.DispatcherServlet**, để từ FrontController (hoặc Spring MVC DispatcherServlet) này có thể gửi yêu cầu người dùng đến phương thức xử lý mong muốn.

- **Request mapping** được gọi là ánh xạ yêu cầu tới (các) phương thức xử lý (handler methods) tương ứng trong Controllers.

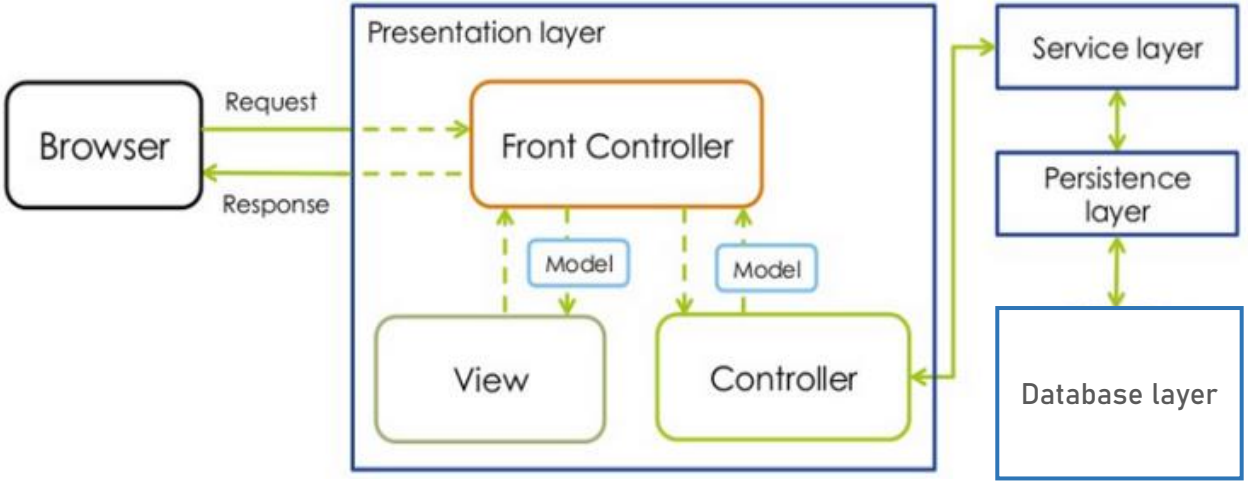
- **Path info**: là đường dẫn bổ xung (extra path) được tính từ sau servlet name đến trước phần parametters (query string(s), nó có thể null hoặc là một đường dẫn bổ sung có thực (real path) được cấu hình trong web.

- **Ví dụ:** Với Request URL là **http://localhost:8080/webstore/products/search?pid=123**, các thành phần cấu hình trong Spring MVC là:

  Context path: **/webstore** , Servlet mapping: **/**

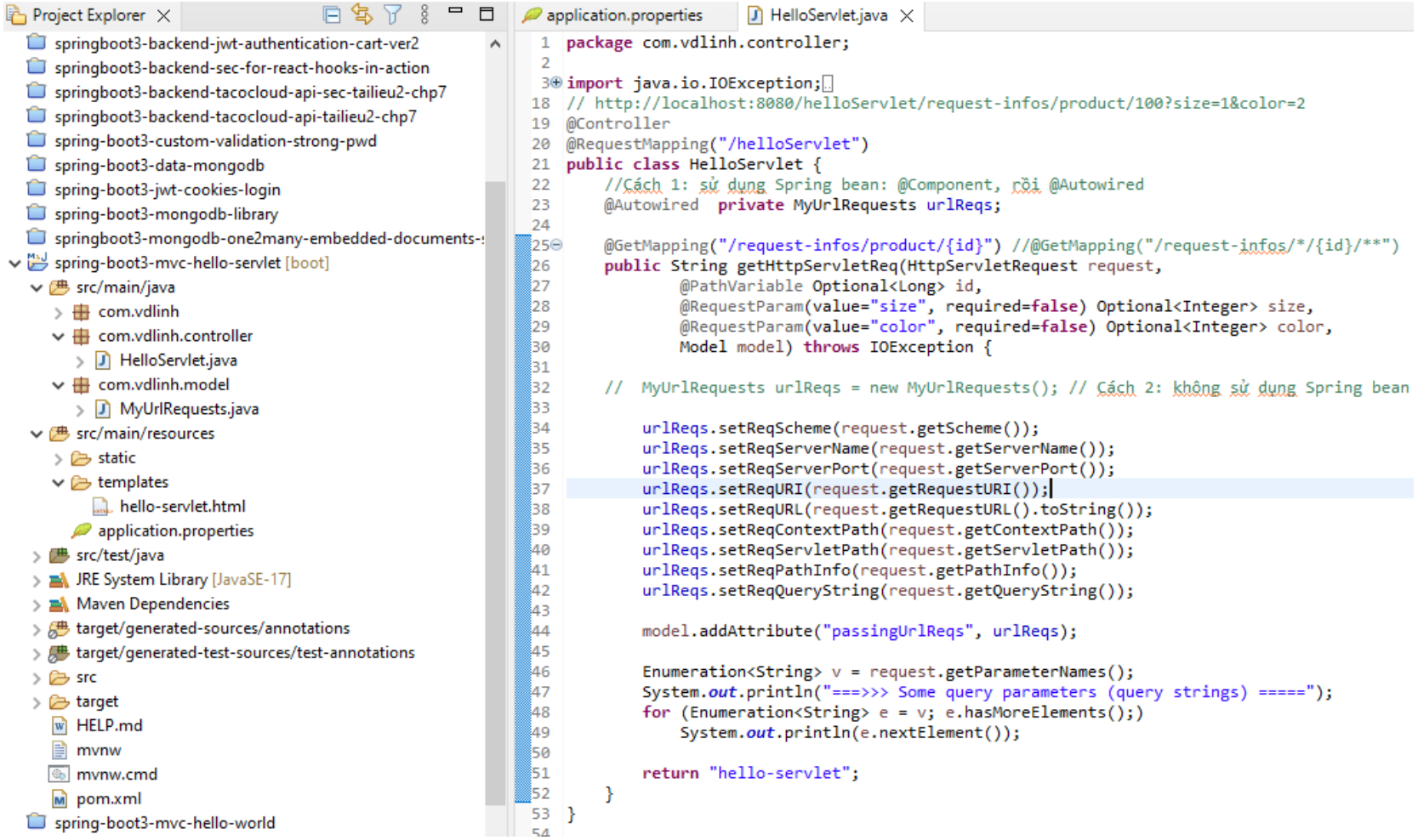
  Request mapping: **/products/search** query string: **?pid = 123**

- **Mô hình Spring Web MVC sử dụng Spring Boot** (gọi tắt là **Spring Boot MVC**)
  - Spring Boot MVC cũng tuân theo kiến trúc phân lớp (Layered architecture), bao gồm các lớp: Trình bày (**Presentation** layer), Dịch vụ (**Service** layer), Dữ liệu bền vững (**Persistence** layer: usually contains repository objects to access domain objects), và Dữ liệu (Database Layer).

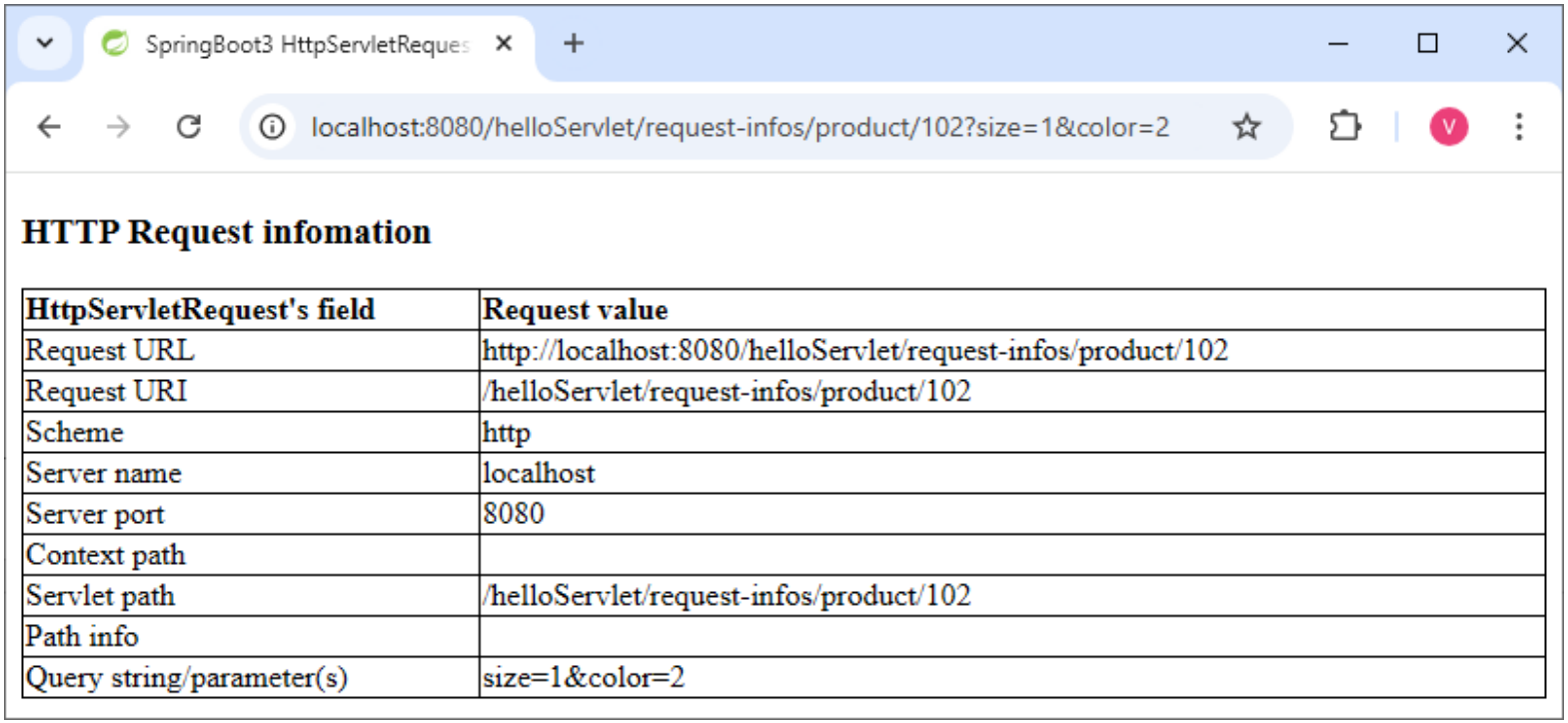



  - Điểm nổi bật của **Spring Boot** là cung cấp khả năng tự động cấu hình (auto-configured) một cách mặc định để giúp Spring MVC [với bộ mẫu hiển thị Thymeleaf] hoạt động tốt với hầu hết các ứng dụng, giúp cho việc cấu hình trở nên trong suốt đối với lập trình viên. Có thể nói, nhờ có Spring Boot, lập trình viên theo công nghệ Java (web) đã được cởi bỏ tất cả các gánh nặng mà trước đây họ phải mất nhiều công sức để chọn một archetype-webapp có cấu trúc dự án phù hợp, cũng như các bước cấu hình (XML) phức tạp cho những dự án Java web MVC framework như: Servlet/JSP, Strut 2, JSF, Spring MVC.

- **Demo D:\Java-web\spring-thymeleaf\spring-boot3-mvc-hello-servlet**



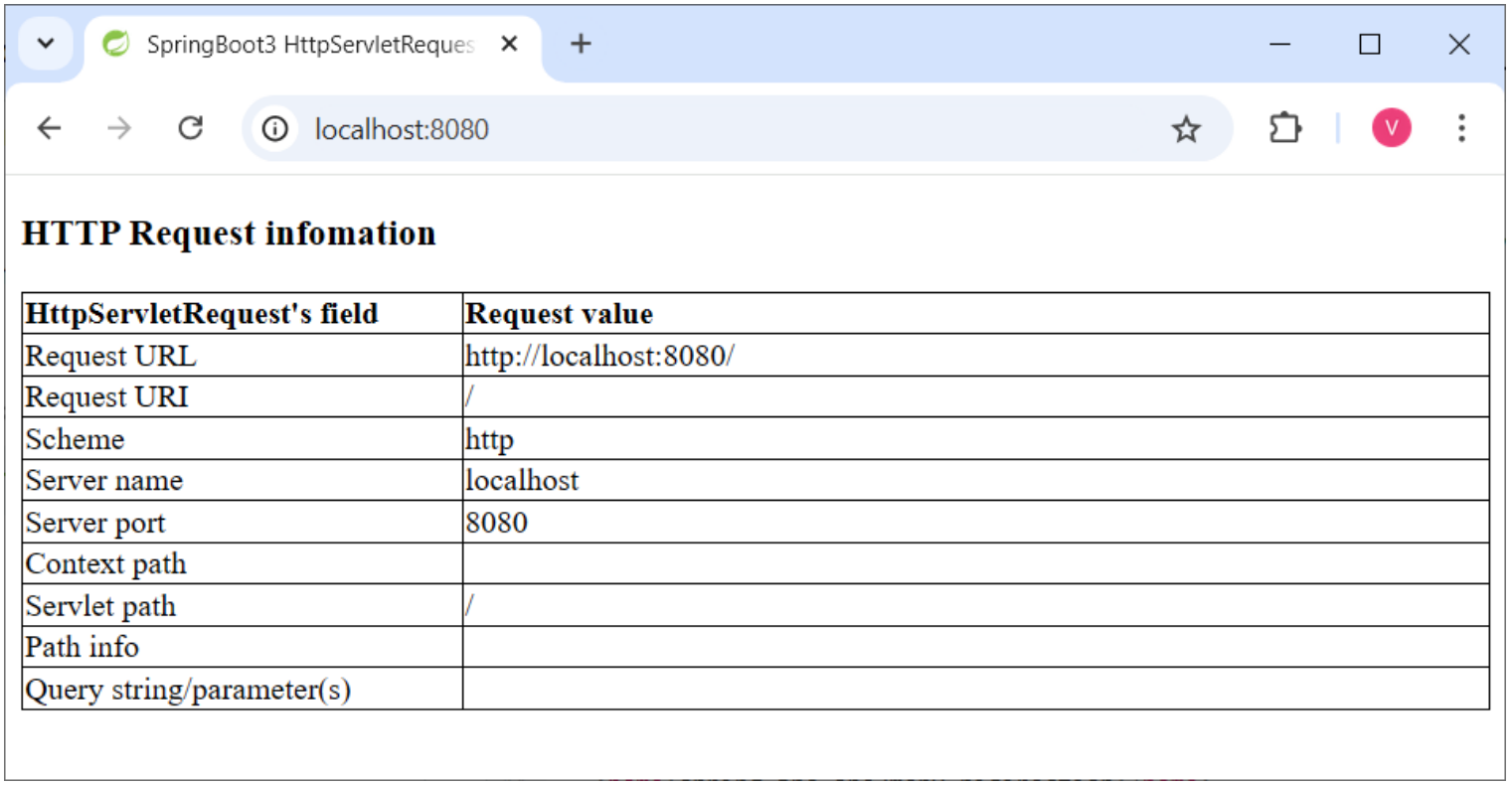
- Chạy chương trình, từ thanh địa chỉ của trình duyệt, gõ Request URL là: http://localhost:8080/helloServlet/request-infos/product/102?size=1&color=2, ta được các thông số cấu hình mặc định của Spring Boot MVC như sau




**HTTP Request infomation**

| HttpServletRequest's field | Request value |
|---|---|
| Request URL | http://localhost:8080/helloServlet/request-infos/product/102 |
| Request URI | /helloServlet/request-infos/product/102 |
| Scheme | http |
| Server name | localhost |
| Server port | 8080 |
| Context path | |
| Servlet path | /helloServlet/request-infos/product/102 |
| Path info | |
| Query string/parameter(s) | size=1&color=2 |

- Nếu **bỏ đi @RequestMapping("/helloServlet")** ở mức class, và chỉnh lại mapping ở mức method chỉ còn là **@GetMapping**, chạy chương trình ta được:

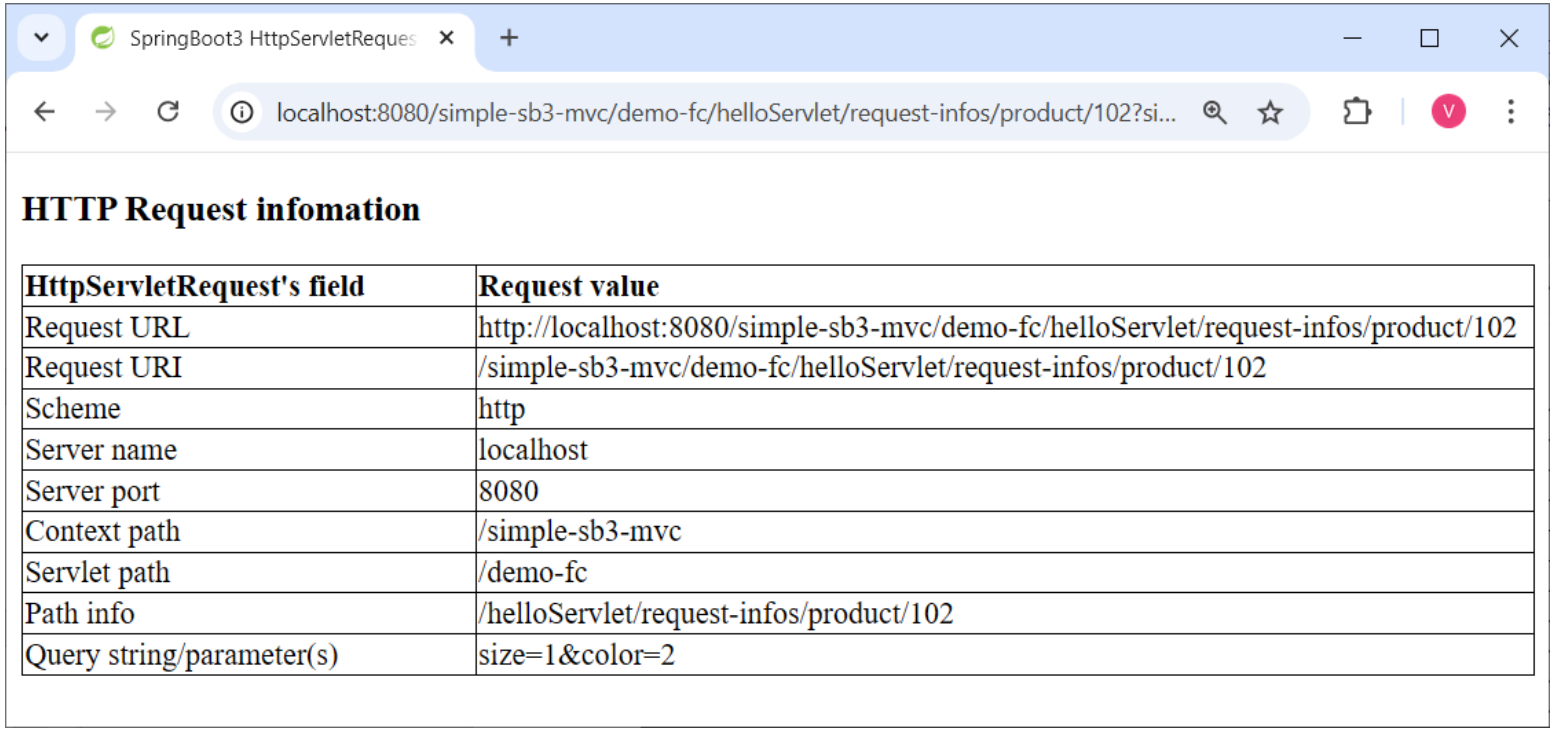


- Trong trường hợp cần thiết, lập trình viên có thể tùy chỉnh lại cấu hình (Embedded Servlet Containers) của bộ chứa Servlet web thông qua tập tin application.properties.

  **server.servlet.context-path=/simple-sb3-mvc**

  **spring.mvc.servlet.path=/demo-fc**

- Từ trình duyệt, gõ Request URL là: http://localhost:8080/simple-sb3-mvc/demo-fc/helloServlet/request-infos/product/102?size=1&color=2 ta được các thông số sau:




**HTTP Request infomation**

| HttpServletRequest's field | Request value |
|---|---|
| Request URL | http://localhost:8080/simple-sb3-mvc/demo-fc/helloServlet/request-infos/product/102 |
| Request URI | /simple-sb3-mvc/demo-fc/helloServlet/request-infos/product/102 |
| Scheme | http |
| Server name | localhost |
| Server port | 8080 |
| Context path | /simple-sb3-mvc |
| Servlet path | /demo-fc |
| Path info | /helloServlet/request-infos/product/102 |
| Query string/parameter(s) | size=1&color=2 |

- There are essential annotations and Jakarta servlets:
  - @SessionAttributes
  - @ModelAttribute
  - @RequestMapping
  - @Getmapping
  - @Postmapping
  - @PutMapping
  - @PatchMappin
  - @DeleteMapping
  - @ControllerAdvice
  - @Controller
  - @RestController
  - @Service
  - @SessionScope
  - @Valid
  - @PathVariable
  - @RequestParam
  - @RequestBody
  - @Configuration
  - @EnableWebSecurity
  - @EnableMethodSecurity
  - @CrossOrigin
  - …
- Some latest Spring APIs:
  - org.springframework.ui.Model
  - org.springframework.ui.ModelMap
  - org.springframework.web.servlet.ModelAndView
  - org.springframework.web.servlet.view.RedirectView
  - …
- Some latest Jakarta APIs in package jakarta.servlet.http
  - Interface HttpServletMapping
  - Interface HttpServletRequest
  - Interface HttpServletResponse
  - Interface HttpSession
  - …

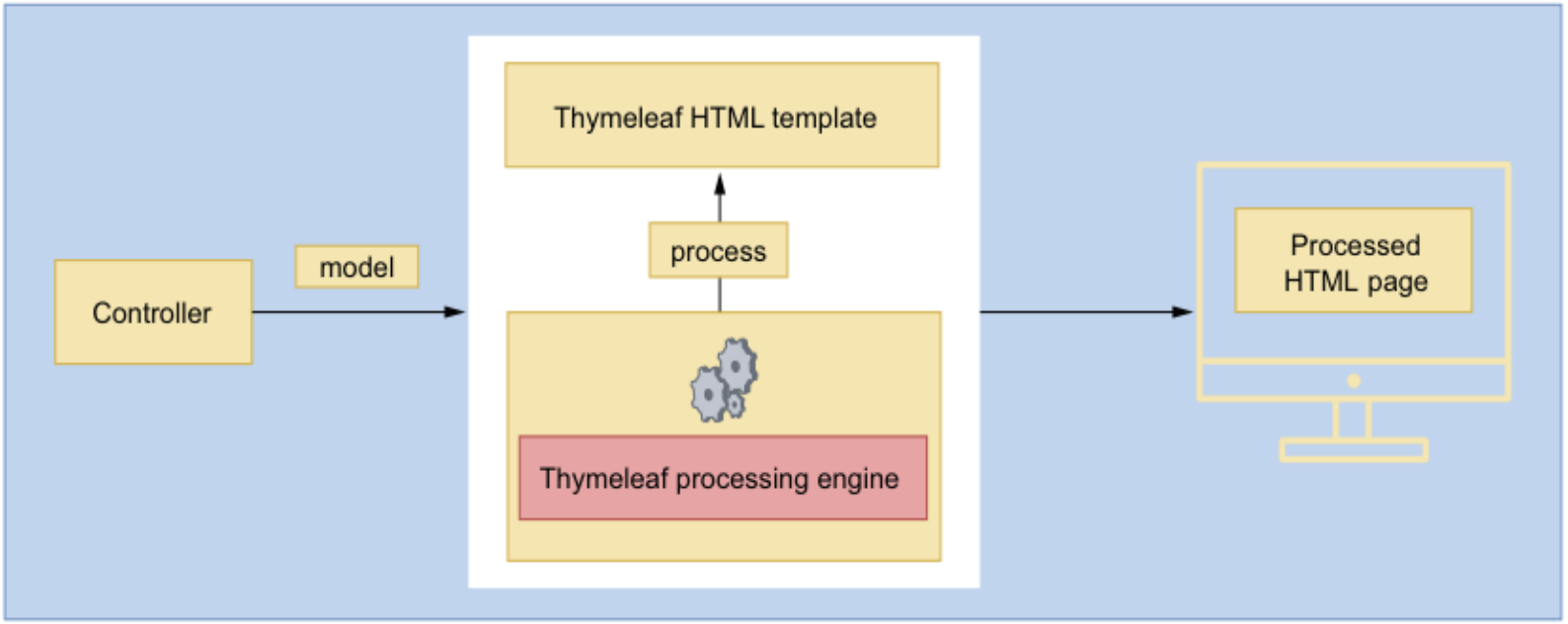
Refs: https://docs.spring.io/spring-framework/reference/web/webmvc.html
https://docs.spring.io/spring-boot/docs/current/reference/html/web.html
https://jakarta.ee/specifications/platform/10/apidocs/jakarta/servlet/http/package-summary.html

- Official link: https://www.thymeleaf.org/index.html


**Thymeleaf** is a modern server-side Java template engine for both web and standalone environments.

- Thymeleaf 3.x:
  - là công cụ tạo mẫu Java theo cách trang nhã và hợp lý phía máy chủ (Java Template Engine), cho phép xử lý 6 loại mẫu Thymeleaf gồm: hai chế độ mẫu đánh dấu (markup) là HTML và XML, ba chế độ mẫu kiểu văn bản (textual template) là TEXT, JAVASCRIPT, CSS và chế độ RAW.
  - Trong số các loại này, mẫu dựa trên HTML là mẫu phổ biến nhất và thường được sử dụng để phát triển các ứng dụng Web dựa trên Java.
  - Mẫu HTML Thymeleaf là một trang HTML được khai báo namespace bằng thuộc tính ngoại **xmlns:th= "http://www.thymeleaf.org"**, và các phần tử HTML có chứa các thuộc tính là các thẻ Thymeleaf (tag attributes/ th:* attributes).
  - Namespace **xmlns:th** được khai báo trong thuộc tính của phần tử html, nhằm mục đích cung cấp phạm vi cho tất cả thuộc tính **th:*** (th:* attributes) trong trang HTML.
  - Nếu có xử lý bảo mật trong Thymeleaf, cần thêm namespace **xmlns:sec= "http://www.thymeleaf.org/thymeleaf-extras-springsecurity6"** ngay sau namespace **xmlns:th**.
  - Các thẻ thuộc tính Thymeleaf này được Thymeleaf engine xử lý trong thời gian chạy và được thay thế bằng dữ liệu được cung cấp để render ra các file tĩnh chứa đầy đủ dữ liệu truyền vào.

- Sơ đồ xử lý các mẫu Thymeleaf với mô hình Spring Boot MVC




- Từ Controller, phương thức xử lý được ánh xạ bởi request mapping sẽ xử lý logic nghiệp vụ và gán các thuộc tính cần thiết vào Model, và được Spring Boot MVC chuyển chúng đến mẫu HTML Thymeleaf bằng cách sử dụng các thuộc tính **th:***, hoặc **sec:***, cũng như các biểu thức **expressions**. Tiếp đến, Thymeleaf engine xử lý các thẻ và biểu thức này bằng dữ liệu của Model đã được cung cấp, kết quả sẽ tạo ra trang HTML đã được xử lý.

- Phương ngữ chuẩn (Dialects: The Standard Dialect)
  - Thymeleaf là một công cụ tạo mẫu có khả năng mở rộng cao (còn được gọi là Khung công cụ tạo mẫu, Template engine framework), cho phép xác định và tùy chỉnh cách xử lý các mẫu ở mức độ chi tiết tốt. Thư viện cốt lõi (core) của Thymeleaf cung cấp một phương ngữ được gọi là Phương ngữ chuẩn (Standard Dialect), đủ tính năng sử dụng cho tất cả mọi người.
  - Spring Framework tạo ra một phương ngữ riêng gọi là SpringStandard Dialect, về cơ bản là giống như phương ngữ chuẩn của Thymeleaf, nhưng với các điều chỉnh nhỏ để sử dụng tốt hơn một số tính năng trong Spring Framework, Chẳng hạn sử dụng ngôn ngữ biểu thức Spring (Spring Expression Language - SpringEL) thay vì sử dụng OGNL (Object-Graph Navigation Language).
  - Hầu hết các bộ xử lý của Phương ngữ Chuẩn là bộ xử lý thuộc tính (***attribute processors***) . Điều này cho phép trình duyệt hiển thị chính xác các tệp mẫu HTML ngay cả trước khi được xử lý vì chúng sẽ đơn giản bỏ qua các thuộc tính bổ sung.
- Một số thuộc tính/thẻ của Thymeleaf (**th:*)** và các biểu thức (expression) của Thymeleaf Standard Dialect:
  - **Cú pháp thẻ chuẩn (th:* attribute/tag):** th:insert, th:replace, th:each, th:if, th:unless, th:switch, th:case, th:action, th:object, th:with, th:attr, th:attrprepend, th:attrappend, th:field, th:value, th:href, th:src, th:text, th:utext, th:fragment, v.v…

- **Cú pháp biểu thức chuẩn (Standard Expression syntax):**

**${...}** : Biểu thức biến (Variable expressions).

***{...}** : Biểu thức lựa chọn (Selection expressions).

**#{...}** : Biểu thức thông báo i18n (Message i18n expressions).

**@{...}** : Biểu thức liên kết URL (Link URL expressions), lấy đường dẫn request URL dựa vào Request mapping.

**~{...}** : Biểu thức phân đoạn (Fragment expressions).

**|Some text ${...}|** : Thay thế theo nghĩa đen (Literal substitutions)

**[[${...}]]** : Biểu thức nội tuyến (Expression inlining)

**(if) ? (then), (if) ? (then) : (else), (value) ?: (defaultvalue)** : Toán tử điều kiện (Conditional operators),
V.v...

- Sinh viên tìm hiểu đầy đủ hơn tại địa chỉ: https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html

- In Spring Boot MVC with Thyemleaf view sample

**URL: http://localhost:8070/**

HomeController.java ❶

```
1  package com.springweb.controller;
2
3  import org.springframework.stereotype.Controller;
6
7  @Controller
8  public class HomeController {
9      @GetMapping("/")
10     public String home(Model model) {
11         model.addAttribute("webTech", "Spring Web MVC security via Spring Boot 3.1.x");
12         model.addAttribute("projectName", "Simple shopping cart with security 6.1.x");
13         return "home";
14     }
15 }
```

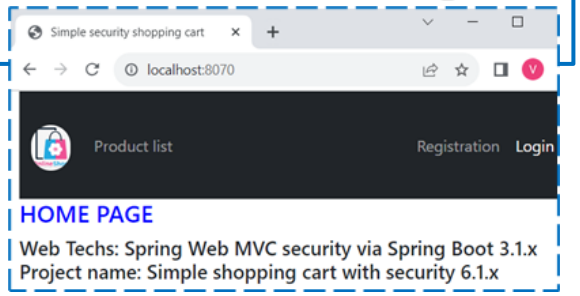
home.html ❷

```
1  <!DOCTYPE HTML>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head th:replace="~{/fragments/header :: head}"></head>
4
5  <body class="container-fluid">
6      <header>
7          <div th:replace="~{/fragments/header :: navbar}"></div>
8      <!-- <div th:replace="/fragments/header :: banner"></div>    -->
9      </header>
10     <main class = "backgrd">
11         <h3>HOME PAGE</h3>
12         <h5>Web Techs: <span th:text="${webTech}"></span><br />
13             Project name: <span th:text="${projectName}"></span>
14         </h5>
15     </main>
16     <div class="fixed-bottom start-50 translate-middle"
17          th:insert="~{/fragments/footer :: footer}"></div>
18 </body>
19 </html>
```

❶ + ❷

**Thymeleaf processing engine**

```
<!DOCTYPE html> == $0
<html>
 <head> … </head>
 <body class="container-fluid">
  <header> … </header>
  <main class="backgrd">
   <h3>HOME PAGE</h3>
   <h5>
    "Web Techs: "
    <span>Spring Web MVC security via Spring Boot 3.1.x</span>
    <br>
    " Project name: "
    <span>Simple shopping cart with security 6.1.x</span>
   </h5>
  </main>
  <footer class="fixed-bottom start-50 translate-middle"> … </footer>
 </body>
</html>
```



- **Demo:** Dự án Taco Cloud

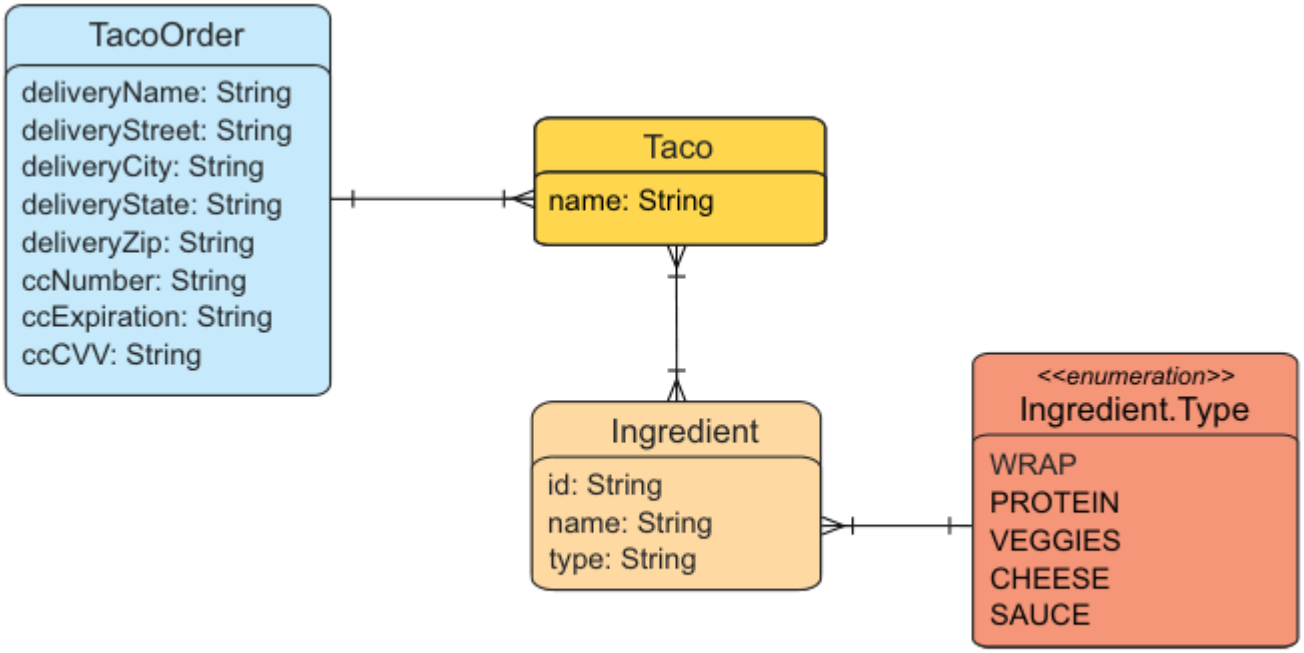
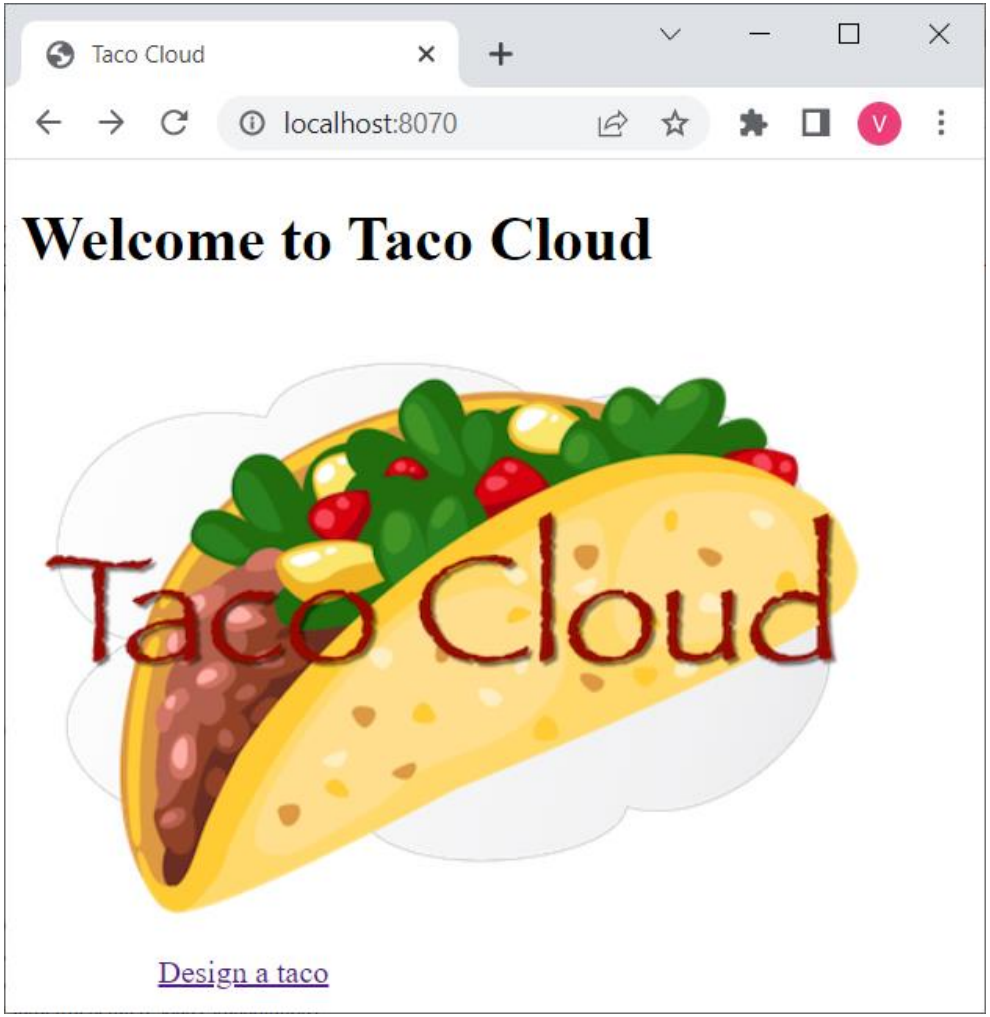



Figure 2.2 The Taco Cloud domain

# Project 1 ◆ Spring Boot MVC ◆ Chap 4

- spring-web-tacos-sd-jpa-tailieu2-chp3 [boot]
  - src/main/java
    - com.javaweb
      - SpringWebTacosSdJpaTailieu2Chp3Application.java
      - WebConfig.java
    - com.javaweb.controllers
      - DesignTacoController.java
      - IngredientByIdConverter.java
      - OrderController.java
    - com.javaweb.models
      - Ingredient.java
      - Taco.java
      - TacoOrder.java
    - com.javaweb.repositories
      - IngredientRepository.java
      - OrderRepository.java
  - src/main/resources
    - static
    - templates
      - design.html
      - home.html
      - orderForm.html
    - application.properties
  - src/test/java
  - JRE System Library [JavaSE-17]
  - Maven Dependencies
  - target/generated-sources/annotations
  - target/generated-test-sources/test-annotations
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

- **Project Tacos**



- Pom.xml

```
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtimek</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- **application.properties**

```
server.port=8070
spring.datasource.url=jdbc:mysql://localhost:3306/
taco-db?createDatabaseIfNotExist=true

spring.datasource.username=root
spring.datasource.password=Vdl#2022

# Hibernate
spring.datasource.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.jpa.database-
platform=org.hibernate.dialect.MySQL8Dialect


# Hibernate ddl auto (none, create, create-drop,
validate, update)
spring.jpa.hibernate.ddl-auto = create
```

Taco.java

```java
package com.javaweb.models;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Entity
public class Taco {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotNull
    @Size(min = 5, message = "Name must be at least 5 characters long")
    private String name;

    private Date createdAt = new Date();

    @Size(min = 1, message = "You must choose at least 1 ingredient")
    @ManyToMany()
    private List<Ingredient> ingredients = new ArrayList<>();

    public void addIngredient(Ingredient ingredient) {
        this.ingredients.add(ingredient);
    }
```

và setters/getters, contructors nếu cần

# Project 1 ◆ Spring Boot MVC ◆ Chap 4

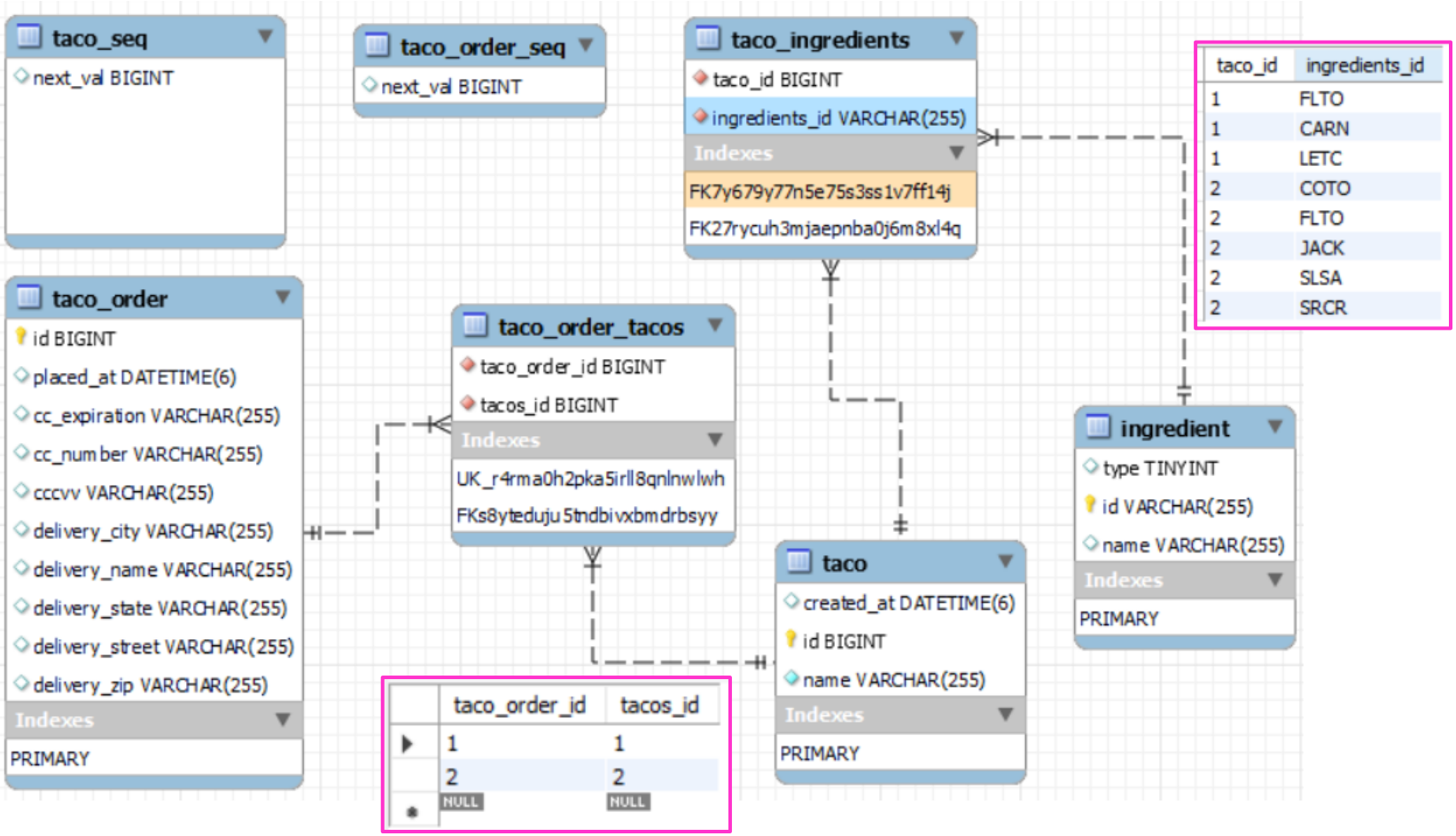
TacoOrder.java

```
19 @Entity
20 public class TacoOrder {
21     @Id @GeneratedValue(strategy = GenerationType.AUTO)
22     private Long id;
23
24     private Date placedAt = new Date();
25
26     @NotBlank(message = "Delivery name is required")
27     private String deliveryName;
28
29     @NotBlank(message = "Street is required")
30     private String deliveryStreet;
31
32     @NotBlank(message = "City is required")
33     private String deliveryCity;
34
35     @NotBlank(message = "State is required")
36     private String deliveryState;
37
38     @NotBlank(message = "Zip code is required")
39     private String deliveryZip;
40
41     @CreditCardNumber(message = "Not a valid credit card number")
42     private String ccNumber; // Ex: 5105105105105100
43
44     @Pattern(regexp = "^(0[1-9]|1[0-2])([\\/])([2-9][0-9])$",
45             message = "Must be formatted MM/YY")
46     private String ccExpiration; // 12/25
47
48     @Digits(integer = 3, fraction = 0, message = "Invalid CVV")
49     private String ccCVV; // 123
50
51     @OneToMany(cascade = CascadeType.ALL)
52     private List<Taco> tacos = new ArrayList<>();
```

Ingredient.java

```
1  package com.javaweb.models;
2
3  import jakarta.persistence.Entity;
5
6  @Entity
7  public class Ingredient {
8
9    @Id
10   private String id;
11   private String name;
12   private Type type;
13
14   public enum Type {
15     WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
16   }
17
18 public String getId() {
19     return id;
20 }
21
22 public void setId(String id) {
23     this.id = id;
24 }
25
```

- JPA/Hibernate sẽ tạo ra mô hình EER như sau




**One-direction Association**

**Followed by TLTK2**

WebConfig.java

```java
package com.javaweb;
import org.springframework.context.annotation.Configuration;

@Configuration
public class WebConfig implements WebMvcConfigurer {
  @Override
  public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/").setViewName("home");
  }
}
```

home.html

```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Taco Cloud</title>
     <link rel="stylesheet" href="styles.css">
  </head>

  <body>
    <h1>Welcome to Taco Cloud</h1>
    <img th:src="@{/images/TacoCloud.png}"/>
    <br/>

    <a th:href="@{/design}" id="design">Design a taco</a>
  </body>
</html>
```

GlobalController.java | DesignTacoController.java

```java
package com.javaweb.controllers;
import jakarta.validation.Valid;

@Controller
@RequestMapping("/design")
@SessionAttributes("tacoOrder")
public class DesignTacoController { //using GlobalController.java

    @ModelAttribute(name = "tacoOrder")
    public TacoOrder order() {
        return new TacoOrder();
    }

    @ModelAttribute(name = "taco")
    public Taco taco() {
        return new Taco();
    }

    @GetMapping
    public String showDesignForm() {
        return "design";        //design 1 bánh taco
    }

    @PostMapping
    public String processTaco(@Valid Taco taco, Errors errors, @ModelAttribute TacoOrder tacoOrder) {
        if (errors.hasErrors()) {
            return "design";
        }

        System.out.println("taco= " + taco.toString());
        tacoOrder.addTaco(taco);    //add 1 bánh taco vào model tacoOrder,
        //...rồi chuyển hướng sang order controller để đặt hàng chiếc taco này.
        return "redirect:/orders/current";
    }
}
```

**Note:** When hit the @RequestMapping("/**design**"), Spring MVC creates 3 model attributes: tacoOrder (session), taco (local), and a list of GLOBAL model attributes: wrap, protein, veggies, cheese, sauce.

- GlobalConroller.jav

GlobalController.java

```
3  import java.util.ArrayList;
15
16 /*
17  * @ModelAttribute annotated at Method Level in @ControllerAdvice Class
18  * @ModelAttribute can also be used at method level. With @ControllerAdvice annotated class,
19  * we can add values in Model which will be known as global.
20  * It means for every request a default value will be there in our response of each and every controller method.
21  * For the demo, we are creating a class annotated with @ControllerAdvice and a method annotated with @ModelAttribute.
22  */
23 @ControllerAdvice
24 public class GlobalController {
25     @Autowired
26     private IngredientRepository ingredientRepo;
27
28     @ModelAttribute
29     public void addIngredientsToModel(Model model) {
30         List<Ingredient> ingredients = new ArrayList<>();
31         ingredientRepo.findAll().forEach(i -> ingredients.add(i));
32
33         Type[] types = Ingredient.Type.values();
34         for (Type type : types) {   //list of GLOBAL model attributes
35             model.addAttribute(type.toString().toLowerCase(), filterByType(ingredients, type));
36         }
37     }
38     private Iterable<Ingredient> filterByType(List<Ingredient> ingredients, Type type) {
39         return ingredients.stream().filter(x -> x.getType().equals(type)).collect(Collectors.toList());
40     }
41 }
```

design.html

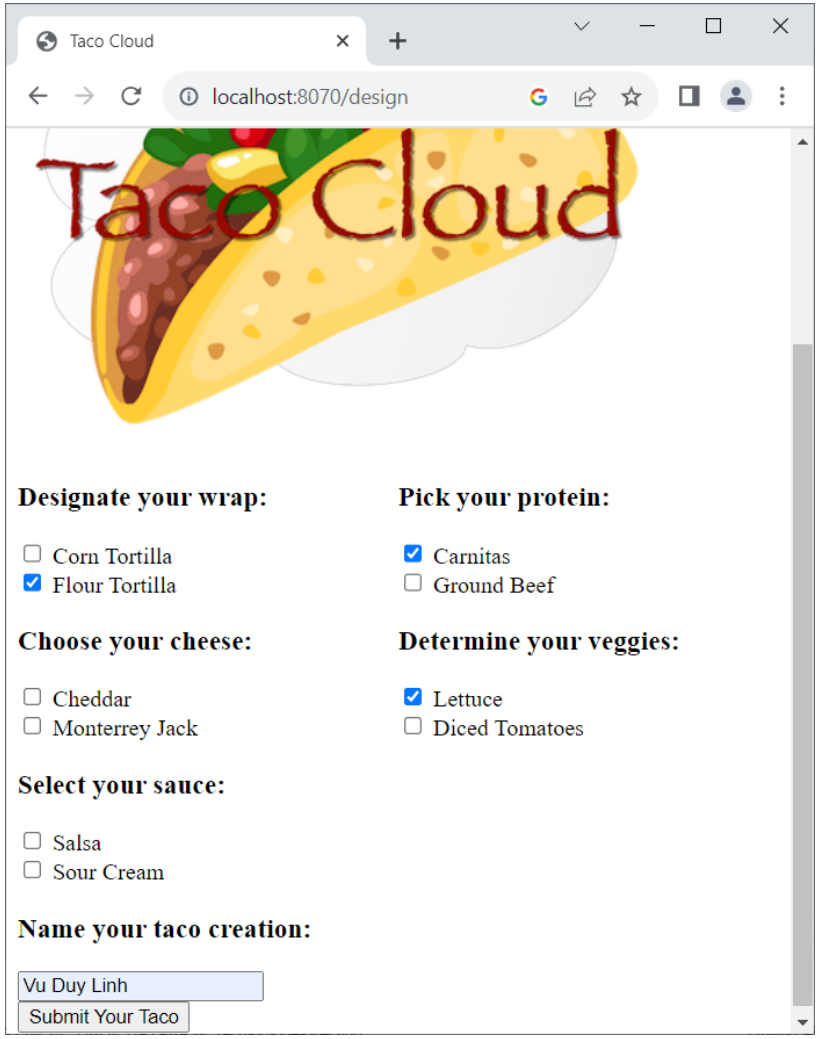
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Taco Cloud</title>
    <link rel="stylesheet" th:href="@{/styles.css}" />
  </head>

  <body>
    <form method="POST" th:object="${taco}">
    <!--when you click submit button it will redirect to
     the current page you are on if you don't include th:action.
     This is the same <form method="POST" th:action="@{/design}" th:object="${taco}">-->

      <!--  some elements are omitted-->
      <div class="ingredient-group" id="wraps">
        <h3>Designate your wrap:</h3>
        <div th:each="ingredient : ${wrap}">
          <input th:field="*{ingredients}"   <!-- taco.ingredients -->
                 type="checkbox" th:value="${ingredient.id}"/>
          <span th:text="${ingredient.name}">INGREDIENT</span><br/>
        </div>
      </div>
    <!--  some elements are omitted-->
      <h3>Name your taco creation:</h3>
      <input type="text" th:field="*{name}"/>   <!-- taco.name -->
      <span class="validationError"
            th:if="${#fields.hasErrors('name')}"
            th:errors="*{name}">Name Error</span>

      <button>Submit Your Taco</button>
    </form>
  </body>
</html>
```

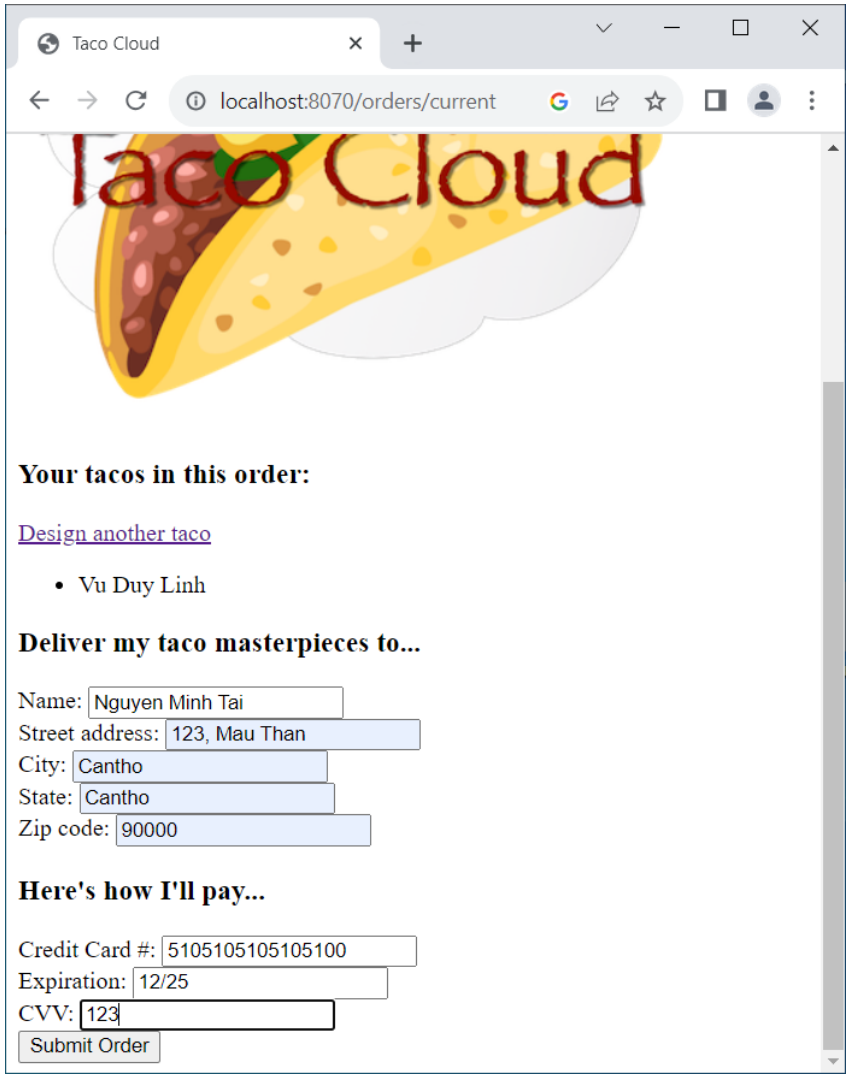
OrderController.java

```java
package com.javaweb.controllers;

import javax.validation.Valid;

@Controller
@RequestMapping("/orders")
@SessionAttributes("tacoOrder")
public class OrderController {
    @Autowired
    private OrderRepository orderRepo;

    @GetMapping("/current")
    public String orderForm() {
        return "orderForm";
    }

    @PostMapping
    public String processOrder(@Valid TacoOrder order,
            Errors errors, SessionStatus sessionStatus) {
        if (errors.hasErrors()) {
            return "orderForm";
        }

        orderRepo.save(order);
        sessionStatus.setComplete();

        return "redirect:/";
    }
}
```

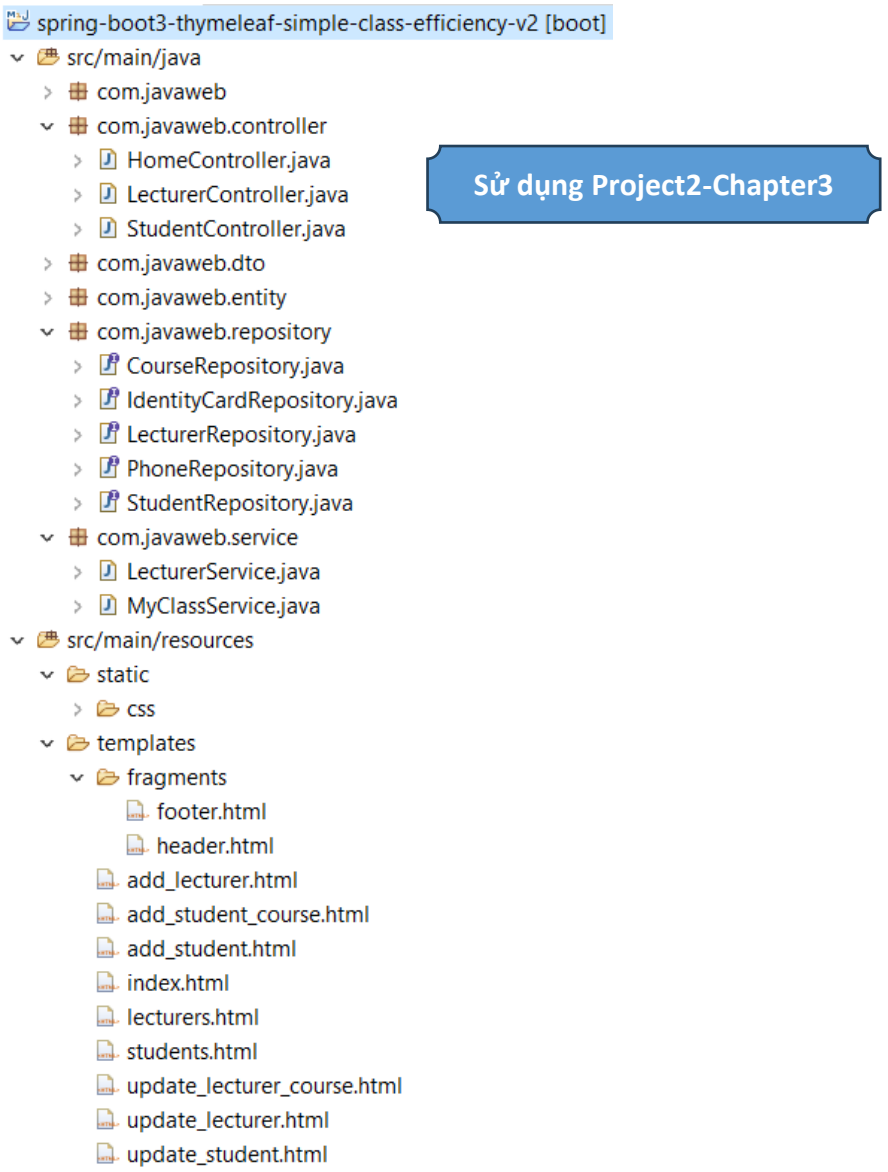
orderForm.html

```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Taco Cloud</title>
    <link rel="stylesheet" th:href="@{/styles.css}" />
  </head>
  <body>
    <form method="POST" th:action="@{/orders}" th:object="${tacoOrder}">
      <h1>Order your taco creations!</h1>

      <img th:src="@{/images/TacoCloud.png}"/>

      <h3>Your tacos in this order:</h3>
      <a th:href="@{/design}" id="another">Design another taco</a><br/>
      <ul>
        <li th:each="taco : ${tacoOrder.tacos}">
            <span th:text="${taco.name}">taco name</span>
        </li>
      </ul>
```

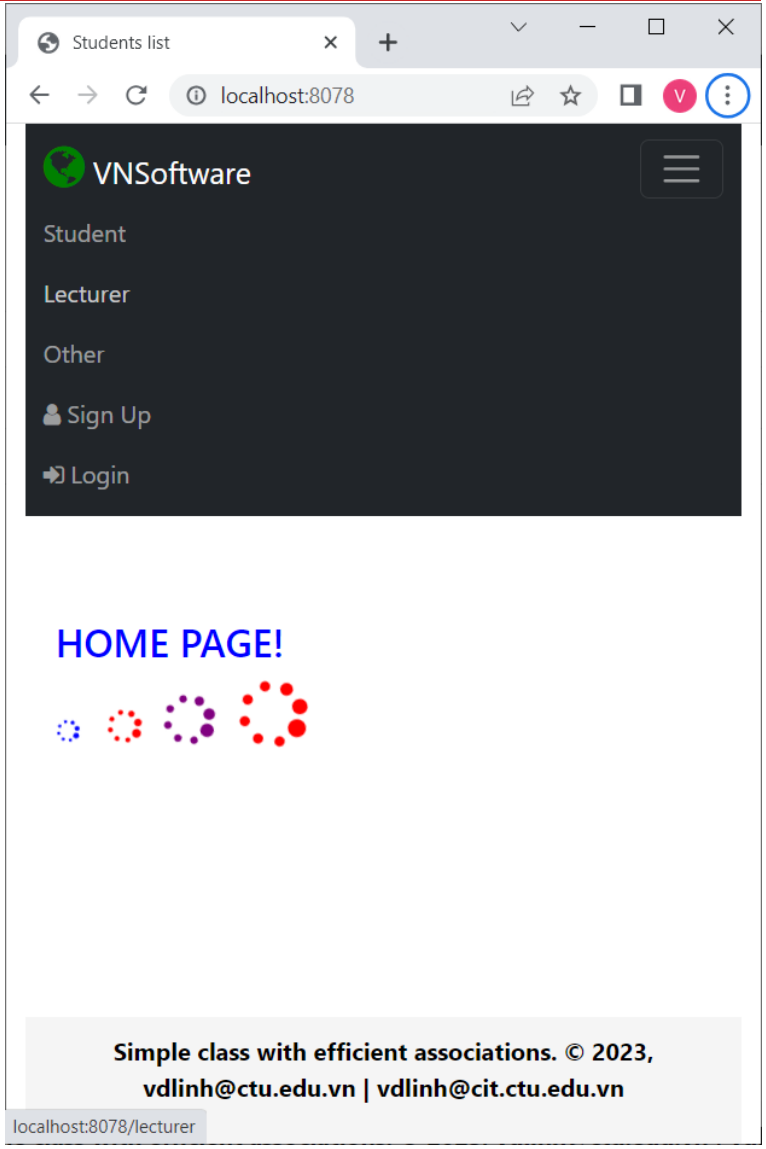

Taco Cloud
localhost:8070/design
Taco Cloud
Designate your wrap:
Corn Tortilla
Flour Tortilla
Pick your protein:
Carnitas
Ground Beef
Choose your cheese:
Cheddar
Monterrey Jack
Determine your veggies:
Lettuce
Diced Tomatoes
Select your sauce:
Salsa
Sour Cream
Name your taco creation:
Vu Duy Linh
Submit Your Taco


Taco Cloud
localhost:8070/orders/current
Taco Cloud
Your tacos in this order:
Design another taco
Vu Duy Linh
Deliver my taco masterpieces to...
Name: Nguyen Minh Tai
Street address: 123, Mau Than
City: Cantho
State: Cantho
Zip code: 90000
Here's how I'll pay...
Credit Card #: 5105105105105100
Expiration: 12/25
CVV: 123
Submit Order

# Project 2 ◆ Spring Boot MVC ◆ Chap 4




**Sử dụng Project2-Chapter3**



```java
package com.javaweb.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;
import com.javaweb.entity.IdentityCard;
public interface IdentityCardRepository extends JpaRepository<IdentityCard, Long> {
   @Transactional(readOnly = true)
   @Query("SELECT ic FROM IdentityCard ic WHERE ic.id = :id")
   IdentityCard fetchIdentityCardOfStudentById(Long id);
}
```

```java
package com.javaweb.repository;
//import …;
public interface LecturerRepository extends JpaRepository<Lecturer, Long> {
   @Transactional(readOnly = true)
   @Query("SELECT l FROM Lecturer l WHERE l.course.id = :id")
   List<Lecturer> fetchLecturersOfCourseById(Long id);

   @Transactional
   @Modifying
   @Query("UPDATE Lecturer l SET l.fullname= ?1, l.technologySpec= ?2 WHERE l.id= ?3")
   void updateLecturerById(String fullname, String technologySpec, Long lecturerId);

   @Transactional
   @Modifying
   @Query("UPDATE Lecturer l SET l.fullname= ?1, l.technologySpec= ?2, l.course= ?3
           WHERE l.id= ?4")
   void updateLecturerByIdWithCourse(String fullname, String technologySpec,
                                     Course course, Long lecturerId);
}
```

```java
package com.javaweb.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.javaweb.entity.Course;
public interface CourseRepository extends JpaRepository<Course, Long> {

}
```

```java
package com.javaweb.repository;
//import …;
public interface PhoneRepository extends JpaRepository<Phone, Long> {
   @Transactional(readOnly = true)
   @Query("SELECT p FROM Phone p WHERE p.student.id = :id")
   List<Phone> fetchPhonesOfStudentById(Long id);
}
```

```java
package com.javaweb.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;
import com.javaweb.entity.Student;
import java.util.List;

public interface StudentRepository extends JpaRepository<Student, Long> {
   List<Student> findByFirstnameContaining(String term);
   @Transactional
   @Modifying
   @Query("UPDATE Student s SET s.firstname = ?1, s.lastname = ?2, s.email = ?3
           WHERE s.id = ?4")
   void updateStudentById(String firstname, String lastname, String email, Long studentId);
}
```

StudentController.java

```
1  package com.javaweb.controller;
2
3  import org.slf4j.Logger;
24
25 @Controller
26 @RequestMapping(value = "/student")
27 public class StudentController {
28     private static final Logger log =
29             LoggerFactory.getLogger(StudentController.class);
30
31     @Autowired private CourseRepository courseRepository;
32     @Autowired private StudentRepository studentRepository;
33     @Autowired private PhoneRepository phoneRepository;
34     @Autowired private IdentityCardRepository identityCardRepository;
35
36     @GetMapping
37     public String getStudents(Model model) {
38         List<Student> students = studentRepository.findAll();
39         model.addAttribute("title", "Students list");
40         model.addAttribute("students", students);
41         log.info("getStudents()");
42
43         return "students";
44     }
45
```

```java
    @GetMapping("/search")
    public String search(@RequestParam("term") String term, Model model) {
        if (term.isEmpty()) {
            return "redirect:/student";
        }
        model.addAttribute("students", studentRepository.findByFirstnameContaining(term));
        return "students";
    }

    @GetMapping(value = "/addStudent")
    public String getAddStudent(Model model) {
        model.addAttribute("title", "Add Student");
        model.addAttribute("student", new Student());
        log.info("getAddStudent()");
        return "add_student";
    }

    @GetMapping(value = "/save")
    public String saveStudent(Student student) {
        studentRepository.save(student);
        log.info("saveStudent()");
        return "redirect:/student";
    }

    @GetMapping(value = "/update/{studentId}/course")
    public String getStudentIdWithCourse(@PathVariable("studentId") Long studentId, Model model) {
        model.addAttribute("title", "Update course");
        List<Course> availableCourses = courseRepository.findAll();
        availableCourses.remove(0);
        model.addAttribute("allCourses", availableCourses);
```

```
76          Student findStudent = studentRepository.getReferenceById(studentId);
77          model.addAttribute("student", findStudent);
78          return "add_student_course";
79      }
80
81      @GetMapping(value = "/update/{studentId}")
82      public String updateStudentId(Model model, @PathVariable("studentId") Long studentId) {
83          Student updateStudent = studentRepository.getReferenceById(studentId);
84
85          model.addAttribute("title", "Update Student");
86          model.addAttribute("updateStudent", updateStudent);
87          Log.info("getAddStudent()");
88          return "update_student";
89      }
90
91      @PostMapping(value = "/update/{studentId}")
92      public String updateStudent(@ModelAttribute("updateStudent") Student updateStudent,
93              @PathVariable("studentId") Long studentId) {
94          studentRepository.updateStudentById(updateStudent.getFirstname(), updateStudent.getLastname(),
95                  updateStudent.getEmail(), studentId);
96  //        System.out.println("====================UPDATED==============");
97          return "redirect:/student";
98      }
99
100     @GetMapping(value = "/delete/{studentId}")
101     public String deleteStudentIdWithAllAssociations(@PathVariable("studentId") Long studentId) {
102         List<Phone> studentPhones = phoneRepository.fetchPhonesOfStudentById(studentId);
103         studentPhones.forEach(p -> phoneRepository.delete(p));
104
105         identityCardRepository.deleteById(studentId); // studentId === cardID
106 //      identityCardRepository.delete(identityCardRepository.fetchIdentityCardOfStudentById(studentId));
```

```
107
108         studentRepository.deleteById(studentId);
109         System.out.println("===========DONE: studentRepository.deleteById(studentId)============");
110
111         Log.info("detelteStudentId() : " + studentId);
112         return "redirect:/student";
113     }
114
115     @GetMapping(value = "/delete/{studentId}/{courseId}")
116     public String deleteCourseWithStudentId(@PathVariable("studentId") Long studentId,
117             @PathVariable("courseId") Long courseId) {
118         Student student = studentRepository.findById(studentId).orElse(null);
119         student.getCourses().remove(courseRepository.findById(courseId).orElse(null));
120         studentRepository.save(student);
121
122         System.out.println("====== DELETE courseID for StudentID======");
123         return "redirect:/student/update/{studentId}/course";
124     }
125
126     @GetMapping(value = "/addCourseStudent/{studentId}/course")
127     public String addCourseStudent(@PathVariable("studentId") Long studentId,
128             @RequestParam("courseId") Long courseId) {
129
130         Course course = courseRepository.getReferenceById(courseId);
131         Student student = studentRepository.getReferenceById(studentId);
132
133         if (student != null) {
134             student.getCourses().add(course);
135             studentRepository.save(student);
136         }
137
138         return "redirect:/student/update/{studentId}/course";
139     }
140 }
```

← header.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
  <head th:fragment="head">
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title th:utext="${title}"></title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" th:href="@{/css/main.css}" href="../../css/main.css" />
    <script src=
"https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
  </head>
<body>
  <nav th:fragment="navbar" class="navbar navbar-expand-sm bg-dark navbar-dark">
    <div class="container-fluid">
      <a class="navbar-brand" th:href="@{/}"><i class="fa fa-globe" style="font-size:32px;
color: green“ aria-hidden="true"></i> VNSoftware</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#collapsibleNavbar"><span class="navbar-toggler-icon"/></button>
      <div class="collapse navbar-collapse" id="collapsibleNavbar">
        <ul class="navbar-nav">
          <li class="nav-item"><a class="nav-link" th:href="@{/student}">Student</a></li>
          <li class="nav-item"><a class="nav-link" th:href="@{/lecturer}">Lecturer</a></li>
          <li class="nav-item"><a class="nav-link" href="#">Other</a></li></ul>
        <ul class="navbar-nav ms-auto">
          <li><a class="nav-link" href="#"> <i class="fa fa-user"></i> Sign Up</a></li>
          <li><a class="nav-link" href="#"><i class="fa fa-sign-in"></i> Login</a></li>
        </ul></div></div>
  </nav>
</body></html>
```

footer.html

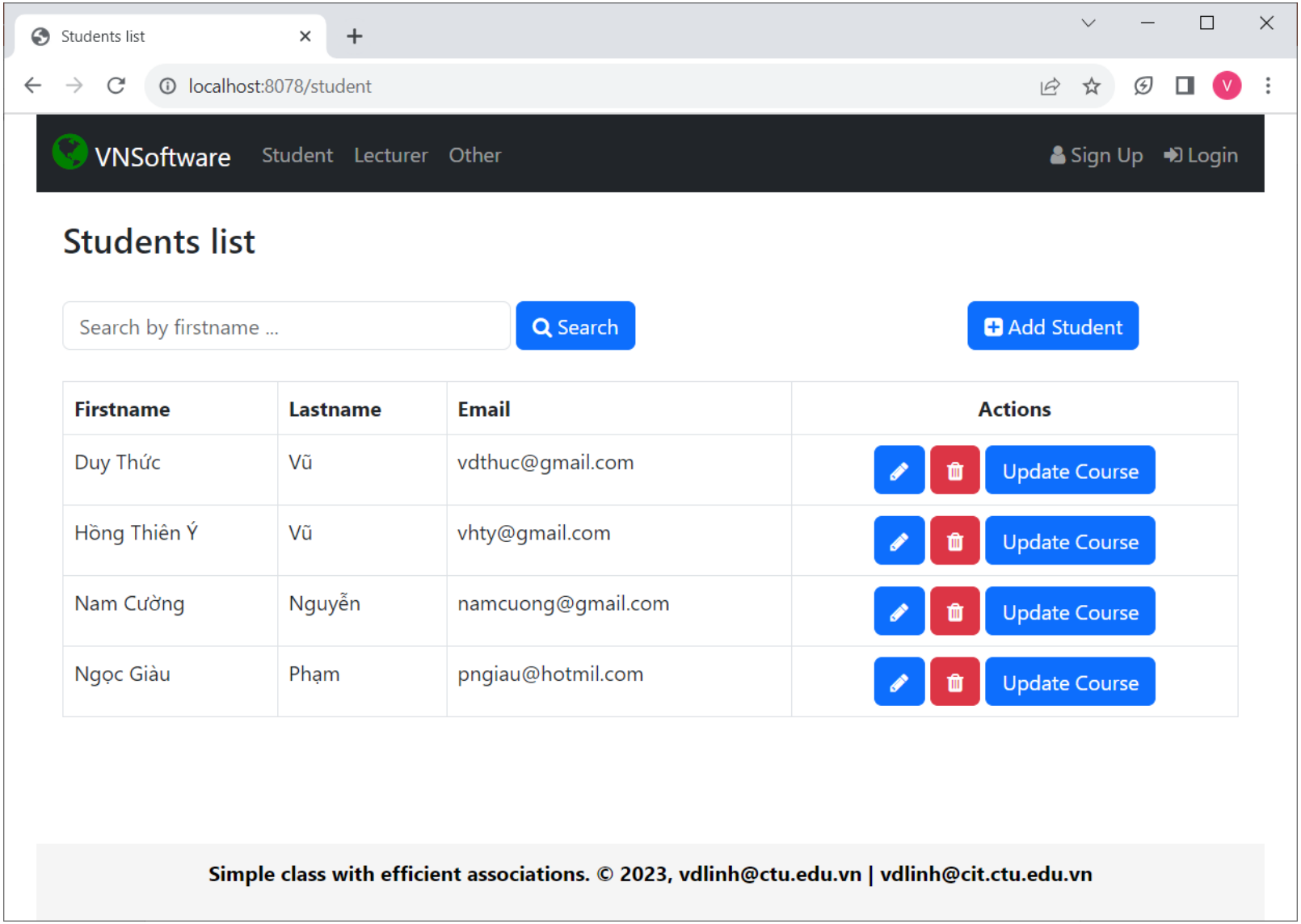
```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">

<body>
    <footer th:fragment="footer" class="sticky-footer">
        <p class="text-center">Simple class with efficient associations.
            &copy; 2023, vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn</p>
    </footer>
</body>
</html>
```

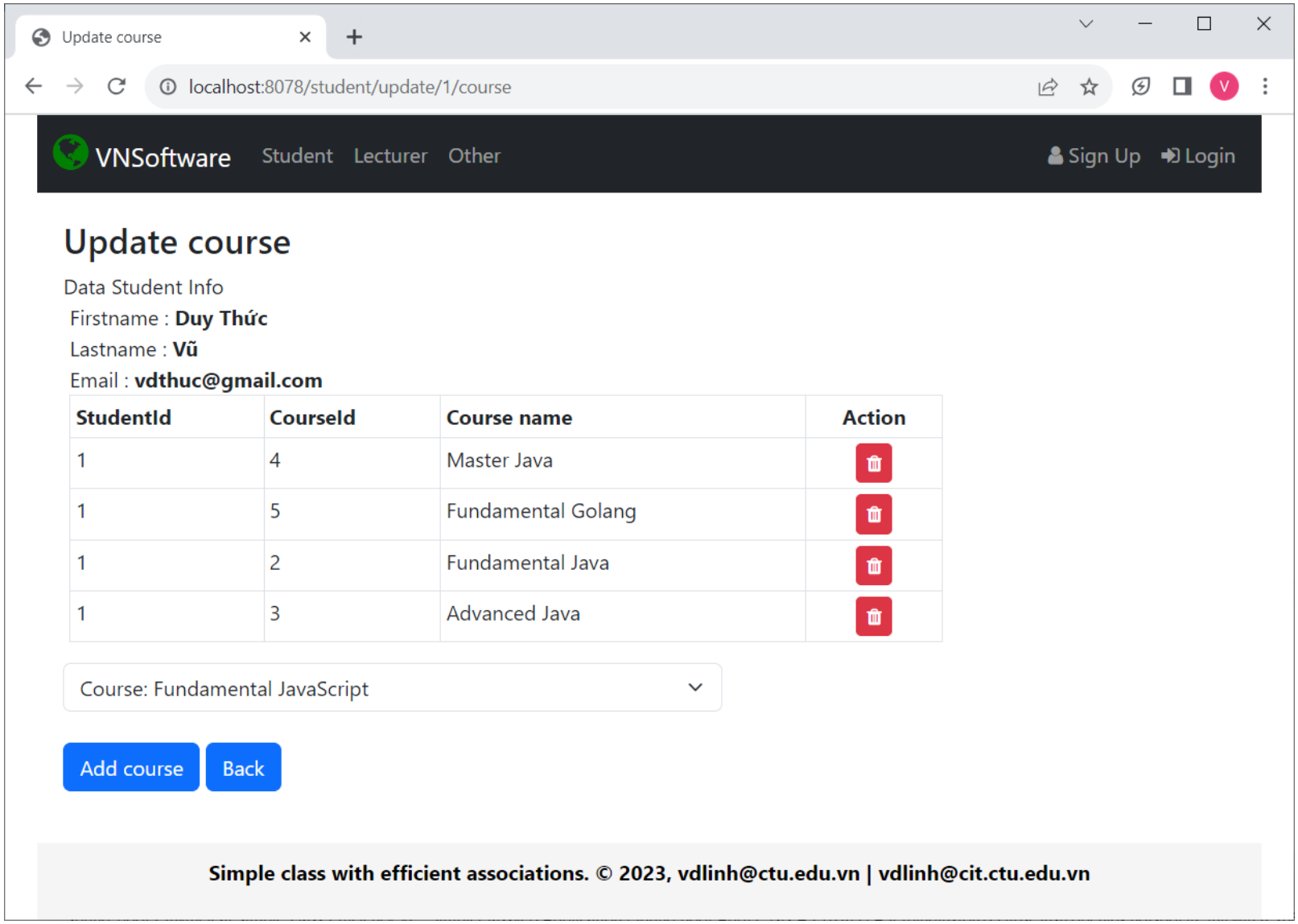
← footer.html

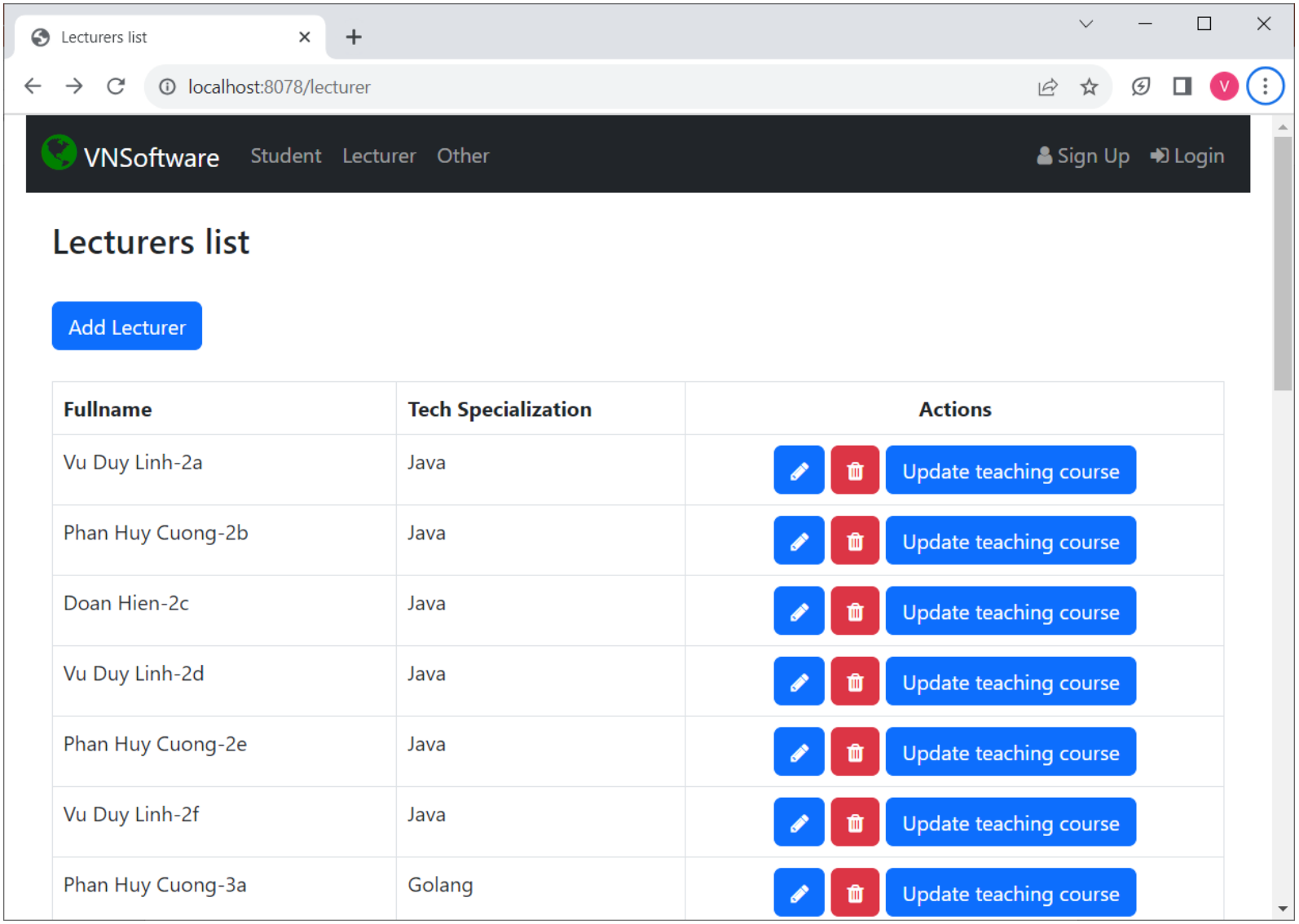
index.html

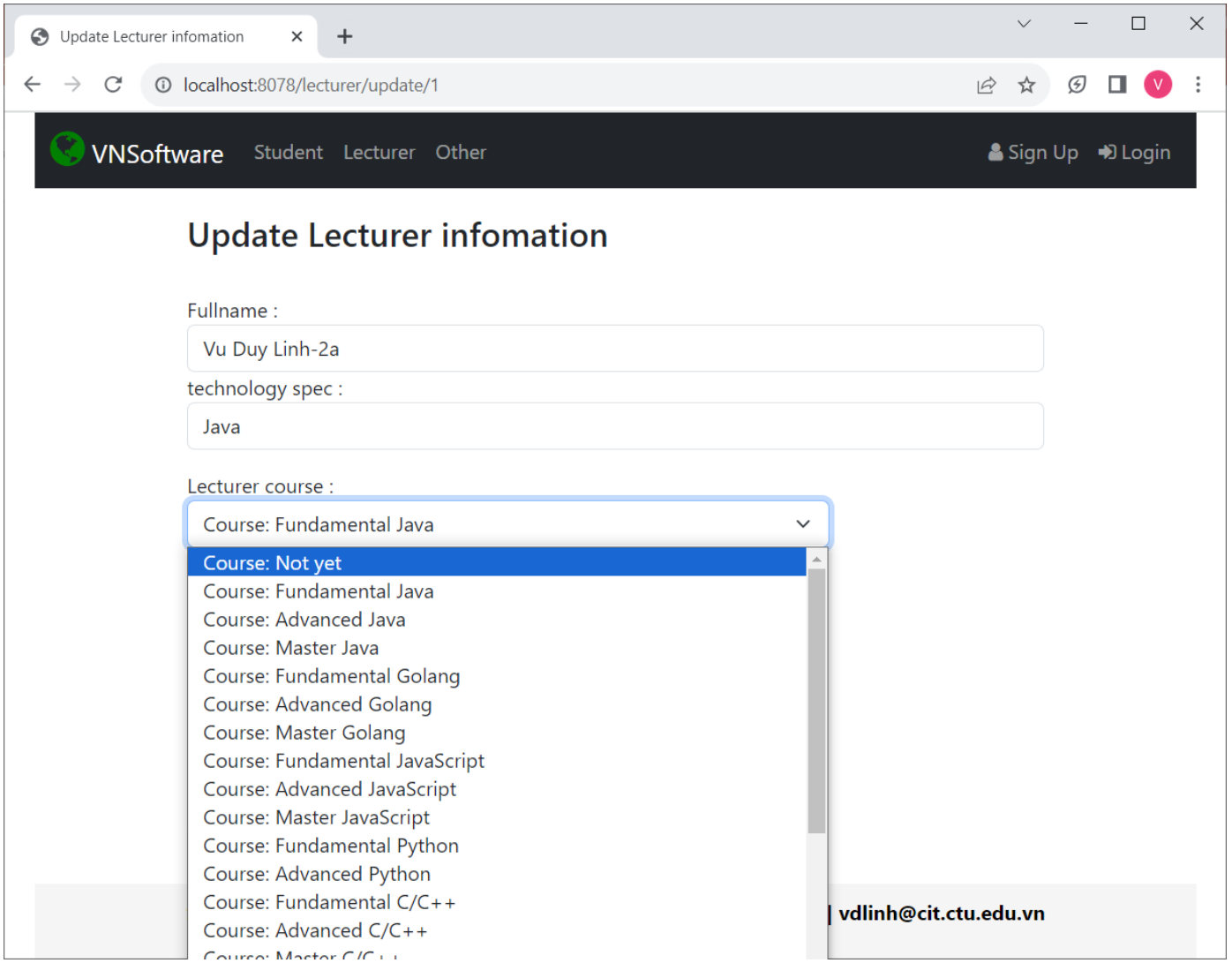
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head th:replace="~{/fragments/header :: head}" />

<body class="page">
    <header th:insert="~{/fragments/header :: navbar}" />
    <main class="mt-5 content">
        <h2 style="color: blue">HOME PAGE!</h2>
        <i class="fa fa-spinner fa-spin" style="font-size:16px; color: blue"></i>
          
        <i class="fa fa-spinner fa-spin" style="font-size:24px; color: red"></i>
          
        <i class="fa fa-spinner fa-spin" style="font-size:36px; color: purple"></i>
          
        <i class="fa fa-spinner fa-spin" style="font-size:48px;color: red"></i>
    </main>
    <div th:replace="~{/fragments/footer :: footer}" />
</body>
</html>
```

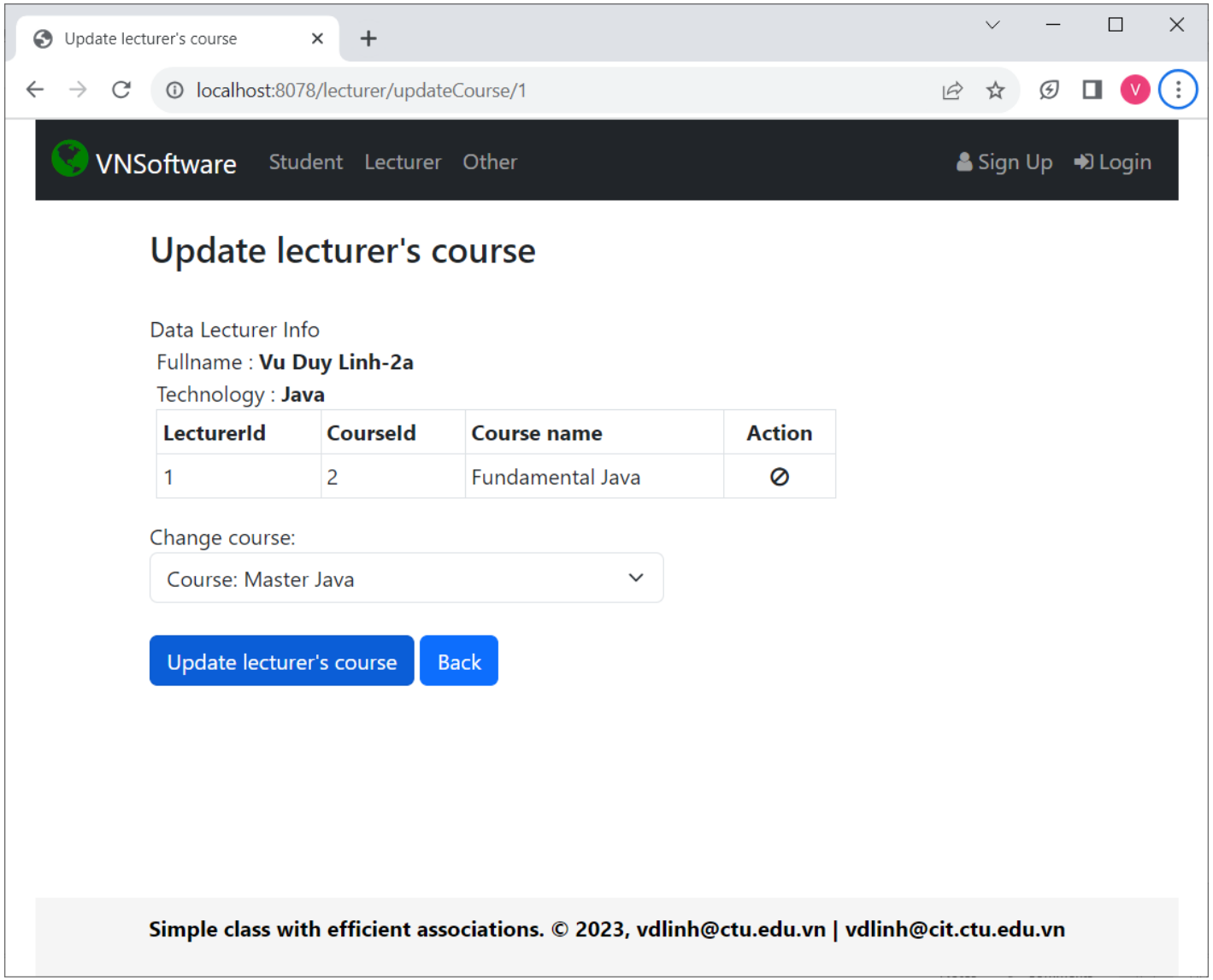
← index.html


Students list
localhost:8078/student
VNSoftware
Student
Lecturer
Other
Sign Up
Login
Students list
Search by firstname ...
Search
Add Student
Firstname
Lastname
Email
Actions
Duy Thức
Vũ
vdthuc@gmail.com
Update Course
Hồng Thiên Ý
Vũ
vhty@gmail.com
Update Course
Nam Cường
Nguyễn
namcuong@gmail.com
Update Course
Ngọc Giàu
Phạm
pngiau@hotmil.com
Update Course
Simple class with efficient associations. © 2023, vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

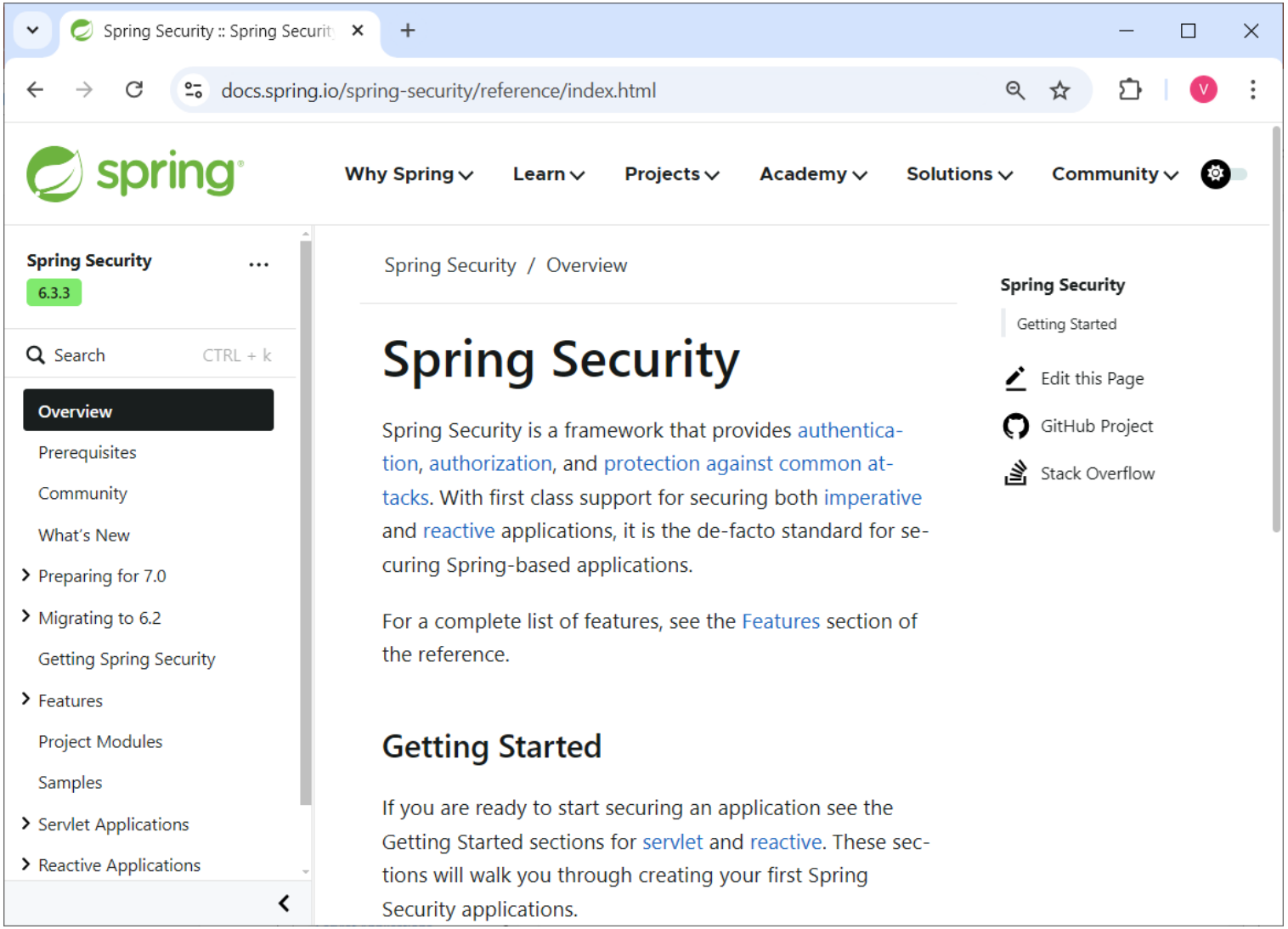

Update course
localhost:8078/student/update/1/course
VNSoftware
Student
Lecturer
Other
Sign Up
Login
Update course
Data Student Info
Firstname : Duy Thức
Lastname : Vũ
Email : vdthuc@gmail.com
StudentId
CourseId
Course name
Action
1
4
Master Java
1
5
Fundamental Golang
1
2
Fundamental Java
1
3
Advanced Java
Course: Fundamental JavaScript
Add course
Back
Simple class with efficient associations. © 2023, vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn


Lecturers list
localhost:8078/lecturer
VNSoftware
Student
Lecturer
Other
Sign Up
Login
Lecturers list
Add Lecturer
Fullname
Tech Specialization
Actions
Vu Duy Linh-2a
Java
Update teaching course
Phan Huy Cuong-2b
Java
Update teaching course
Doan Hien-2c
Java
Update teaching course
Vu Duy Linh-2d
Java
Update teaching course
Phan Huy Cuong-2e
Java
Update teaching course
Vu Duy Linh-2f
Java
Update teaching course
Phan Huy Cuong-3a
Golang
Update teaching course


Update Lecturer infomation
localhost:8078/lecturer/update/1
VNSoftware
Student
Lecturer
Other
Sign Up
Login
Update Lecturer infomation
Fullname :
Vu Duy Linh-2a
technology spec :
Java
Lecturer course :
Course: Fundamental Java
Course: Not yet
Course: Fundamental Java
Course: Advanced Java
Course: Master Java
Course: Fundamental Golang
Course: Advanced Golang
Course: Master Golang
Course: Fundamental JavaScript
Course: Advanced JavaScript
Course: Master JavaScript
Course: Fundamental Python
Course: Advanced Python
Course: Fundamental C/C++
Course: Advanced C/C++
vdlinh@cit.ctu.edu.vn


Update lecturer's course
localhost:8078/lecturer/updateCourse/1
VNSoftware
Student
Lecturer
Other
Sign Up
Login
Update lecturer's course
Data Lecturer Info
Fullname : Vu Duy Linh-2a
Technology : Java
LecturerId
CourseId
Course name
Action
1
2
Fundamental Java
Change course:
Course: Master Java
Update lecturer's course
Back
Simple class with efficient associations. © 2023, vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

- **Official reference:** https://docs.spring.io/spring-security/reference/index.html



- **Reactive vs Servlet Stack**
  - https://germanoschneider.medium.com/reactive-vs-servlet-stack-d4f87ec35e39

- **Spring Security (Servlet-stack web applications)**
  - Là một khung ứng dụng (framework), được xây dựng trên Spring framework, tập trung vào việc cung cấp các cơ chế bảo mật cho các ứng dụng (Web app, RESTful/SOAP Web services, Microservices) ở tầm doanh nghiệp từ việc **xác thực (authentication)**, **phân quyền (authorization)** đến các cơ chế bảo vệ khác như CSRF, XSS, Man-in-the-Middle (MITM) attacks,… Giống như tất cả các dự án Spring, sức mạnh thực sự của Spring Security nằm ở việc nó có thể được mở rộng dễ dàng và tùy biến cao.

  - Về cơ bản, bảo mật Spring hoạt động dựa trên một khái niệm gọi là Dịch vụ ủy quyền và xác thực Java, **JAAS** (Java Authentication and Authorization Services), được hoạt động dựa trên các Bộ lọc chuẩn (standard Servlet Filter: **jakarta.servlet.Filter**). Khi xây dựng ứng dụng bằng Spring Boot có triển khai các tính năng bảo mật, lúc người dùng gửi yêu cầu (request) đến ứng dụng, nó sẽ đi qua bộ lọc bảo mật trước rồi mới đến **DispatcherServlet**. Do vậy, chỉ những người dùng đã xác thực tính hợp lệ bảo mật của bộ lọc này thì yêu cầu mới được chuyển đến DispatcherServlet để phục vụ mục đích của người dùng.

  - Cũng như **Spring Boot MVC**, phần Spring Security cũng chỉ được trình bày phần bảo mật cho dạng **Servlet applications** [được gọi chung là **Spring Boot Security MVC**].

- **Servlet Security Architecture (Kiến trúc bảo mật Servlet)**
  - Ứng dụng Servlet của Spring Security dựa trên các bộ lọc Servlet (Servlet Filters) được tích hợp sẵn trong bộ chứa Servlet (Servlet Container). Các bộ lọc này nằm phía trước các servlet (hay Controllers/RestControllers) và chặn (intercept) tất cả các yêu cầu (requests) để thực hiện việc kiểm tra bảo mật **trước khi** chúng đến được các servlets như **DispatcherServlet** để thực hiện các nghiệp vụ, hoặc gọi Rest APIs để truy cập các tài nguyên (resources) hợp lệ.
  - Khi client gửi yêu cầu đến ứng dụng, Servlet Container tạo **FilterChain** chứa các thể hiện Filter, và Java Servlet sẽ xử lý HttpServletRequest dựa trên đường dẫn của request URI.
  - Bộ lọc là một thành phần rất hữu ích trong đặc tả Servlet. Spring Security sử dụng nó để triển khai một số chức năng cốt lõi và chiến lược xác thực.
  - Do mỗi thể hiện Filter là một thành phần bộ chứa servlet nên vòng đời của nó được quản lý bởi bộ chứa servlet, và theo Servlet specification không yêu cầu bất kỳ loại tích hợp Spring nào để giao tiếp và xử lý Filter. Do vậy Spring Security cung cấp một bộ lọc có tên **DelegatingFilterProxy** để thu hẹp khoảng cách này, cho phép kết nối giữa vòng đời của bộ chứa Servlet (Servlet Container) và ApplicationContext của Spring (do Spring Container quản lý).

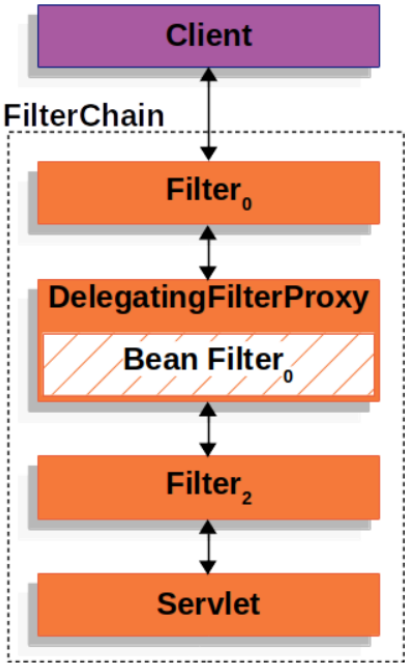
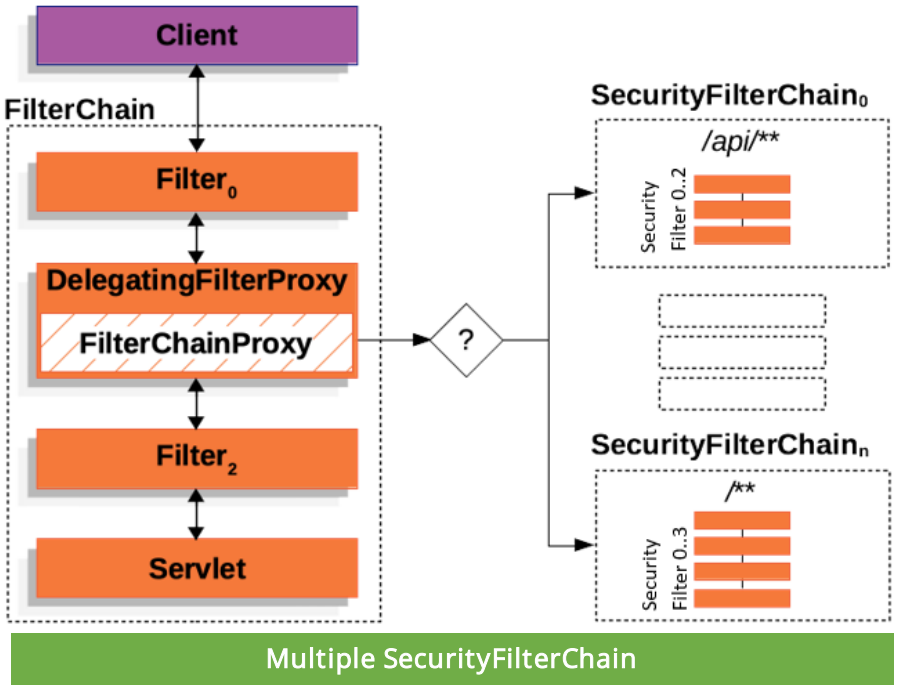



*Figure 2. DelegatingFilterProxy*

**<*> Link:** https://docs.spring.io/spring-security/reference/servlet/architecture.html

- **Servlet Security Architecture**
  - **Filter chain**: The client sends a request to the application, and the container creates a FilterChain to process the HttpServletRequest, based on the path of the request URI.
  - **DelegatingFilterProxy** filter: allows bridging between the Servlet container's lifecycle and Spring's ApplicationContext.
  - DelegatingFilterProxy: là một bộ lọc đặc biệt được sử dụng để kích hoạt Spring Security. Bộ lọc này (trong Spring Container) được đăng ký vào thùng chứa servlet (để vòng đời của nó được quản lý bởi bộ chứa servlet) và nó bắt đầu chặn các yêu cầu đến. Trong ứng dụng Spring Boot, việc đăng ký này được cấu hình tự động cho Spring Security.




- Ở thời điểm runtime, **DelegatingFilterProxy** tìm ra bộ lọc Bean **Filter$_0$** từ **ApplicationContext** do Spring Container quản lý, và sau đó gọi Bean **Filter$_0$** để ủy quyền yêu cầu xử lý (lọc bảo mật).

- **FilterChainProxy**:
  - ✓ is a special Filter provided by Spring Security that allows delegating to many Filter instances through SecurityFilterChain. Since FilterChainProxy is a Bean, it is typically wrapped in a **DelegatingFilterProxy**.
  - ✓ provides a starting point for all of Spring Security's Servlet support (know debug point) and is central to Spring Security usage to perform tasks flexibly in determining when a **SecurityFilterChain** should be invoked.

- **SecurityFilterChain**: is used by FilterChainProxy to determine which Spring Security Filter instances should be invoked for the current request (Only the first SecurityFilterChain that matches is invoked).

- **SecurityFilterChain** (Chuỗi bộ lọc) được FilterChainProxy sử dụng để xác định thể hiện Spring Security Filter nào sẽ được gọi cho yêu cầu hiện tại **(Only the first SecurityFilterChain that matches is invoked)**. SecurityFilterChain có hai phương thức: matches(...) và getFilters(...). Phương thức đầu tiên cho phép Spring Security đánh giá xem SecurityFilterChain hiện tại có khớp với yêu cầu đến hay không. Spring Security cung cấp giao diện requestMatcher và cung cấp một số cách triển khai để thực hiện đối sánh/so khớp. Spring Security cũng cung cấp một trình so khớp kiểu AntPathRequestMatcher phù hợp với các đường dẫn URL.

- **Security Filters** (in **SecurityFilterChain**) are typically Beans. It uses **HttpSecurity** object to configure how HTTP requests are secured. Các Bộ lọc bảo mật (Security Filters) trong SecurityFilterChain thường là các Bean, được đăng ký với **FilterChainProxy** thay vì DelegatingFilterProxy. Những bộ lọc này được sử dụng cho một số mục đích khác nhau, như xác thực (authentication), phân quyền (authorization), bảo vệ chống khai thác (exploit protection),…

- Trong phiên bản hiện hành, Spring Security 6.x, khuyến nghị thực hiện việc định cấu hình HttpSecurity bằng cách đăng ký một bean SecurityFilterChain như sau:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
   @Bean
   SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
        .csrf(Customizer.withDefaults())

        .authorizeHttpRequests(authorize -> authorize
        .requestMatchers("/", "/login", "/403", "/css/**", "/images/**")
        .permitAll()

        .requestMatchers("/registration","/webjars/**").permitAll()

        .anyRequest().authenticated())

        .httpBasic(Customizer.withDefaults())

        .formLogin(Customizer.withDefaults());   //default login form

     return http.build();

  }
}
```

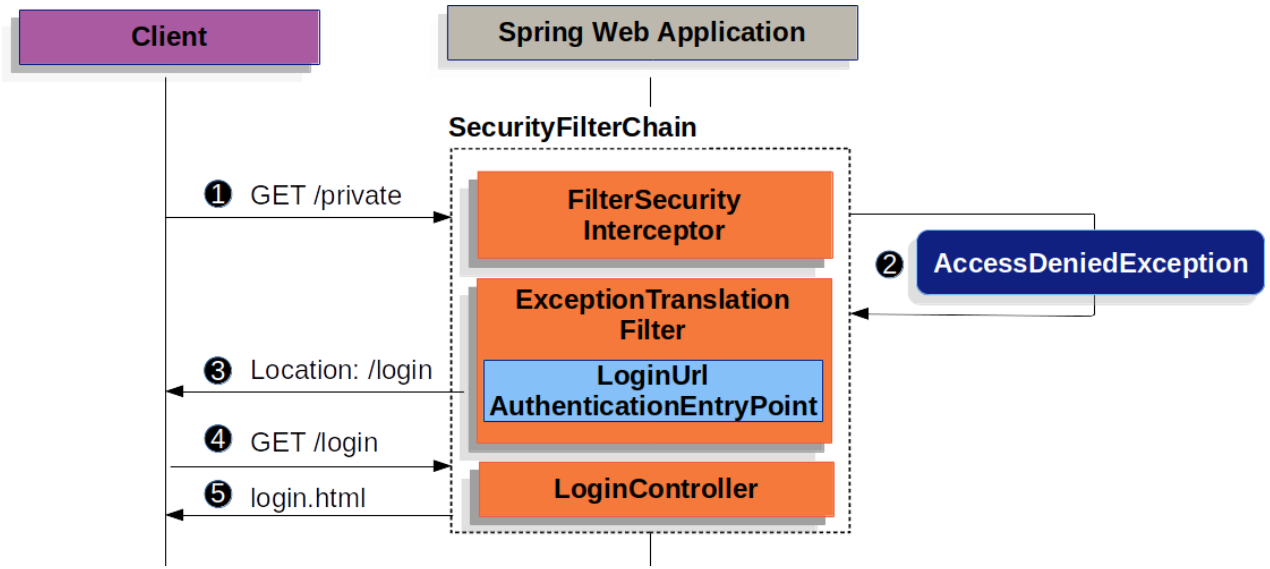
- **Servlet Authentication Architecture (Xác thực)**

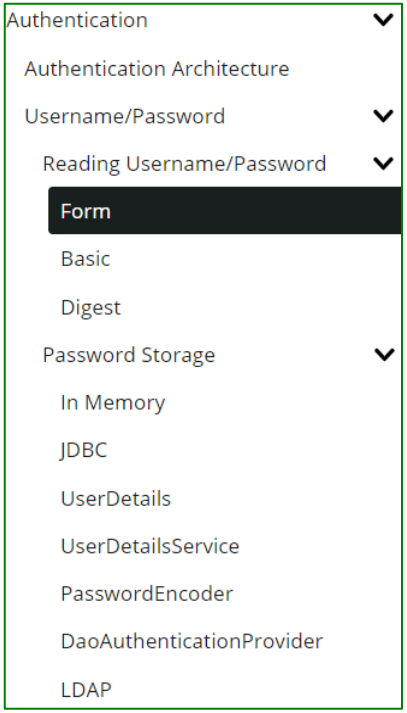

REF: https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html

- **The SecurityContextHolder** is where Spring Security stores the details of who is authenticated.
- **SecurityContext** is obtained from the SecurityContextHolder and contains the Authentication of the currently authenticated user.
- **Authentication** can be the input to AuthenticationManager to provide the credentials a user has provided to authenticate or the current user from the SecurityContext.
- The **Authentication** serves two main purposes within Spring Security:
  - An input to AuthenticationManager to provide the **credentials** a user has provided to authenticate. When used in this scenario, isAuthenticated() returns false.
  - Represents the currently authenticated user. The current Authentication can be obtained from the SecurityContext.
- The **Authentication** contains
  - **principal** - identifies the user. When authenticating with a username/password this is often an instance of UserDetails.
  - **credentials** - often a password. In many cases this will be cleared after the user is authenticated to ensure it is not leaked.
  - **Authorities (quyền hạn trong xử lý phân quyền/ Authorization)** - the GrantedAuthoritys are high level permissions the user is granted. A few examples are roles or scopes.

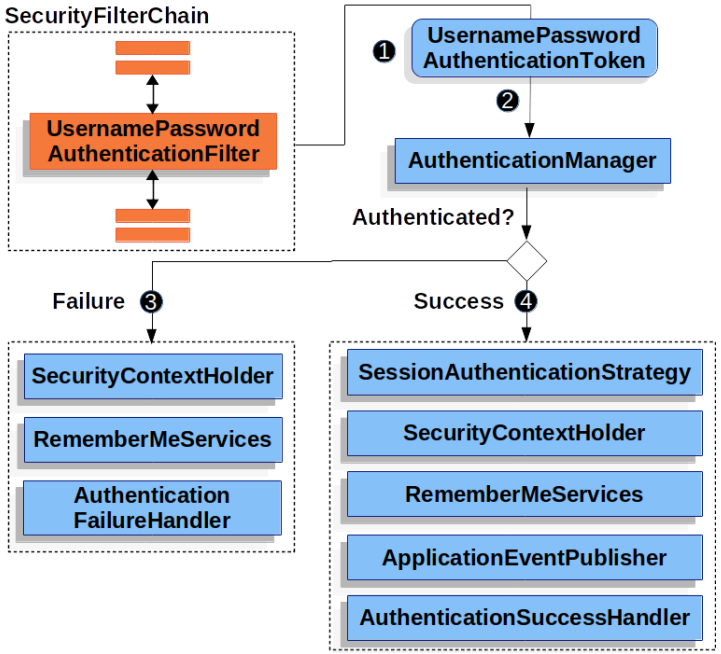
- **Form-based login works within Spring Security**

Ref: https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/form.html




1. First, a user makes an unauthenticated request to the resource ( `/private` ) for which it is not authorized.
2. Spring Security's `AuthorizationFilter` indicates that the unauthenticated request is *Denied* by throwing an `AccessDeniedException`.
3. Since the user is not authenticated, `ExceptionTranslationFilter` initiates *Start Authentication* and sends a redirect to the login page with the configured `AuthenticationEntryPoint`. In most cases, the `AuthenticationEntryPoint` is an instance of `LoginUrlAuthenticationEntryPoint`.
4. The browser requests the login page to which it was redirected.
5. Something within the application, must render the login page.




1 When the user submits their username and password, the `UsernamePasswordAuthenticationFilter` creates a `UsernamePasswordAuthenticationToken`, which is a type of `Authentication`, by extracting the username and password from the `HttpServletRequest` instance.

2 Next, the `UsernamePasswordAuthenticationToken` is passed into the `AuthenticationManager` instance to be authenticated. The details of what `AuthenticationManager` looks like depend on how the user information is stored.

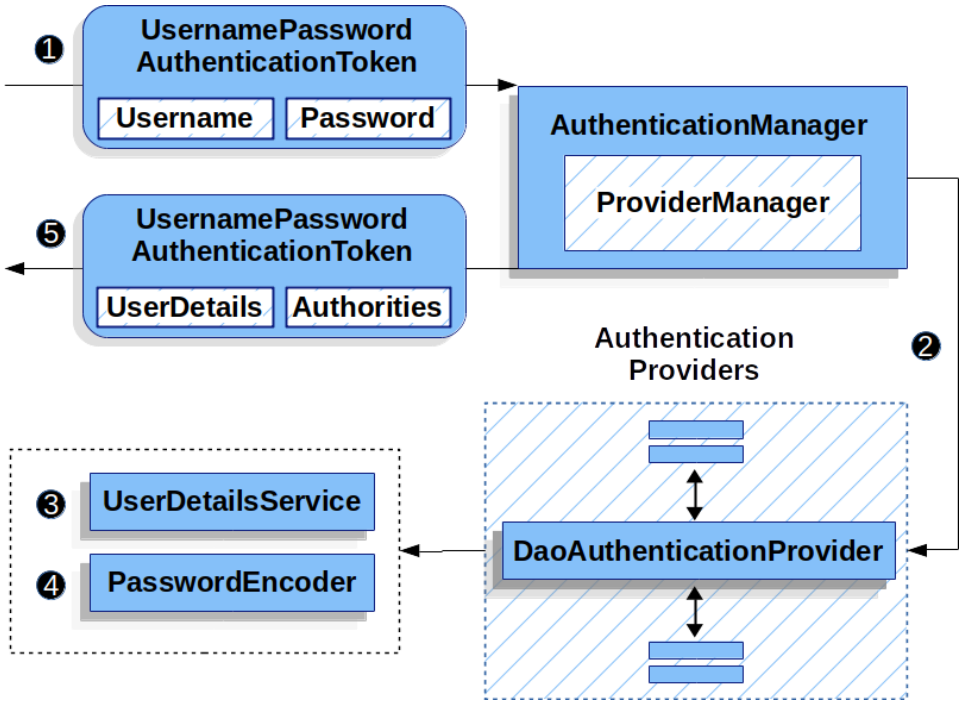
3 If authentication fails, then *Failure*.

1. The SecurityContextHolder is cleared out.
2. `RememberMeServices.loginFail` is invoked. If remember me is not configured, this is a no-op. See the `RememberMeServices` interface in the Javadoc.
3. `AuthenticationFailureHandler` is invoked. See the `AuthenticationFailureHandler` class in the Javadoc

4 If authentication is successful, then *Success*.

1. `SessionAuthenticationStrategy` is notified of a new login. See the `SessionAuthenticationStrategy` interface in the Javadoc.
2. The Authentication is set on the SecurityContextHolder. See the `SecurityContextPersistenceFilter` class in the Javadoc.
3. `RememberMeServices.loginSuccess` is invoked. If remember me is not configured, this is a no-op. See the `RememberMeServices` interface in the Javadoc.
4. `ApplicationEventPublisher` publishes an `InteractiveAuthenticationSuccessEvent`.
5. The `AuthenticationSuccessHandler` is invoked. Typically, this is a `SimpleUrlAuthenticationSuccessHandler`, which redirects to a request saved by `ExceptionTranslationFilter` when we redirect to the login page.

- **DaoAuthenticationProvider (in theory)** is an AuthenticationProvider implementation **that uses** a **UserDetailsService** and **PasswordEncoder** to authenticate a username and password.




REF: https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/dao-authentication-provider.html

(1) The authentication Filter from Reading the Username & Password passes a UsernamePasswordAuthenticationToken to the AuthenticationManager which is implemented by ProviderManager.

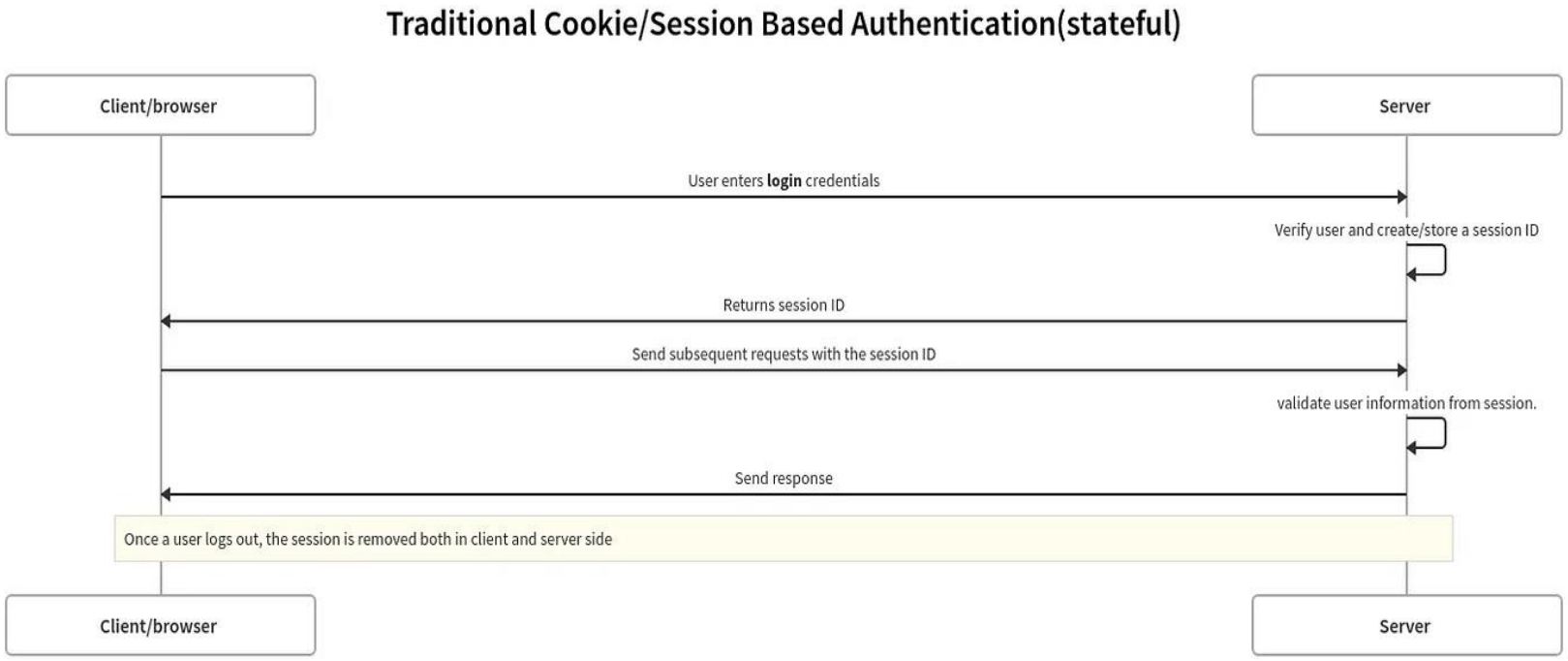
(2) The ProviderManager is configured to use an AuthenticationProvider of type DaoAuthenticationProvider.

(3) DaoAuthenticationProvider looks up the UserDetails from the UserDetailsService.

(4) DaoAuthenticationProvider then uses the PasswordEncoder to validate the password on the UserDetails returned in the previous step.

(5) When authentication is successful, the Authentication that is returned is of type UsernamePasswordAuthenticationToken and has a principal that is the UserDetails returned by the configured UserDetailsService. Ultimately, the returned UsernamePasswordAuthenticationToken will be set on the SecurityContextHolder by the authentication Filter.

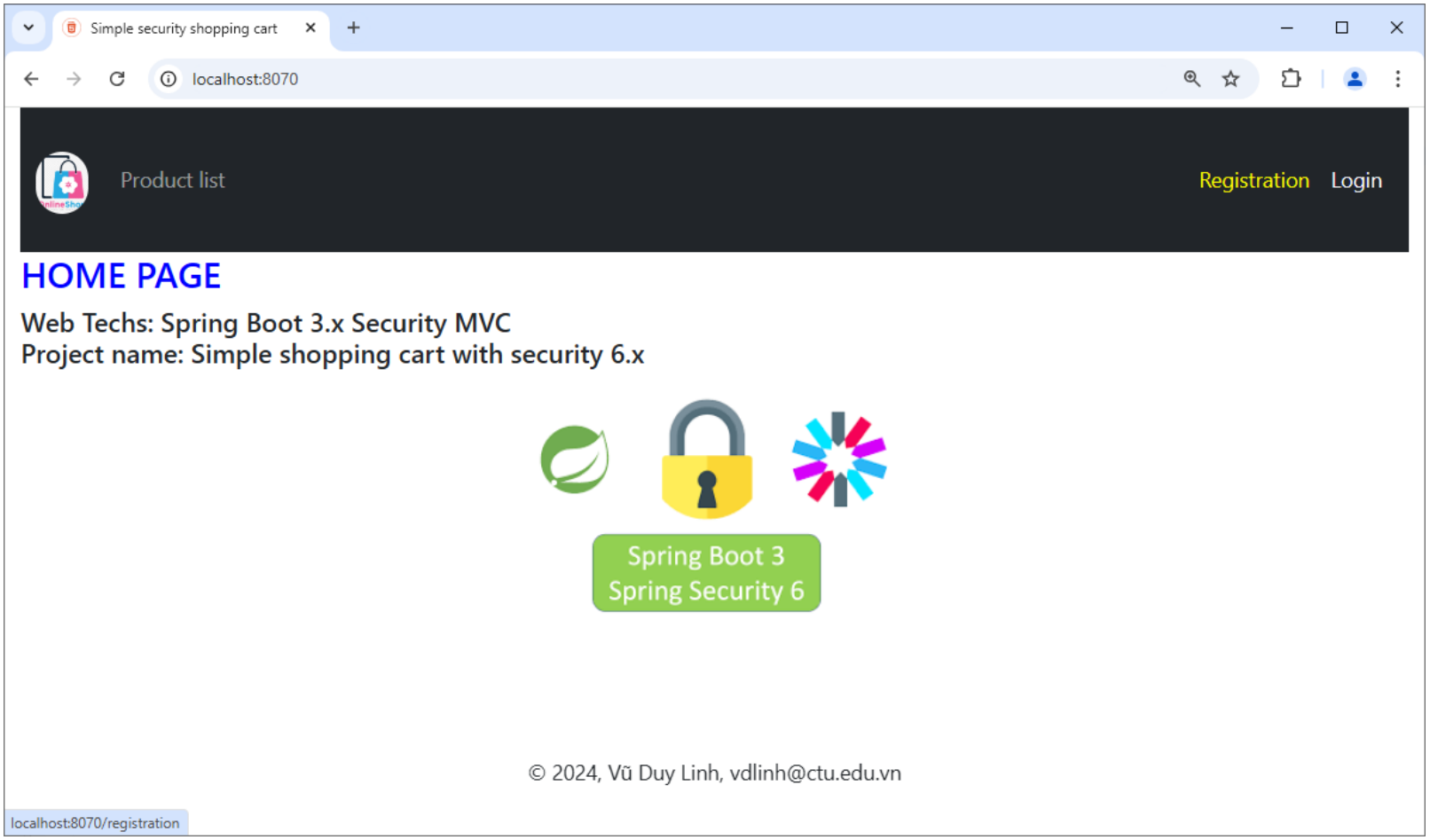
- **Stateful Authentication (Cookie/Session Based Authentication)**
  - This is the default and traditional method for handling user authentication. In this approach, the backend is responsible for creating, storing the session ID, and verifying the user's identity.
  - Here is how it works: The server creates a session ID upon a user's login request, storing it in either a database or an in-memory cache on the server. This session ID is then stored on a cookie in the user's browser. With each subsequent request, the server receives the cookie containing the session ID and validates the user's identity by comparing it with the corresponding session information stored on the server.



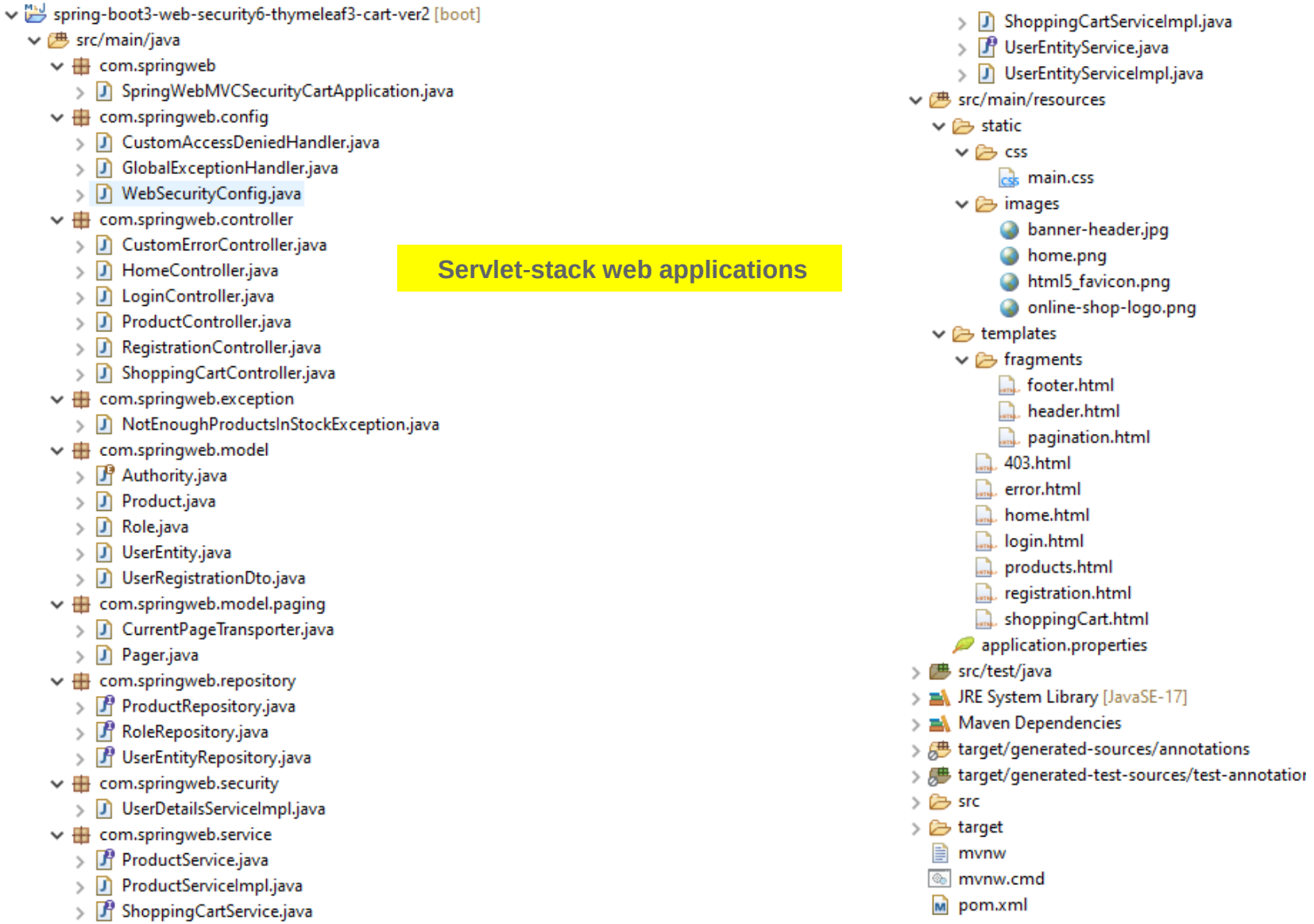

Source: https://medium.com/@minadev

Ref: https://medium.com/@minadev/authentication-and-authorization-with-spring-security-bf22e985f2cb

- **Project 3: The web of Simple security shopping cart**
  - using **Session Based Authentication,**
  - D:\java-web\...**spring-boot3-web-security6-thymeleaf3-cart-ver2**
  - Referenced from Dusan Reljic: https://github.com/reljicd/spring-boot-shopping-cart



# Project 3 ◆ Spring Boot Security MVC ◆ Chap 4



pom.xml

```xml
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

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

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

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

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

    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity6</artifactI
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>5.3.0</version>
    </dependency>

    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>font-awesome</artifactId>
        <version>6.4.0</version>
    </dependency>

    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>webjars-locator</artifactId>
        <version>0.47</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

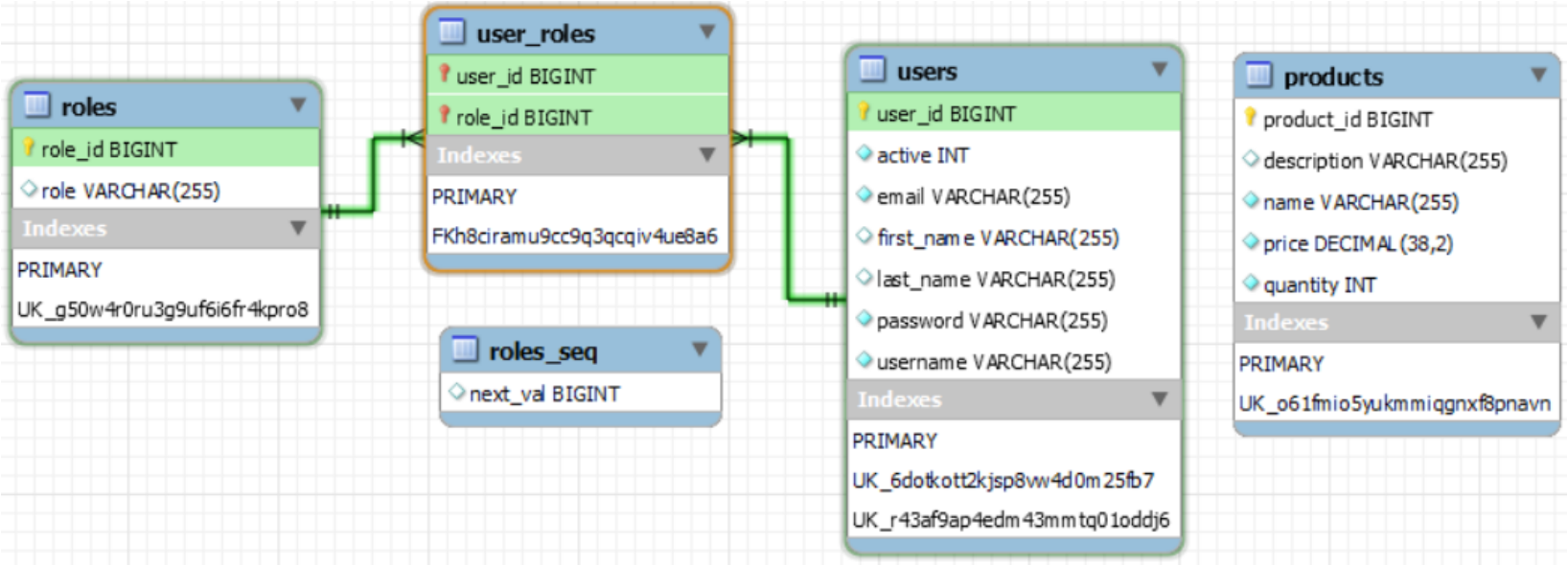
WebSecurityConfig.java

```java
package com.springweb.config;

import org.springframework.beans.factory.annotation.Autowired;
/*
 * @author Vu Duy Linh
 * References:
 *    https://www.javainuse.com/webseries/spring-security-jwt/chap3
 *    https://www.bezkoder.com/websecurityconfigureradapter-deprecated-spring-boot/
 * */

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
    @Autowired
    private CustomAccessDeniedHandler customAccessDeniedHandler;

//  @Autowired  private UserDetailsServiceImpl userDetailsServiceImpl;

    @Bean   //encoder pwd to inject into UserEntityServiceImpl
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

//  @Bean  //can be omitted
//  AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
//      return authConfig.getAuthenticationManager();
//  }
//
//  @Bean //can be omitted
//  DaoAuthenticationProvider authenticationProvider() {
//      DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
//      authProvider.setUserDetailsService(userDetailsServiceImpl);
//      authProvider.setPasswordEncoder(passwordEncoder());
//      return authProvider;
//  }

```

# Project 3 ◆ Spring Boot Security MVC ◆ Chap 4

```
/* https://docs.spring.io/spring-security/reference/servlet/authentication/session-management.html
 * "In Spring Security 6, the default is that authentication mechanisms themselves
 *  must invoke the SessionAuthenticationStrategy".
 */
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http.authorizeHttpRequests( (authorizeRequests) -> authorizeRequests
                .requestMatchers("/","/home", "/login", "/403", "/css/**", "/images/**").permitAll()
                .requestMatchers("/registration", "/webjars/**").permitAll()

                // Nếu muốn cho khách vãng lai (chưa đăng nhập) xem danh sách các sản phẩm
                .requestMatchers("/products").permitAll()

                .anyRequest().authenticated()
            )
            .formLogin((formBased) -> {
                    formBased
//                  .usernameParameter("email")  --> NEED <input type="email" name="email">
//                  .passwordParameter("password")
                    .loginPage("/login")
                    .defaultSuccessUrl("/products").permitAll()
                    .failureUrl("/login?error=true").permitAll();
                }
            )
            .logout((logout) -> logout
                .invalidateHttpSession(true)
                .clearAuthentication(true)
                .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                .logoutSuccessUrl("/login?logout")
                .permitAll()
            )
            .exceptionHandling(excpt -> excpt
                .accessDeniedHandler(customAccessDeniedHandler)
                .accessDeniedPage("/403")
            );
//      http.authenticationProvider(authenticationProvider());  //can be omitted

        return http.build();
    }
}
```

- EER Diagram




| user_id | active | email | first_name | last_name | password | username |
|---|---|---|---|---|---|---|
| 1 | 1 | vdlinh@ctu.edu.vn | Duy Linh | Vũ | $2a$10$xBG3IOtJ5SGDeogV9KIcO.FWdjzQWSAFIf6KLkbxiGT66UfC4HetC | vdlinh |
| 2 | 1 | ntkthi@ctu.edu.vn | T.K.Thi | Nguyễn | $2a$10$.8WNhFNiWm/gB2wfdo83Z.3gSzHf0buFT9vxa0dQ5JMxZQoSqOoMW | ntkthi |

| user_id | role_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 4 |

| role_id ▲ 1 | role |
|---|---|
| 1 | OWNER |
| 2 | ADMIN |
| 3 | SALES |
| 4 | GUEST |

```java
@Table(name = "users")
@Entity
public class UserEntity {
    @Id  @GeneratedValue(strategy =
            GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long id;

    @Column(name = "email", unique = true,
            nullable = false)
    private String email;

    @Column(name = "password", nullable = false)
    @JsonIgnore
    private String password;

    @Column(name = "username", nullable = false,
            unique = true)
    private String username;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "active", nullable = false)
    private int active=1;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();

    public UserEntity() {
    }
```

Authority.java

```java
package com.springweb.model;

public enum Authority {
    OWNER,
    ADMIN,
    SALES,
    GUEST
}
```

Role.java

```java
package com.springweb.model;

import jakarta.persistence.*;

@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "role_id")
    private Long id;

    @Column(name = "role", unique = true)
    private String role;

    @ManyToMany(mappedBy = "roles")
    private Set<UserEntity> userEntities = new HashSet<>();

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }

    public Role() {}
```

UserRegistrationDto.java

```java
package com.springweb.model;

import javax.validation.constraints.Size;

/* Now, the goal of a DTO is to read the required information with as few database queries
 * as possible and to provide it in an efficient and easy to use (registration) form.
 *****/
public class UserRegistrationDto {
    @Size(min = 5,  max = 30,
      message = "Your password '${validatedValue}' must be between {min} and {max} characters long")
    private String password;

    @Size(min = 5,  max = 15,
      message = "Your username '${validatedValue}' must be between {min} and {max} characters long")
    private String username;

    @Size(min=1, message = "Please provide your first name")
    private String firstName;

    @Size(min=1, message = "Please provide your last name")
    private String lastName;

    @Size(
      min = 10,  max = 25,
      message = "Your email '${validatedValue}' must be between {min} and {max} characters long")
    private String email;

    @Size(min=1, max=10,
          message = "Your ROLE '${validatedValue}' must be between {min} and {max} characters long")
    private Set<String> roles;

    public UserRegistrationDto() {
    }

    public String getPassword() {
        return password;
    }
```

```
39
40      public UserRegistrationDto(
41          @Size(min = 5, max = 30, message = "Your password '${validatedValue}' "
42                  + "must be between {min} and {max} characters long") String password,
43          @Size(min = 5, max = 15, message = "Your username '${validatedValue}' "
44                  + "must be between {min} and {max} characters long") String username,
45          @Size(min = 1, message = "Please provide your first name") String firstName,
46          @Size(min = 1, message = "Please provide your last name") String lastName,
47          @Size(min = 10, max = 25, message = "Your email '${validatedValue}'"
48                  + " must be between {min} and {max} characters long") String email,
49          @Size(min = 1, max = 10, message = "Your ROLE '${validatedValue}' "
50                  + "must be between {min} and {max} characters long") Set<String> roles) {
51          super();
52          this.password = password;
53          this.username = username;
54          this.firstName = firstName;
55          this.lastName = lastName;
56          this.email = email;
57          this.roles = roles;
58      }
59
60      public void setPassword(String password) {
61          this.password = password;
62      }
63
64      public String getUsername() {
65          return username;
66      }
67
68      public void setUsername(String username) {
69          this.username = username;
70      }
71
72      public String getFirstName() {
73          return firstName;
74      }
75
```

```
76      public void setFirstName(String firstName) {
77          this.firstName = firstName;
78      }
79
80      public String getLastName() {
81          return lastName;
82      }
83
84      public void setLastName(String lastName) {
85          this.lastName = lastName;
86      }
87
88      public String getEmail() {
89          return email;
90      }
91
92      public void setEmail(String email) {
93          this.email = email;
94      }
95
96      public Set<String> getRoles() {
97          return roles;
98      }
99
100     public void setRoles(Set<String> roles) {
101         this.roles = roles;
102     }
103
104     @Override
105     public String toString() {
106         return "UserRegistrationDto [password=" + password
107                 + ", username=" + username + ", firstName=" + firstName
108                 + ", lastName=" + lastName + ", email=" + email + ", roles=" + roles + "]";
109     }
110 }
```

RoleRepository.java

```java
package com.springweb.repository;

import java.util.Set;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.springweb.model.Role;

public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByRole(String role);

    @Query("SELECT r FROM Role r WHERE r.role=?1")  //Call one by one
    Set<Role> findByRoles(String role);

    @Query(value = "SELECT r FROM Role r WHERE r.role IN :roleNames")
    Set<Role> findRolesByNameSet(@Param("roleNames") Set<String> roleNames);

}
```

UserEntityRepository.java

```java
package com.springweb.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserEntityRepository extends JpaRepository<UserEntity, Long> {
    Optional<UserEntity> findByEmail(String email);
    Optional<UserEntity> findByUsername(String username);
}
```

UserDetailsServiceImpl.java

```java
package com.springweb.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.springweb.model.UserEntity;
import com.springweb.repository.UserEntityRepository;
import java.util.ArrayList;
import java.util.Collection;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired  private UserEntityRepository userEntityRepository;

    @Override @Transactional
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserEntity userEntity = userEntityRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("Email " + username + " not found"));
        return new User(userEntity.getUsername(), userEntity.getPassword(), getAuthorities(userEntity));
    }

    private static Collection<? extends GrantedAuthority> getAuthorities(UserEntity userEntity) {
        Collection<SimpleGrantedAuthority> authorities = new ArrayList<>();
        userEntity.getRoles().forEach(role -> {
            authorities.add(new SimpleGrantedAuthority(role.getRole()));
        });
        return authorities;
    }
}
```

RegistrationController.java

```java
@Controller
public class RegistrationController {
    @Autowired private RoleRepository roleRepository;
    @Autowired private UserEntityServiceImpl userEntityServiceImpl;

    @GetMapping("/registration")
    public String registration(Model model) {
        UserRegistrationDto userRegistrationDto = new UserRegistrationDto();
        model.addAttribute("userView", userRegistrationDto);

        return "/registration";
    }

    @PostMapping("/registration")
    public ModelAndView createNewUser(@ModelAttribute("userView") @Valid  UserRegistrationDto
                            userRegistration, BindingResult bindingResult) {

        //BindingResult là nơi Spring chứa kết quả của việc xác thực dữ liệu.
        if (userEntityServiceImpl.findByEmail(userRegistration.getEmail()).isPresent()) {
            bindingResult.rejectValue("email", "error.user",
                    "The email is already registered, please use another one.");

        }

        if (userEntityServiceImpl.findByUsername(userRegistration.getUsername()).isPresent()) {
            bindingResult.rejectValue("username", "error.user",
                        "There is already a user registered with the username provided");
        }
```

```
49
50          ModelAndView modelAndView = new ModelAndView();
51
52          if (bindingResult.hasErrors()) {
53              modelAndView.setViewName("/registration");
54              return  modelAndView;
55          } else {
56              UserEntity userEntity= new UserEntity();
57              userEntity.setFirstName(userRegistration.getFirstName());
58              userEntity.setLastName(userRegistration.getLastName());
59              userEntity.setUsername(userRegistration.getUsername());
60              userEntity.setEmail(userRegistration.getEmail());
61              userEntity.setPassword(userRegistration.getPassword());
62
63              // Convert enum.values() into Roles:
64              Set<Role> userRoles = roleRepository.findRolesByNameSet(userRegistration.getRoles());
65              userEntity.setRoles(userRoles);
66
67              userEntityServiceImpl.createNewUserEntity(userEntity);
68
69              modelAndView.addObject("successMessage","Username, "+ userEntity.getUsername()+
70                      " has been registered successfully. Now you should login for shopping.");
71
72              modelAndView.addObject("userView", userRegistration);
73
74              modelAndView.setViewName("/registration");  //"/login" or sth like that if u want
75              return modelAndView;
76          }
77      }
78  }
```

UserEntityServiceImpl.java

```java
package com.springweb.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import com.springweb.model.UserEntity;
import com.springweb.repository.UserEntityRepository;
import java.util.Optional;

@Service
public class UserEntityServiceImpl implements UserEntityService {
    @Autowired
    private  UserEntityRepository userEntityRepository;

    @Autowired
    private PasswordEncoder pwEncoder;

    @Override
    public Optional<UserEntity> findByUsername(String username) {
        return userEntityRepository.findByUsername(username);
    }

    @Override
    public Optional<UserEntity> findByEmail(String email) {
        return userEntityRepository.findByEmail(email);
    }

    @Override
    public void createNewUserEntity(UserEntity userEntity) {
        userEntity.setPassword(pwEncoder.encode(userEntity.getPassword()));
     userEntityRepository.save(userEntity);//.saveAndFlush(user);
    }
}
```

# Project 3 ◆ Spring Boot Security MVC ◆ Chap 4

ProductController.java

```java
package com.springweb.controller;
import org.springframework.beans.factory.annotation.Autowired;

@Controller
public class ProductController {
    @Autowired private  ProductService productService;
    private static final int INITIAL_PAGE = 0;

    @GetMapping(value={"/products"})
    public ModelAndView fetchProducts(@RequestParam("page") Optional<Integer> page) {

        // Evaluate page. If requested parameter is null or less than 0 (to prevent exception),
        // return initial size. Otherwise, return value of param decreased by 1.

        int evalPage = (page.orElse(0) < 1) ? INITIAL_PAGE : page.get() - 1;  //pageNo

        //////////To click BUY a product is still in current page,
        /// we use the object CurrentPageTransporter to pass it into ShoppingCartController
        int currentPage = evalPage + 1;
        CurrentPageTransporter.setCurrentPage(currentPage);
        System.out.println("---------------SET qsCurrentPage------- = "+currentPage);
        ////////////////////////////////////////////////////////////////////////////////

        Sort sort =  Sort.by("Name").ascending();
        Page<Product> products = productService.findAllProductsPageable(
                PageRequest.of(evalPage, 5, sort));

        Pager pager = new Pager(products);

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("products", products);
        modelAndView.addObject("pager", pager);
        modelAndView.setViewName("/products");

        return modelAndView;
    }
}
```

```java
package com.springweb.model.paging;

import org.springframework.data.domain.Page;
import com.springweb.model.Product;


/** Pager class, a utility class used for preparing pagination component, is used to wrap
the result list of products.  */

public class Pager {
  private final Page<Product> products;


  public Pager(Page<Product> products) {
    this.products = products;
  }


  public int getPageIndex() {
    return products.getNumber() + 1;
  }


  public int getPageSize() {
    return products.getSize();
  }


  public boolean hasNext() {
    return products.hasNext();
  }

  public boolean hasPrevious() {
    return products.hasPrevious();
  }


  public int getTotalPages() {
    return products.getTotalPages();
  }


  public long getTotalElements() {
    return products.getTotalElements();
  }


  public boolean indexOutOfBounds() {
    return  this.getPageIndex() < 0   ||
            this.getPageIndex() >
                this.getTotalElements();
    }
}
```

ProductServiceImpl.java

```java
package com.springweb.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class ProductServiceImpl implements ProductService {
    @Autowired private ProductRepository productRepository;

    @Override
    public Page<Product> findAllProductsPageable(Pageable pageable) {
        return productRepository.findAll(pageable);
    }

    @Override
    public Optional<Product> findById(Long id) {
        return productRepository.findById(id);
    }
}
```

CurrentPageTransporter.java

```java
package com.springweb.model.paging;

/* How to pass an object from one controller to another controller.
 * Using CurrentPageTransporter to share between controllers.
 *
 * */
public class CurrentPageTransporter { //Transportation class
     private static int currentPage;

    public static int getCurrentPage() {
        return currentPage;
    }

    public static void setCurrentPage(int currentPage) {
        CurrentPageTransporter.currentPage = currentPage;
    }

}
```

- CurrentPageTransporter.java

CurrentPageTransporter.java

```java
package com.springweb.model.paging;

/*
 * @author Vu Duy Linh
 *
 *
 * How to pass an object from one controller to another controller.
 * Using CurrentPageTransporter to share between controllers.
 *
 * https://stackoverflow.com/questions/47698081/how-to-pass-an-object-from-one-controller-to-another-controller/47698398
 * */

public class CurrentPageTransporter { //Transportation class
    private static int currentPage;

    public static int getCurrentPage() {
        return currentPage;
    }

    public static void setCurrentPage(int currentPage) {
        CurrentPageTransporter.currentPage = currentPage;
    }

}

```

ShoppingCartController.java

```java
package com.springweb.controller;
import org.springframework.beans.factory.annotation.Autowired;
/*
 * @author Vu Duy Linh
 *
 * Note: Some source code was referenced from Dusan Reljic
 *       https://github.com/reljicd/spring-boot-shopping-cart
 * */

@Controller
public class ShoppingCartController {
    @Autowired private ShoppingCartService shoppingCartService;

    @Autowired private  ProductService productService;

    @GetMapping("/shoppingCart")
    public ModelAndView shoppingCart() {
        ModelAndView modelAndView = new ModelAndView("/shoppingCart");
        modelAndView.addObject("products", shoppingCartService.getProductsInCart());
        modelAndView.addObject("total", shoppingCartService.getTotal().toString());

        return modelAndView;
    }

    //Sau khi click "BUY" button to buy a product, then redirect to exact the current page of products list.
    @GetMapping("/shoppingCart/addProduct/{productId}")
    public String addProductToCartStayCurrentPage(@PathVariable("productId") Long productId) {

        productService.findById(productId).ifPresent(shoppingCartService::addProduct);
        int qsCurrentPage = CurrentPageTransporter.getCurrentPage();

        return "redirect:/products"+"?page=" + qsCurrentPage;
    }

```

```
/*    Sau khi click "BUY" button to buy a product, then redirect to the first of page list of products.
      @GetMapping("/shoppingCart/addProduct/{productId}")
      public String addProductToCart(@PathVariable("productId") Long productId) {
        productService.findById(productId).ifPresent(shoppingCartService::addProduct);
        return "redirect:/products";
      } */

    @GetMapping("/shoppingCart/removeProduct/{productId}")
    public ModelAndView removeProductFromCart(@PathVariable("productId") Long productId) {

        productService.findById(productId).ifPresent(shoppingCartService::removeProduct);

        return shoppingCart();
    }

    @GetMapping("/shoppingCart/checkout")
    public ModelAndView checkoutCart() {

        ModelAndView modelAndView = new ModelAndView("/shoppingCart");
        modelAndView.addObject("products", shoppingCartService.getProductsInCart());
        modelAndView.addObject("total", shoppingCartService.getTotal().toString());

        try {
            shoppingCartService.checkout();
            modelAndView.addObject("success", "You have checked out successfully.");

        } catch (NotEnoughProductsInStockException e) {
             modelAndView.addObject("error", "Sorry, there is not enough quantity for your order.");
            return shoppingCart().addObject("outOfStockMessage", e.getMessage());
        }

        return modelAndView;
    }
}
```

ShoppingCartServiceImpl.java

```java
package com.springweb.service;
import org.springframework.beans.factory.annotation.Autowired;

/*
 * Shopping cart is implemented with Stateful authentication (session-based authentication or cookie-based authentication)
 * @author Vu Duy Linh
 *
 * Some source code was referenced from Dusan Reljic:  https://github.com/reljicd/spring-boot-shopping-cart
 * */

@Transactional
@Service
public class ShoppingCartServiceImpl implements ShoppingCartService {

    @Autowired private ProductRepository productRepository;
    private Map<Product, Integer> products = new HashMap<>();

    @Override
    public void addProduct(Product product) {
        if (products.containsKey(product)) {//If product is in the map just increment quantity by 1
            products.replace(product, products.get(product) + 1);
        } else {
            products.put(product, 1);  //it with quantity 1
        }
    }

    @Override
    public void removeProduct(Product product) {
        if (products.containsKey(product)) {
            if (products.get(product) > 1)  //just decrement quantity by 1.
                products.replace(product, products.get(product) - 1);
            else if (products.get(product) == 1) { //remove it from map
                products.remove(product);
            }
        }
    }

```

```
50      @Override
51      public Map<Product, Integer> getProductsInCart() {
52      /* The real map is products, so you can modify value of the products value (in back-end),
53         BUT user (in view/ front-end) can not modify in Collections.unmodifiableMap(products)  */
54
55          return Collections.unmodifiableMap(products); //unmodifiable copy of the map.
56      }
57
58      @Override     //  Checkout will rollback if there is not enough of some product in stock
59      public void checkout() throws NotEnoughProductsInStockException {
60          Product product;
61          for (Map.Entry<Product, Integer> entry : products.entrySet()) {
62              // Refresh quantity for every product before checking
63              product = productRepository.getReferenceById(entry.getKey().getId());
64
65              if (product.getQuantity() < entry.getValue())
66                  throw new NotEnoughProductsInStockException(product);
67
68              entry.getKey().setQuantity(product.getQuantity() - entry.getValue());
69          }
70          productRepository.saveAll(products.keySet());
71          productRepository.flush();
72
73          //Nên quản lý session để chọn phương thức thanh toán hoặc kết thúc giao dịch ở đây.
74      }
75
76      @Override
77      public BigDecimal getTotal() {
78          return products.entrySet().stream().map(entry -> entry.getKey().getPrice()
79                  .multiply(BigDecimal.valueOf(entry.getValue())))
80                  .reduce(BigDecimal::add).orElse(BigDecimal.ZERO);
81      }
82  }
```

ProductService.java

```java
package com.springweb.service;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import com.springweb.model.Product;
import java.util.Optional;

public interface ProductService {
    Optional<Product> findById(Long id);
    Page<Product> findAllProductsPageable(Pageable pageable);
}
```

ProductServiceImpl.java

```java
package com.springweb.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import com.springweb.model.Product;
import com.springweb.repository.ProductRepository;
import java.util.Optional;

@Service
public class ProductServiceImpl implements ProductService {
    @Autowired private ProductRepository productRepository;

    @Override
    public Page<Product> findAllProductsPageable(Pageable pageable) {
        return productRepository.findAll(pageable);
    }

    @Override
    public Optional<Product> findById(Long id) {
        return productRepository.findById(id);
    }
}
```

← header.html

```
<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">

<!--/* this is header::head> */-->
<head th:fragment="head">
  <title th:attr="data-custom=#{thymeleaf.app.title}">Simple security shopping cart</title>
  <link rel="icon" type="image/png" href="../../images/html5_favicon.png"/>
  <link th:rel="stylesheet" th:href="@{/webjars/bootstrap/5.3.0/css/bootstrap.min.css}" />
  <link th:rel="stylesheet" th:href="@{/webjars/font-awesome/6.4.0/css/all.css}" />
  <link rel="stylesheet" th:href="@{/css/main.css}" href="../../css/main.css" />
</head>

<body>
  <section th:fragment="navbar">
    <nav class="navbar navbar-expand-sm navbar-dark bg-dark py-4">
      <div class="container-fluid">

        <a class="navbar-brand" th:href="@{/}">
          <img src="../../images/online-shop-logo.png" alt="Avatar Logo"
          style="width:40px;" class="rounded-pill">
        </a>

        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
           data-bs-target="#mynavbar"><span class="navbar-toggler-icon"></span></button>

        <div class="collapse navbar-collapse" id="mynavbar">
           <ul class="navbar-nav me-auto">
              <li class="nav-item">
                <a class="nav-link" th:href="@{/products}">Product list</a>
              </li>
           </ul>
```

```
        <div class="d-flex">
           <!-- show shoppingCart only if user is authenticated -->
           <ul class="navbar-nav me-auto" sec:authorize="isAuthenticated()">
             <li class="nav-item active">
                <a class="nav-link" th:href="@{/shoppingCart}">Shopping Cart</a></li></ul>

          <!-- show registration only if user is not yet authenticated -->
          <ul class="navbar-nav me-auto" sec:authorize="!isAuthenticated()">
            <li class="nav-item active">
               <a class="nav-link" th:href="@{/registration}">Registration</a></li></ul>


          <!-- show login only if user is not yet authenticated -->
          <ul class="navbar-nav me-auto" sec:authorize="!isAuthenticated()">
            <li class="nav-item">
                <a class="nav-link active" th:href="@{/login}">Login</a></li></ul>

          <!-- show sign out only if user is authenticated -->
          <ul class="navbar-nav me-auto" sec:authorize="isAuthenticated()">
            <li class="nav-item">
                <a class="nav-link active" th:href="@{/logout}">Sign Out</a>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </nav>
 </section>
</body>

</html>
```

← pagination.html

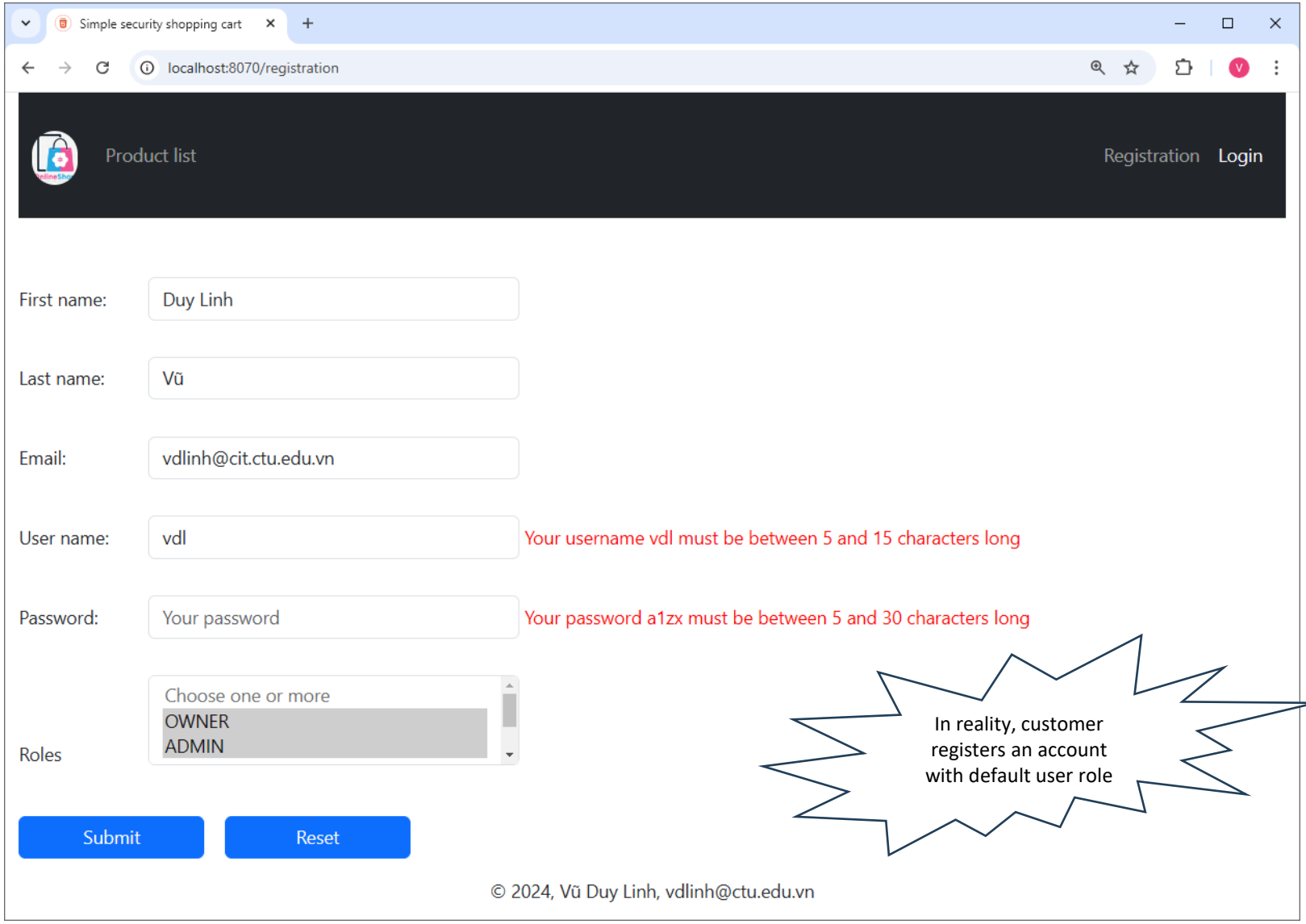
```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<section th:fragment="pagination">
  <div class="pagination" th:with="baseUrl=${URLparameter}">

    <span th:if="${pager.indexOutOfBounds()}">Page is out of bounds. Go back to
       <a class="pageLink“ th:href="@{${baseUrl}(page=1)}">Home</a>.
    </span>

    <span th:unless="${pager.indexOutOfBounds()}">
      <span th:if="${pager.hasPrevious()}">
        <a class="pageLink" th:href="@{${baseUrl}(page=1)}">&laquo; first</a>
        <a class="pageLink"
           th:href="@{${baseUrl}(page=${pager.getPageIndex() - 1})}">previous</a>
      </span>

      <span th:if="${pager.getTotalPages() != 1}" th:text="'Page ' +
              ${pager.getPageIndex()} + ' of ' + ${pager.getTotalPages()} + '.'">
      </span>

      <span th:if="${pager.hasNext()}">
        <a class="pageLink"
           th:href="@{${baseUrl}(page=${pager.getPageIndex() + 1})}">next</a>
        <a class="pageLink"
           th:href="@{${baseUrl}(page=${pager.getTotalPages()})}">last &raquo;</a>
      </span>

    </span>
  </div>
</section>
</body>
</html>
```

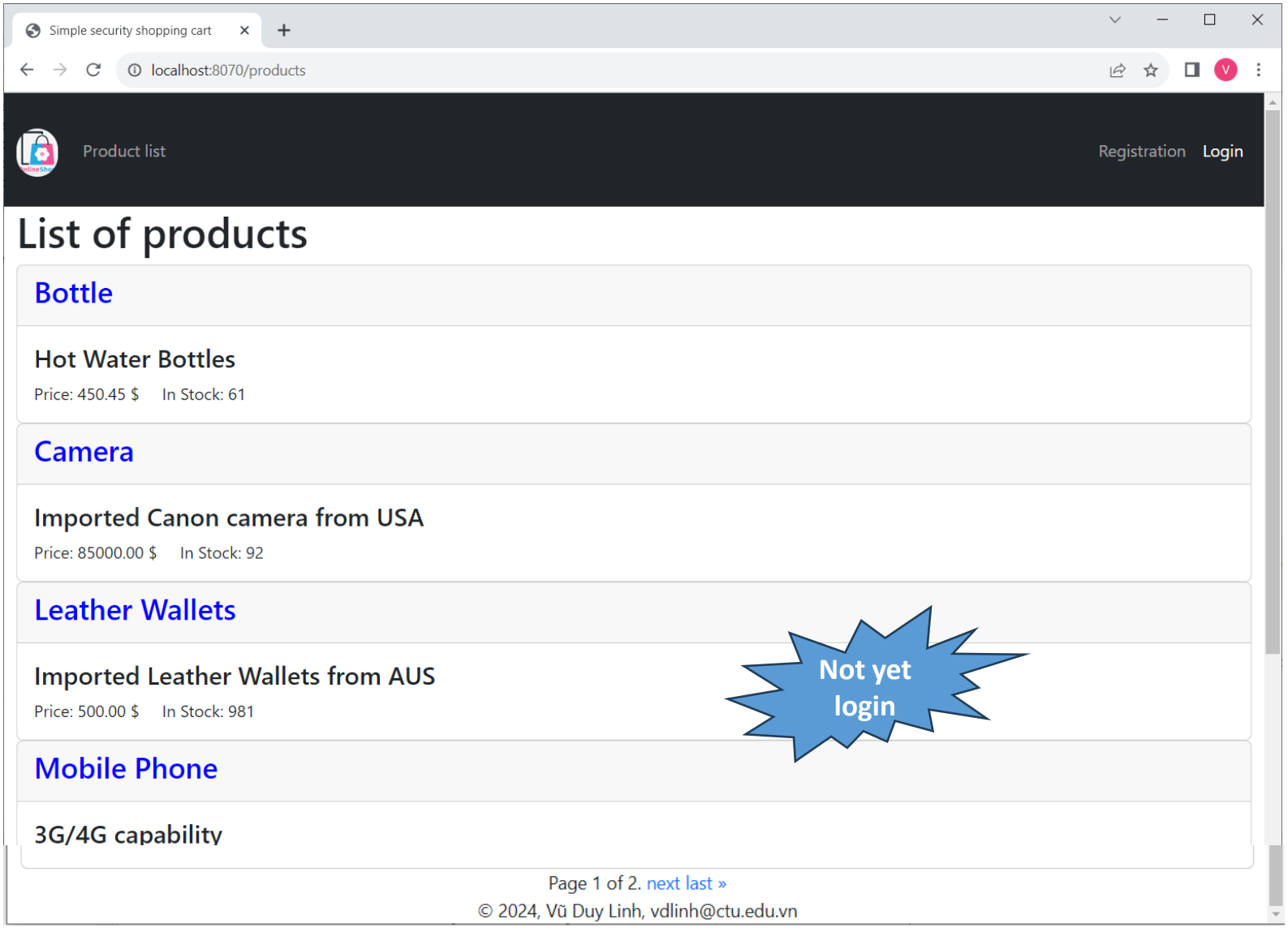
← footer.html

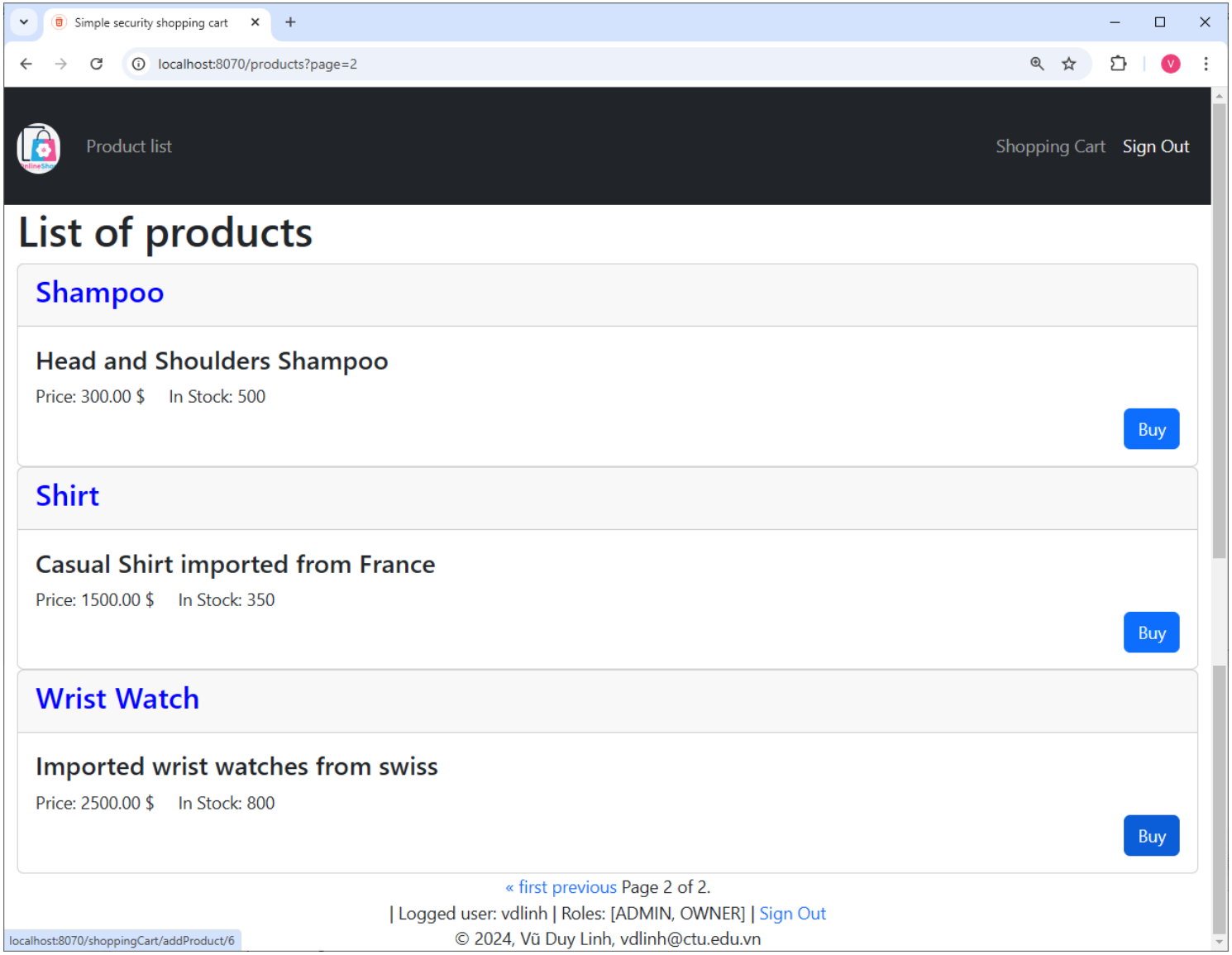
```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">

<body>
  <!-- this is footer::footer -->
  <section th:fragment="footer">

    <div class="container">
      <div class="row">
        <div class="col-sm-12 text-center">

          <span sec:authorize="isAuthenticated()">
            | Logged user: <span sec:authentication="name">
            </span> | Roles: <span sec:authentication="principal.authorities"></span> |
            <a th:href="@{/logout}">Sign Out</a>
          </span>

          <p class="text-center fs-6 ">
             &copy; 2024, Vũ Duy Linh, vdlinh@ctu.edu.vn  </p>

        </div>

      </div>
    </div>

    <script th:src="@{/webjars/bootstrap/5.3.0/js/bootstrap.bundle.min.js}"></script>
  </section>

</body>
</html>
```

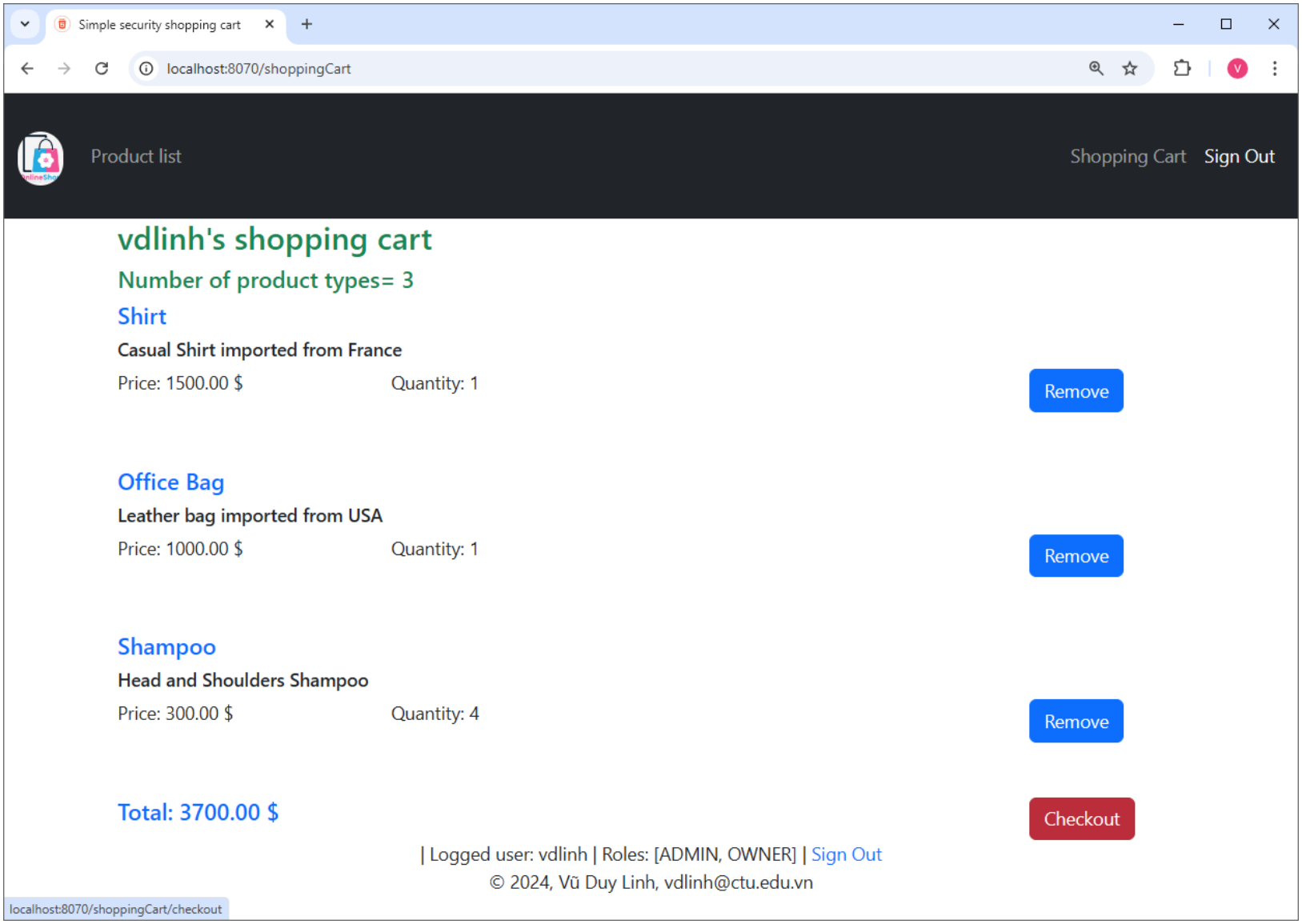
← products.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">

  <head th:replace="~{/fragments/header :: head}" />

  <body>
    <header th:insert="~{/fragments/header :: navbar}" />

    <main class="container-fluid">
      <h1>List of products</h1>
      <div class="card " style="width: 100%;" th:each="product : ${products}">
        <div class="card-header">
          <h3 th:text="${product.name}" />
        </div>

        <div class="card-body">
          <h4 th:text="${product.description}" >Description</h4>
          <span th:inline="text">Price: [[${product.price}]] $</span>    
          <span th:inline="text">In Stock: [[${product.quantity}]]</span>
          <a class="d-flex justify-content-end"
             th:href="@{'/shoppingCart/addProduct/{id}'(id=${product.id})}"
             sec:authorize="isAuthenticated()" th:if="${product.quantity}>0">
                <button type="button" class="btn btn-primary" th:text="Buy">Buy</button>
          </a>
        </div>
      </div>
      <div th:replace="~{/fragments/pagination :: pagination(URLparameter='/products')}"/>
    </main>

    <div th:block th:replace="~{/fragments/footer :: footer}"/>
</body>
</html>
```


Simple security shopping cart
localhost:8070/registration
Product list
Registration
Login
First name:
Duy Linh
Last name:
Vũ
Email:
vdlinh@cit.ctu.edu.vn
User name:
vdl
Your username vdl must be between 5 and 15 characters long
Password:
Your password
Your password a1zx must be between 5 and 30 characters long
Roles
Choose one or more
OWNER
ADMIN
In reality, customer registers an account with default user role
Submit
Reset
© 2024, Vũ Duy Linh, vdlinh@ctu.edu.vn


Simple security shopping cart
localhost:8070/products
Product list
Registration
Login
List of products
Bottle
Hot Water Bottles
Price: 450.45 $ In Stock: 61
Camera
Imported Canon camera from USA
Price: 85000.00 $ In Stock: 92
Leather Wallets
Imported Leather Wallets from AUS
Price: 500.00 $ In Stock: 981
Not yet login
Mobile Phone
3G/4G capability
Page 1 of 2. next last »
© 2024, Vũ Duy Linh, vdlinh@ctu.edu.vn


Simple security shopping cart
localhost:8070/products?page=2
Product list
Shopping Cart
Sign Out
List of products
Shampoo
Head and Shoulders Shampoo
Price: 300.00 $ In Stock: 500
Buy
Shirt
Casual Shirt imported from France
Price: 1500.00 $ In Stock: 350
Buy
Wrist Watch
Imported wrist watches from swiss
Price: 2500.00 $ In Stock: 800
Buy
« first previous Page 2 of 2.
| Logged user: vdlinh | Roles: [ADMIN, OWNER] | Sign Out
© 2024, Vũ Duy Linh, vdlinh@ctu.edu.vn
localhost:8070/shoppingCart/addProduct/6


Simple security shopping cart
localhost:8070/shoppingCart
Product list
Shopping Cart
Sign Out
vdlinh's shopping cart
Number of product types= 3
Shirt
Casual Shirt imported from France
Price: 1500.00 $
Quantity: 1
Remove
Office Bag
Leather bag imported from USA
Price: 1000.00 $
Quantity: 1
Remove
Shampoo
Head and Shoulders Shampoo
Price: 300.00 $
Quantity: 4
Remove
Total: 3700.00 $
Checkout
| Logged user: vdlinh | Roles: [ADMIN, OWNER] | Sign Out
© 2024, Vũ Duy Linh, vdlinh@ctu.edu.vn
localhost:8070/shoppingCart/checkout

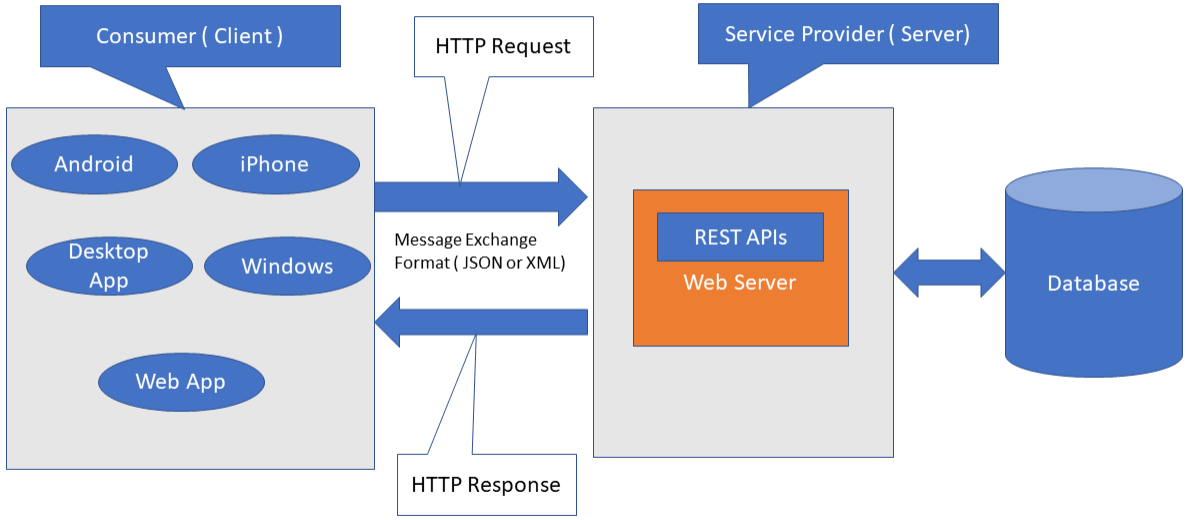
# Chap 5. Backend Rest API (Security) with Spring Boot

## Building Web Applications with Java

**Biên soạn: Vũ Duy Linh**
vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

- Basics of **Re**presentational **S**tate **T**ransfer (**REST**) is an architectural style for creating web services. REST is not standard, but it defines a set of constraints defined by Roy Fielding.
- The RESTful web service that we are going to develop in the following topics follows the REST architectural principles.

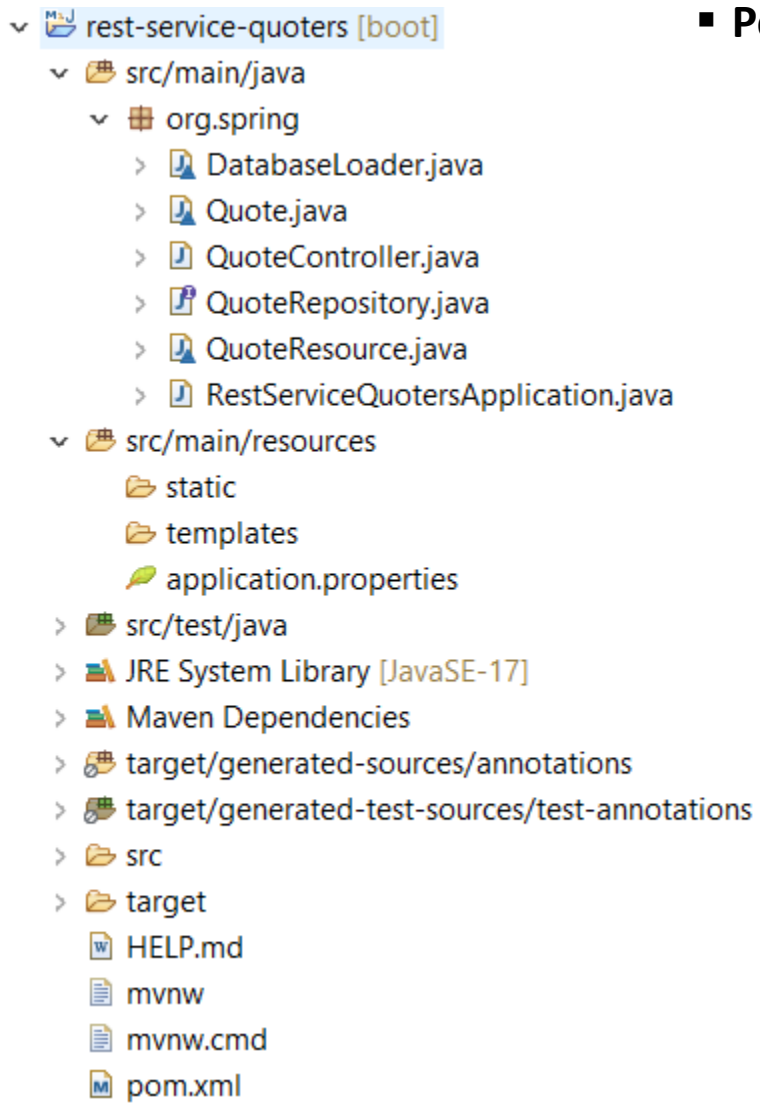
REST – Architecture




By Ramesh Fadatare ( Java Guides)

- REF: https://restfulapi.net/rest-architectural-constraints

- REST Architectural Constraints: The six constraints are as follows
  - Stateless: The server doesn't hold any information about the client state.
  - Client-server: The client and server act independently. The server does not send any information without a request from the client.
  - Cacheable: Many clients often request the same resources; therefore, it is useful to cache responses in order to improve performance.
  - Uniform interface
    - Requests from different clients look the same. Clients may include, for example, a browser, a Java application, and a mobile application.
    - **The uniform interface** is an important constraint and it stipulates that every REST architecture should have the following elements:
      - **Identification of resources**: There are resources with their unique identifiers, for example, URIs in web-based REST services. REST resources should expose easily understood directory structure URIs. Therefore, a good resource naming strategy is very important.
      - **Resource manipulation through representation**: When making a request to a resource, the server responds with a representation of the resource. Typically, the format of the representation is JSON or XML.
      - **Self descriptive messages**: Messages should have sufficient information that the server knows how to process them.
      - **Hypermedia and the Engine of Application State (HATEOAS)**: Responses can contain links to other areas of service.
  - Layered system: REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.
  - Code on demand: This is an optional constraint




**Project: consuming-rest-service-quotes-rt**

- **Pom.xml**

```
<name>rest-service-quoters</name>
<description>REST service to support the guides</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

- **class RestServiceQuotersApplication**

RestServiceQuotersApplication.java

```java
package org.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestServiceQuotersApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestServiceQuotersApplication.class, args);
    }

}
```

- **interface QuoteRepository**

QuoteRepository.java

```java
/* Copyright 2014-2018 the original author or authors.

package org.spring;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QuoteRepository extends JpaRepository<Quote, Long> {

}
```

Quote.java

```
2  * Copyright 2014-2018 the original author or authors.
16
17 package org.spring;
18
19 import java.util.Objects;
24
25 @Entity
26 class Quote {
27     @Id @GeneratedValue private Long id;
28     private String quote;
29
30     Quote(String quote) {
31         this.quote = quote;
32     }
```

// and setters/getters,…

QuoteResource.java

```
2  * Copyright 2014-2018 the original author or authors.
16
17 package org.spring;
18
19 import java.util.Objects;
20
21 class QuoteResource {
22
23     private String type;
24     private Quote value;
25
26     QuoteResource(Quote value, String type) {
27
28         this.value = value;
29         this.type = type;
30     }
```

DatabaseLoader.java

```
2  * Copyright 2014-2018 the original author or authors.
16
17 package org.spring;
18
19 import org.springframework.boot.CommandLineRunner;
20 import org.springframework.context.annotation.Bean;
21 import org.springframework.context.annotation.Configuration;
22
23 @Configuration
24 class DatabaseLoader {
25     @Bean
26     CommandLineRunner init(QuoteRepository repository) {
27         return args -> {
28             repository.save(new Quote("Working with Spring Boot is like pair-programming with the Spring developers."));
29             repository.save(new Quote("With Boot you deploy everywhere you can find a JVM basically."));
30             repository.save(new Quote("Spring has come quite a ways in addressing developer enjoyment and "
31                     + "ease of use since the last time I built an application using it."));
32             repository.save(new Quote(
33                     "Previous to Spring Boot, I remember XML hell, confusing set up, and " + "many hours of frustration."));
34             repository.save(new Quote("Spring Boot solves this problem. It gets rid of XML and wires up "
35                     + "common components for me, so I don't have to spend hours scratching my "
36                     + "head just to figure out how it's all pieced together."));
37             repository.save(new Quote("It embraces " + "convention over configuration, providing an experience on par with "
38                     + "frameworks that excel at early stage development, such as Ruby on " + "Rails."));
39             repository.save(new Quote("The real benefit of Boot, however, is that it's just Spring. That "
40                     + "means any direction the code takes, regardless of complexity, I know " + "it's a safe bet."));
41             repository.save(new Quote("I don't worry about my code scaling. Boot allows the "
42                     + "developer to peel back the layers and customize when it's appropriate "
43                     + "while keeping the conventions that just work."));
44             repository.save(new Quote("So easy it is to switch container in #springboot."));
45             repository.save(new Quote("Really loving Spring Boot, makes stand alone Spring apps easy."));
46             repository.save(new Quote("I have two hours today to build an app from scratch. @springboot to the rescue!"));
47             repository.save(new Quote("@springboot with @springframework is pure productivity! Who said in #java one has "
48                     + "to write double the code than in other langs? #newFavLib"));
49         };
50     }
51 }
```

QuoteController.java

```
 2 * Copyright 2014-2018 the original author or authors.
16
17 package org.spring;
18 import java.util.List;
26
27 @RestController
28 public class QuoteController {
29
30     private final static Quote NONE = new Quote("None");
31     private final static Random RANDOMIZER = new Random();
32
33     @Autowired private QuoteRepository repository;
34
35     @GetMapping("/api")
36     public List<QuoteResource> getAll() {
37
38         return repository.findAll().stream()
39             .map(quote -> new QuoteResource(quote, "success"))
40             .collect(Collectors.toList());
41     }
42
43     @GetMapping("/api/{id}")
44     public QuoteResource getOne(@PathVariable Long id) {
45
46         return repository.findById(id)
47             .map(quote -> new QuoteResource(quote, "success"))
48             .orElse(new QuoteResource(NONE, "Quote " + id + " does not exist"));
49     }
50
51     @GetMapping("/api/random")
52     public QuoteResource getRandomOne() {
53         return getOne(nextLong(1, repository.count() + 1));
54     }
55
56     private long nextLong(long lowerRange, long upperRange) {
57         return (long) (RANDOMIZER.nextDouble() * (upperRange - lowerRange)) + lowerRange;
58     }
59 }
```

- **GET** http://localhost:8089/api

GET http://localhost:8089/api

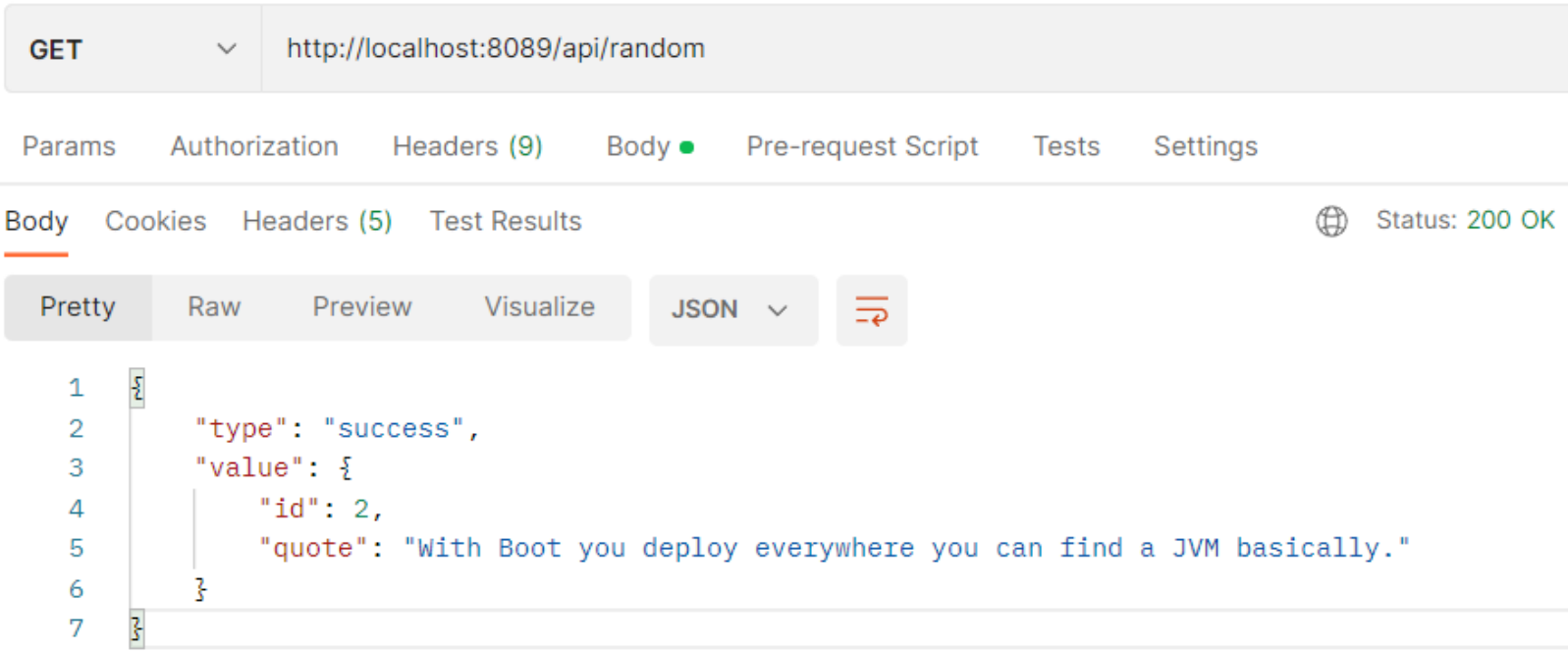
Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results Status: 200 OK Ti

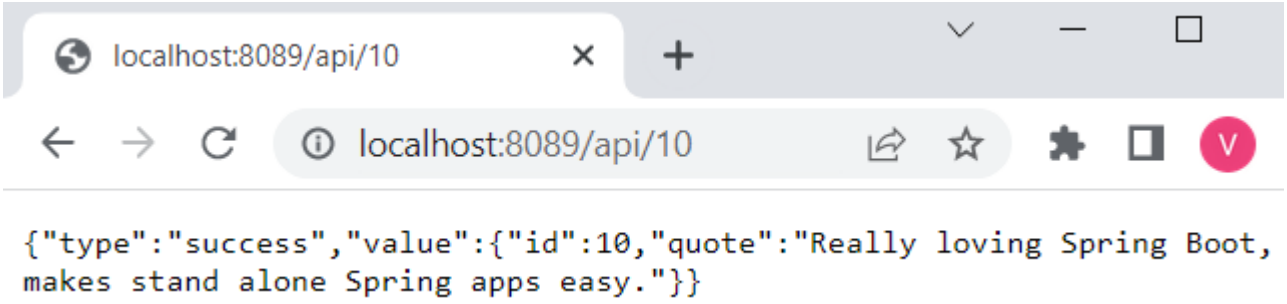
Pretty Raw Preview Visualize JSON

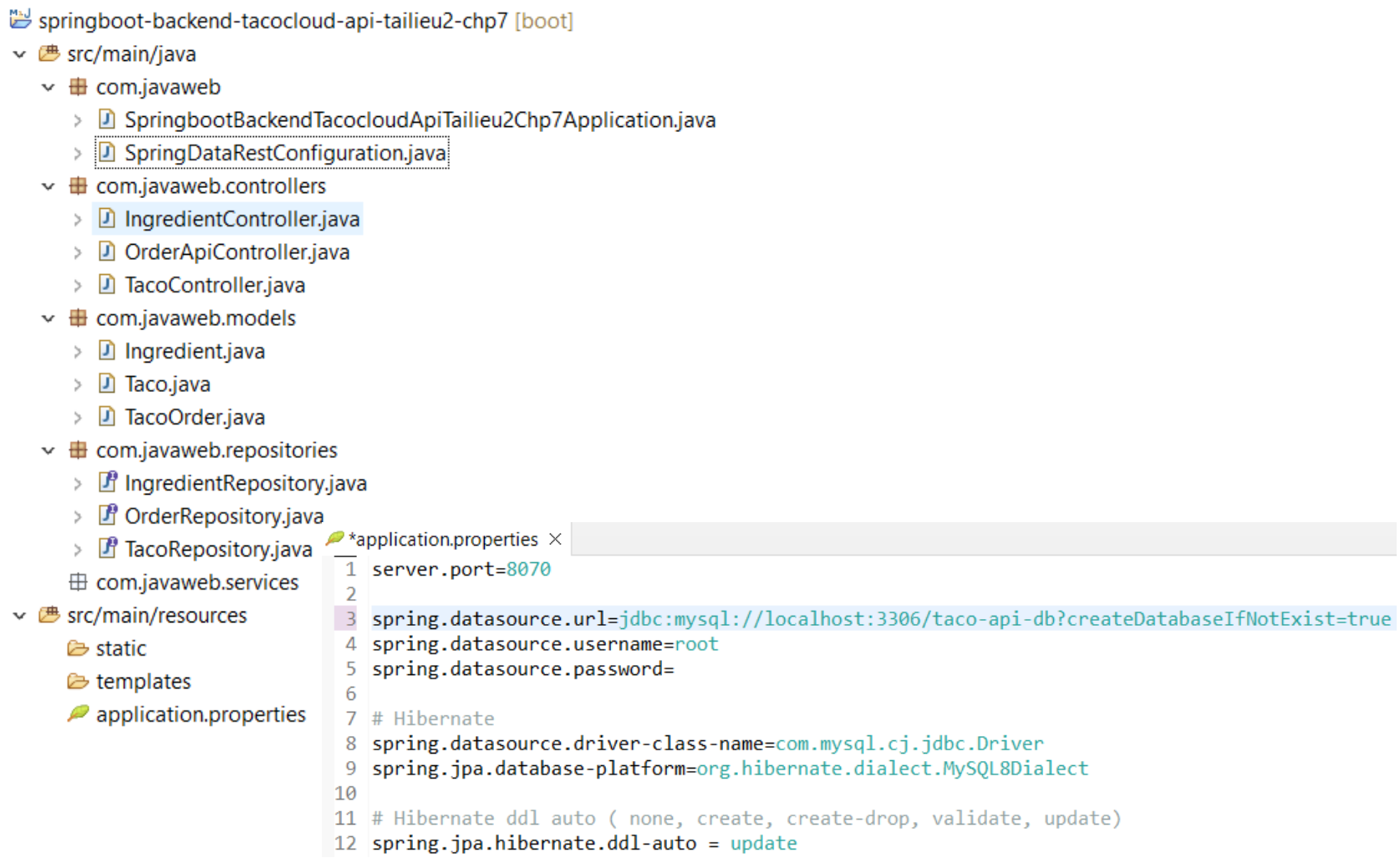

```
[
    {
        "type": "success",
        "value": {
            "id": 1,
            "quote": "Working with Spring Boot is like pair-programming with the Spring developers."
        }
    },
    {
        "type": "success",
        "value": {
            "id": 2,
            "quote": "With Boot you deploy everywhere you can find a JVM basically."
        }
    },
    {
        "type": "success",
        "value": {
            "id": 3,
            "quote": "Spring has come quite a ways in addressing developer enjoyment and ease of use since the last time I built an application using it."
        }
    },
    {
        "type": "success",
        "value": {
            "id": 4,
            "quote": "Previous to Spring Boot, I remember XML hell, confusing set up, and many hours of frustration."
```

- GET http://localhost:8089/api/random




- GET http://localhost:8089/api/3



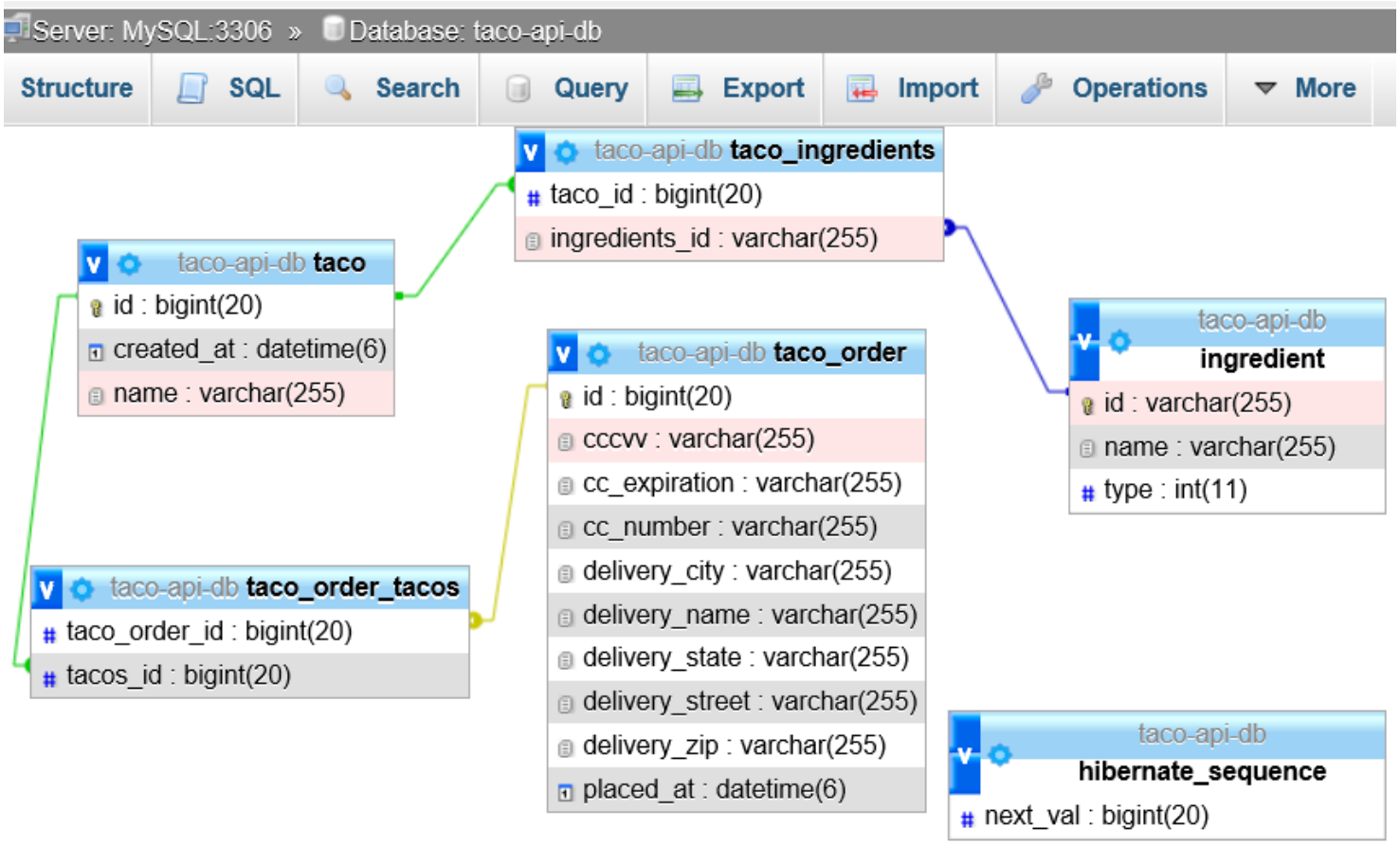

springboot-backend-tacocloud-api-tailieu2-chp7 [boot]
src/main/java
com.javaweb
SpringbootBackendTacocloudApiTailieu2Chp7Application.java
SpringDataRestConfiguration.java
com.javaweb.controllers
IngredientController.java
OrderApiController.java
TacoController.java
com.javaweb.models
Ingredient.java
Taco.java
TacoOrder.java
com.javaweb.repositories
IngredientRepository.java
OrderRepository.java
TacoRepository.java
com.javaweb.services
src/main/resources
static
templates
application.properties
*application.properties
server.port=8070
spring.datasource.url=jdbc:mysql://localhost:3306/taco-api-db?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
# Hibernate
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
# Hibernate ddl auto ( none, create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

- pom.xml

*springboot-backend-tacocloud-api-tailieu2-chp7/pom.xml

```
    <groupId>com.javaweb</groupId>
    <artifactId>springboot-backend-tacocloud-api-tailieu2-chp7</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot-backend-tacocloud-api-tailieu2-chp7</name>
    <description>Rest API with Spring Data REST</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-rest</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

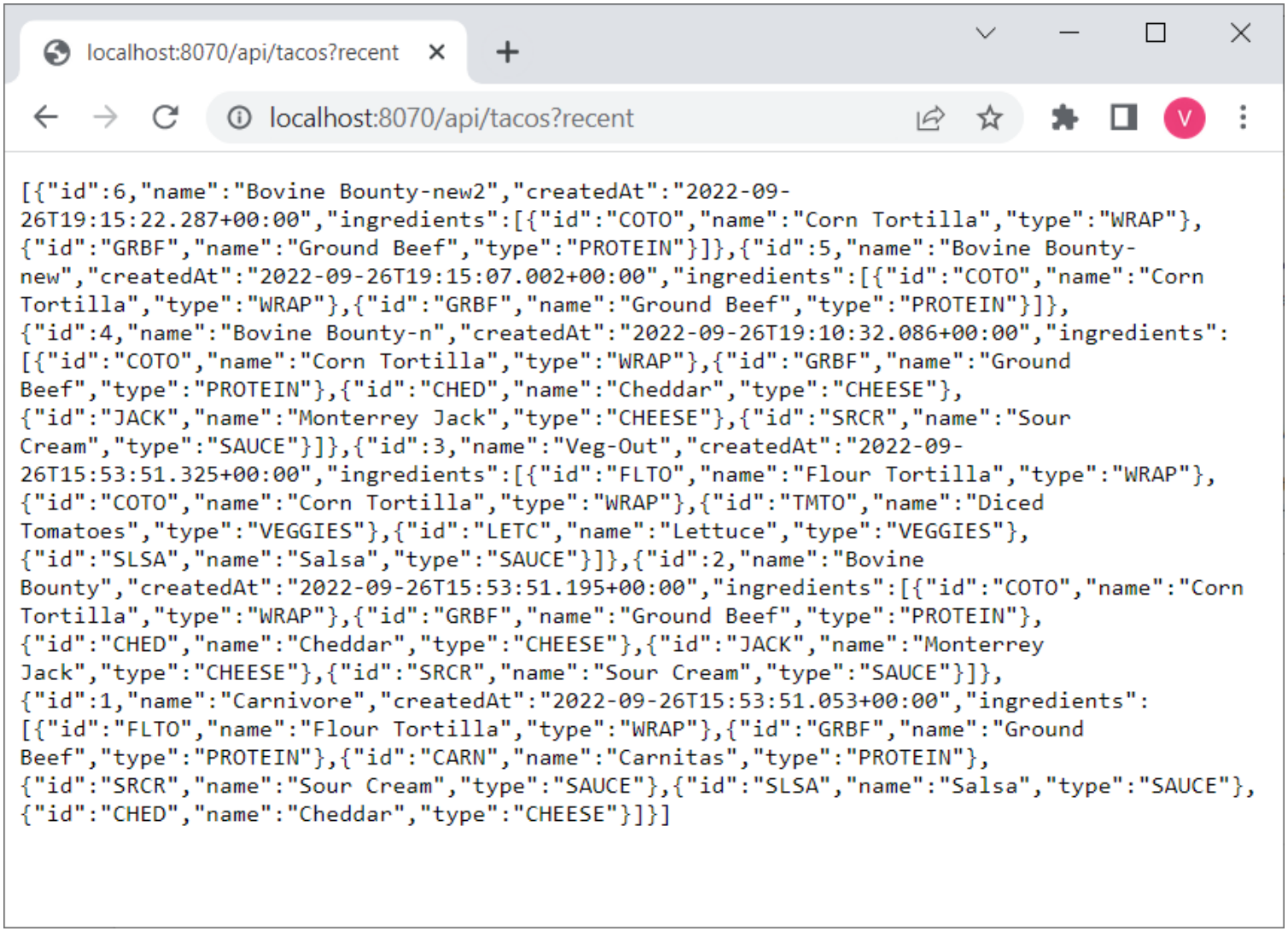
- Relationships of taco-api-db



TacoController.java

```java
package com.javaweb.controllers;
import java.util.List;

@RestController
@RequestMapping(path="/api/tacos", produces="application/json")
@CrossOrigin(origins="http://tacocloud:8080")
public class TacoController {
  @Autowired private TacoRepository tacoRepo;

  @GetMapping(params="recent")
  public Iterable<Taco> recentTacos() {
    PageRequest page = PageRequest.of(0, 12, Sort.by("createdAt").descending());
    return tacoRepo.findAll(page).getContent();
  }

  @PostMapping(consumes="application/json")
  @ResponseStatus(HttpStatus.CREATED)
  public Taco postTaco(@RequestBody Taco taco) {
      Taco newTaco = new Taco();
      newTaco.setName(taco.getName());
      List<Ingredient> ingredients = taco.getIngredients();
      newTaco.setIngredients(ingredients);

      return tacoRepo.save(newTaco);
  }

  @GetMapping("/{id}")
  public Optional<Taco> tacoById(@PathVariable("id") Long id) {
    return tacoRepo.findById(id);
  }

}
```

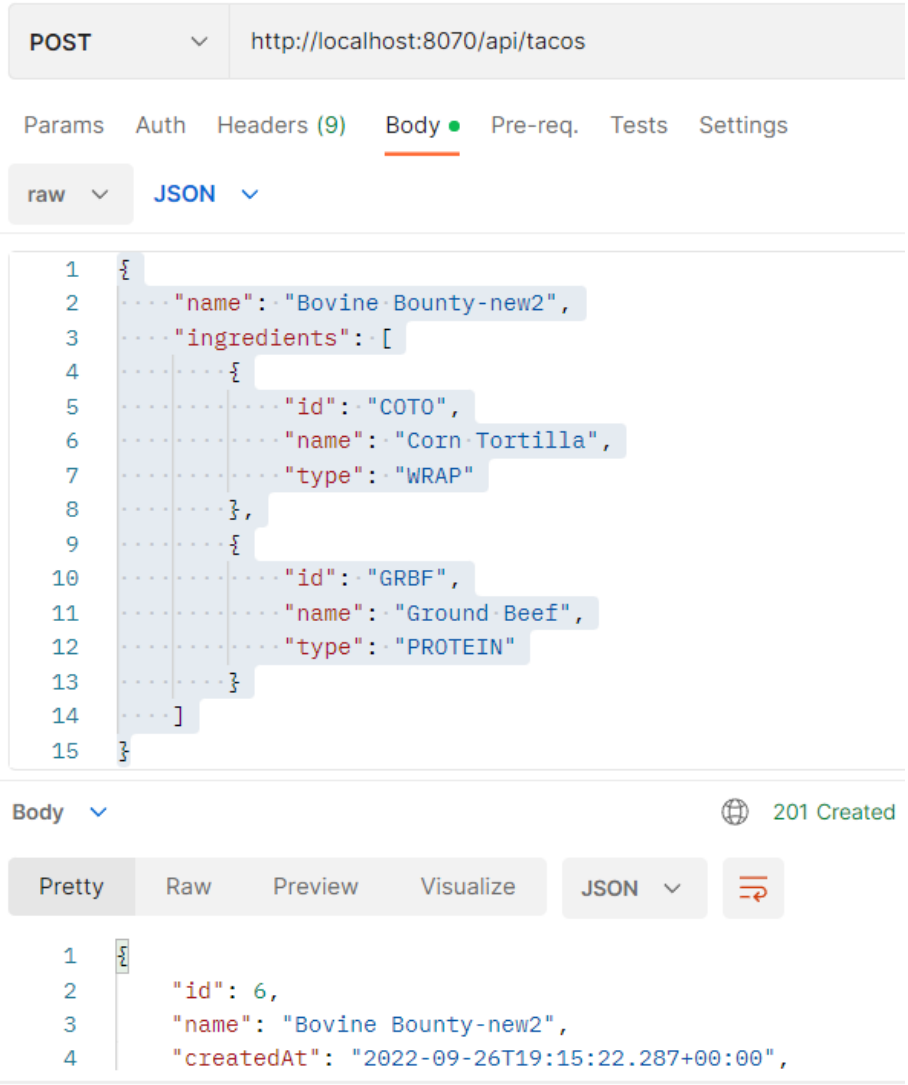
- **GET** http://localhost:8070/api/tacos?recent





```
[{"id":6,"name":"Bovine Bounty-new2","createdAt":"2022-09-
26T19:15:22.287+00:00","ingredients":[{"id":"COTO","name":"Corn Tortilla","type":"WRAP"},
{"id":"GRBF","name":"Ground Beef","type":"PROTEIN"}]},{"id":5,"name":"Bovine Bounty-
new","createdAt":"2022-09-26T19:15:07.002+00:00","ingredients":[{"id":"COTO","name":"Corn
Tortilla","type":"WRAP"},{"id":"GRBF","name":"Ground Beef","type":"PROTEIN"}]},
{"id":4,"name":"Bovine Bounty-n","createdAt":"2022-09-26T19:10:32.086+00:00","ingredients":
[{"id":"COTO","name":"Corn Tortilla","type":"WRAP"},{"id":"GRBF","name":"Ground
Beef","type":"PROTEIN"},{"id":"CHED","name":"Cheddar","type":"CHEESE"},
{"id":"JACK","name":"Monterrey Jack","type":"CHEESE"},{"id":"SRCR","name":"Sour
Cream","type":"SAUCE"}]},{"id":3,"name":"Veg-Out","createdAt":"2022-09-
26T15:53:51.325+00:00","ingredients":[{"id":"FLTO","name":"Flour Tortilla","type":"WRAP"},
{"id":"COTO","name":"Corn Tortilla","type":"WRAP"},{"id":"TMTO","name":"Diced
Tomatoes","type":"VEGGIES"},{"id":"LETC","name":"Lettuce","type":"VEGGIES"},
{"id":"SLSA","name":"Salsa","type":"SAUCE"}]},{"id":2,"name":"Bovine
Bounty","createdAt":"2022-09-26T15:53:51.195+00:00","ingredients":[{"id":"COTO","name":"Corn
Tortilla","type":"WRAP"},{"id":"GRBF","name":"Ground Beef","type":"PROTEIN"},
{"id":"CHED","name":"Cheddar","type":"CHEESE"},{"id":"JACK","name":"Monterrey
Jack","type":"CHEESE"},{"id":"SRCR","name":"Sour Cream","type":"SAUCE"}]},
{"id":1,"name":"Carnivore","createdAt":"2022-09-26T15:53:51.053+00:00","ingredients":
[{"id":"FLTO","name":"Flour Tortilla","type":"WRAP"},{"id":"GRBF","name":"Ground
Beef","type":"PROTEIN"},{"id":"CARN","name":"Carnitas","type":"PROTEIN"},
{"id":"SRCR","name":"Sour Cream","type":"SAUCE"},{"id":"SLSA","name":"Salsa","type":"SAUCE"},
{"id":"CHED","name":"Cheddar","type":"CHEESE"}]}]
```

- **POST** http://localhost:8070/api/tacos



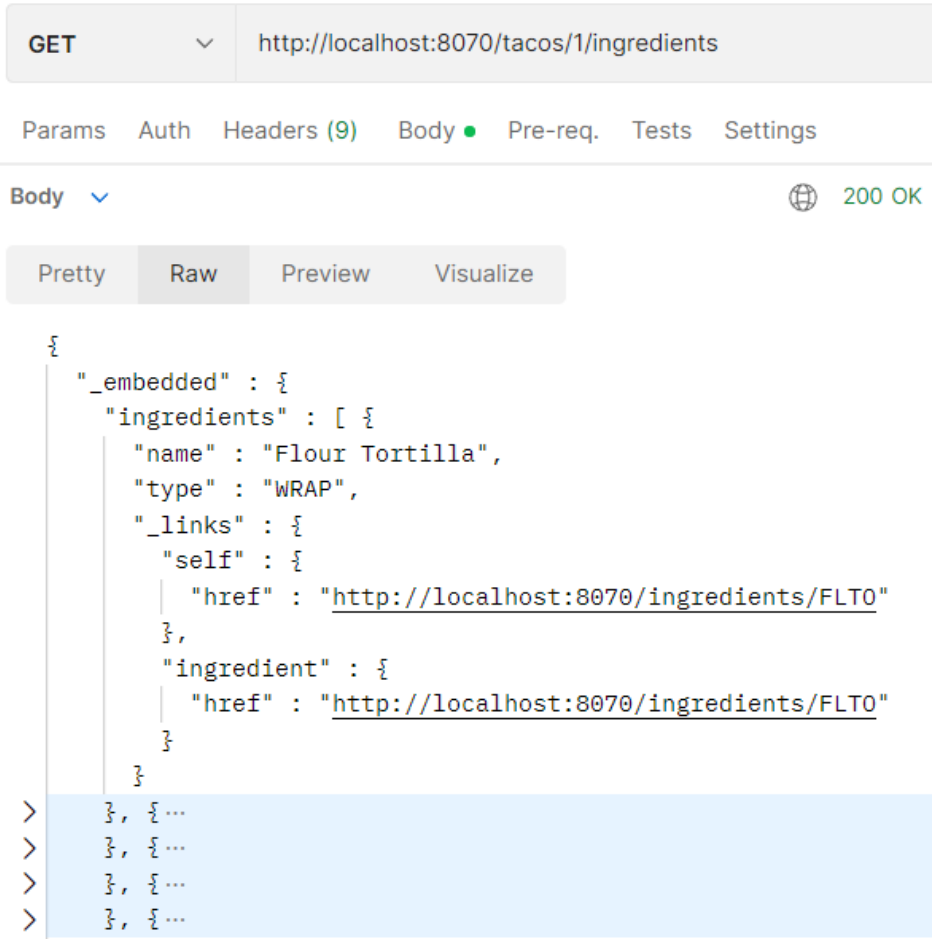

{
"name": "Bovine Bounty-new2",
"ingredients": [
{
"id": "COTO",
"name": "Corn Tortilla",
"type": "WRAP"
},
{
"id": "GRBF",
"name": "Ground Beef",
"type": "PROTEIN"
}
]
}

- **GET** http://localhost:8070/tacos



```
{
  "_embedded" : {
    "tacos" : [ {
      "name" : "Carnivore",
      "createdAt" : "2022-09-26T15:53:51.053+00:00",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8070/tacos/1"
        },
        "taco" : {
          "href" : "http://localhost:8070/tacos/1"
        },
        "ingredients" : {
          "href" : "http://localhost:8070/tacos/1/ingredients"
        }
      }
    }, {
      "name" : "Bovine Bounty",
      "createdAt" : "2022-09-26T15:53:51.195+00:00",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8070/tacos/2"
        },
        "taco" : {
          "href" : "http://localhost:8070/tacos/2"
        },
        "ingredients" : {
          "href" : "http://localhost:8070/tacos/2/ingredients"
        }
      }
    }, {
      "name" : "Veg-Out",
      "createdAt" : "2022-09-26T15:53:51.325+00:00",
      "_links" : {
```


- **GET** http://localhost:8070/tacos/1/ingredients

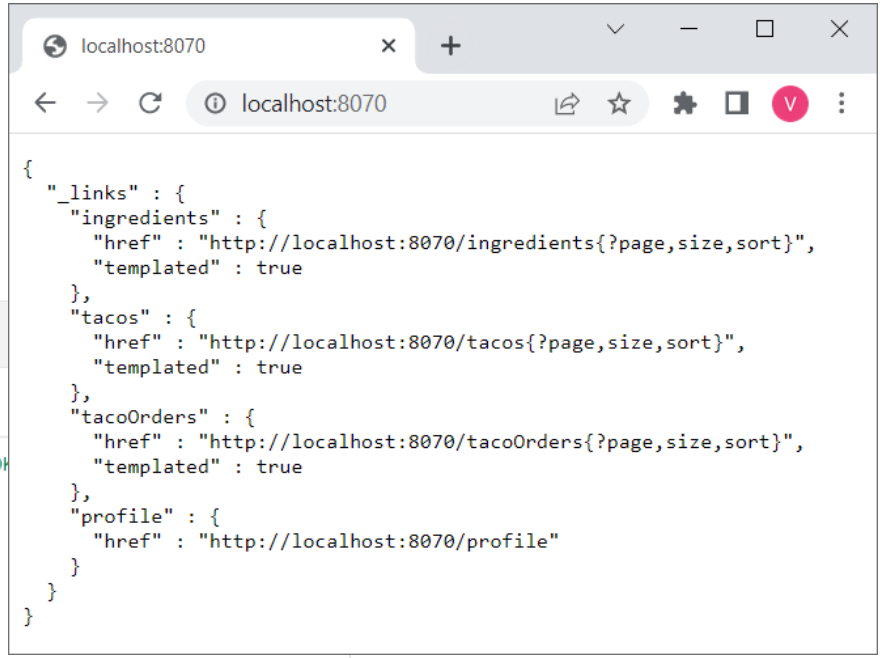




```
{
  "_embedded" : {
    "ingredients" : [ {
      "name" : "Flour Tortilla",
      "type" : "WRAP",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8070/ingredients/FLTO"
        },
        "ingredient" : {
          "href" : "http://localhost:8070/ingredients/FLTO"
        }
      }
    }, {…
    }, {…
    }, {…
    }, {…
```


**Using Spring Data REST**
**REF: https://spring.io/guides/tutorials/rest/**

- **GET** http://locahost:8070




- **Add Spring Data Rest base path**
  - **spring.data.rest.basePath=/data-api**
  - **GET** http://localhost:8070/data-api

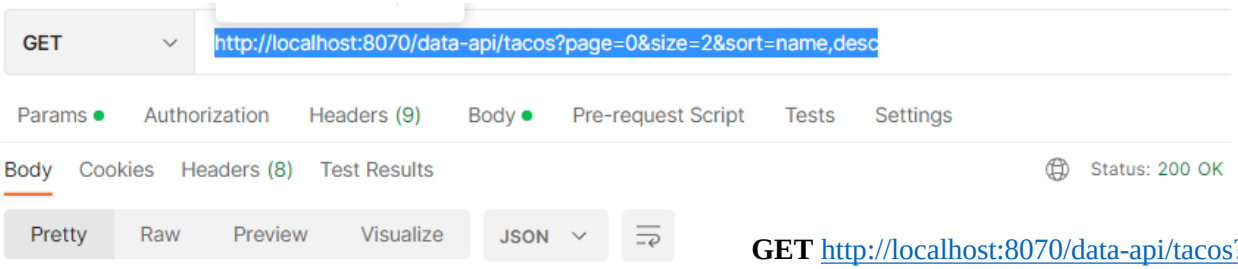
GET http://localhost:8070/data-api

Params Auth Headers (9) Body Pre-req. Tests Settings

Body 200 OK

Pretty Raw Preview Visualize JSON

```
{
    "_links": {
        "ingredients": {
            "href": "http://localhost:8070/data-api/ingredients{?page,size,sort}",
            "templated": true
        },
        "tacos": {
            "href": "http://localhost:8070/data-api/tacos{?page,size,sort}",
            "templated": true
        },
        "tacoOrders": {
            "href": "http://localhost:8070/data-api/tacoOrders{?page,size,sort}",
            "templated": true
        },
        "profile": {
            "href": "http://localhost:8070/data-api/profile"
        }
    }
}
```



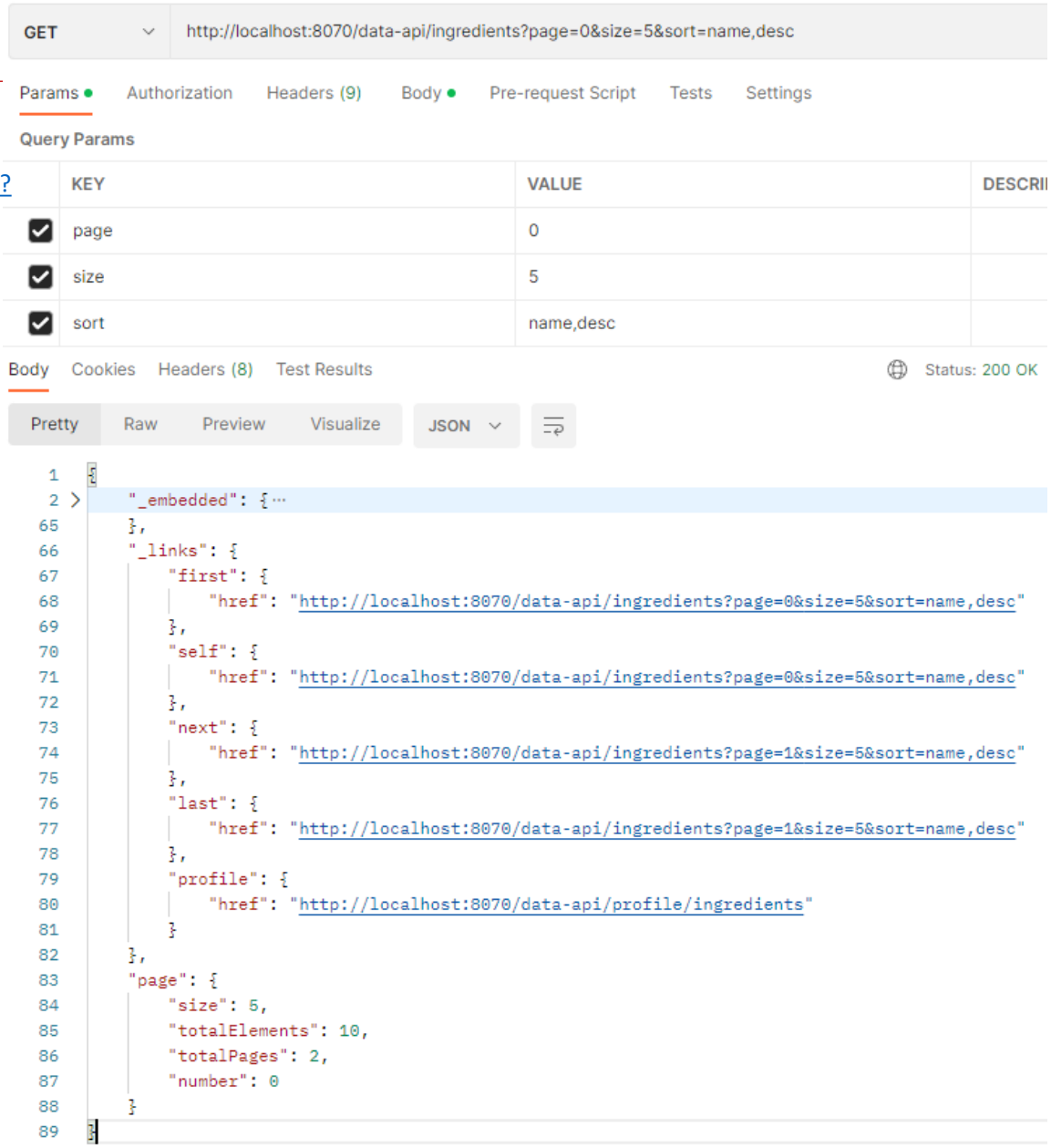

**GET** http://localhost:8070/data-api/tacos?page=0&size=2&sort=name,desc


```
{
    "_embedded": {…
    },
    "_links": {
        "first": {
            "href": "http://localhost:8070/data-api/tacos?page=0&size=2&sort=name,desc"
        },
        "self": {
            "href": "http://localhost:8070/data-api/tacos?page=0&size=2&sort=name,desc"
        },
        "next": {
            "href": "http://localhost:8070/data-api/tacos?page=1&size=2&sort=name,desc"
        },
        "last": {
            "href": "http://localhost:8070/data-api/tacos?page=2&size=2&sort=name,desc"
        },
        "profile": {
            "href": "http://localhost:8070/data-api/profile/tacos"
        },
        "recents": {
            "href": "http://localhost:8070/data-api/tacos/recent"
        }
    },
    "page": {
        "size": 2,
        "totalElements": 6,
        "totalPages": 3,
        "number": 0
    }
}
```

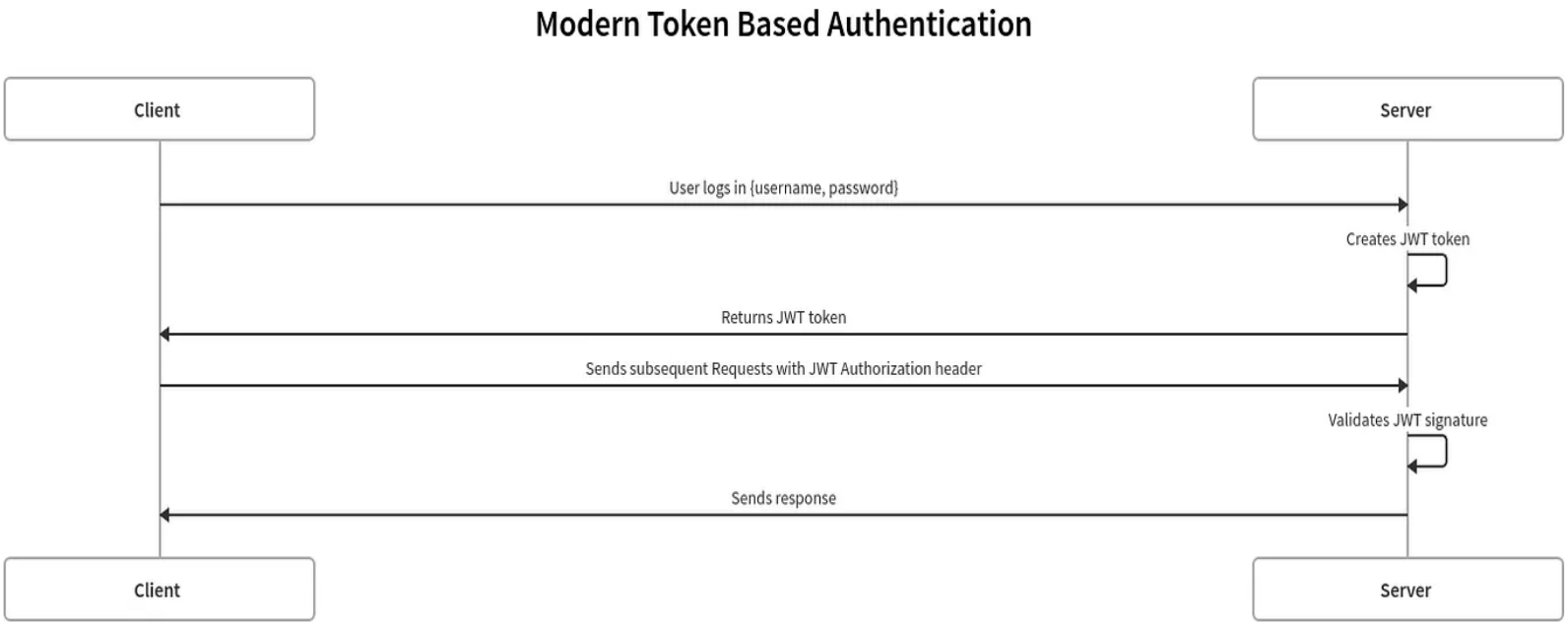
# ...tacocloud-api...

- **GET**

http://localhost:8070/data-api/ingredients?page=0&size=5&sort=name,desc



- **Stateless Authentication:** using tokens (e.g., JWT) that are gaining popularity, especially in modern Microservices and distributed systems.
- **Token-Based Authentication**
  - Issuing a token (e.g., JWT) upon successful authentication.
  - The token is sent to the server with each request for authorization.
  - It is Stateless and scalable.




Source: https://medium.com/@minadev

## JWT Authentication

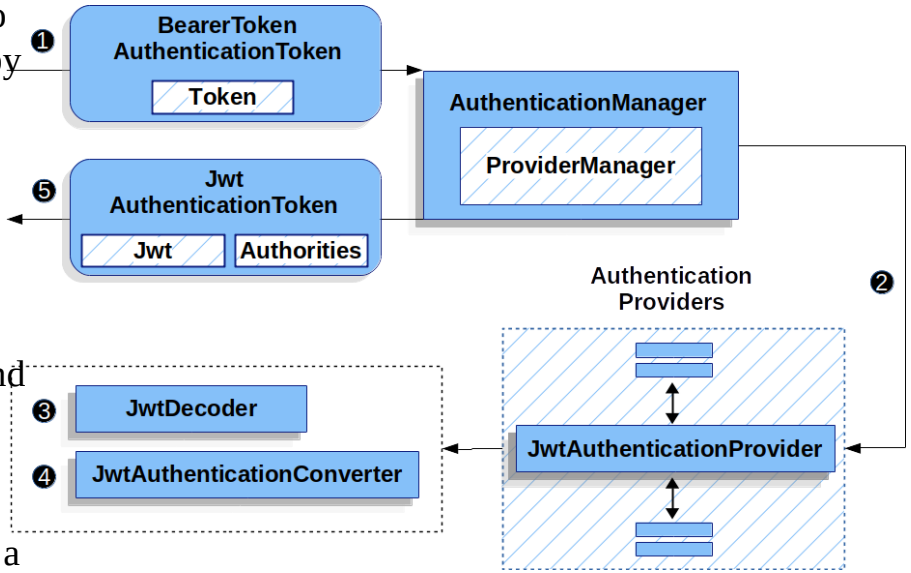
(1) The authentication Filter from Reading the Bearer Token passes a BearerTokenAuthenticationToken to the AuthenticationManager which is implemented by ProviderManager.

(2) The ProviderManager is configured to use an AuthenticationProvider of type JwtAuthenticationProvider.

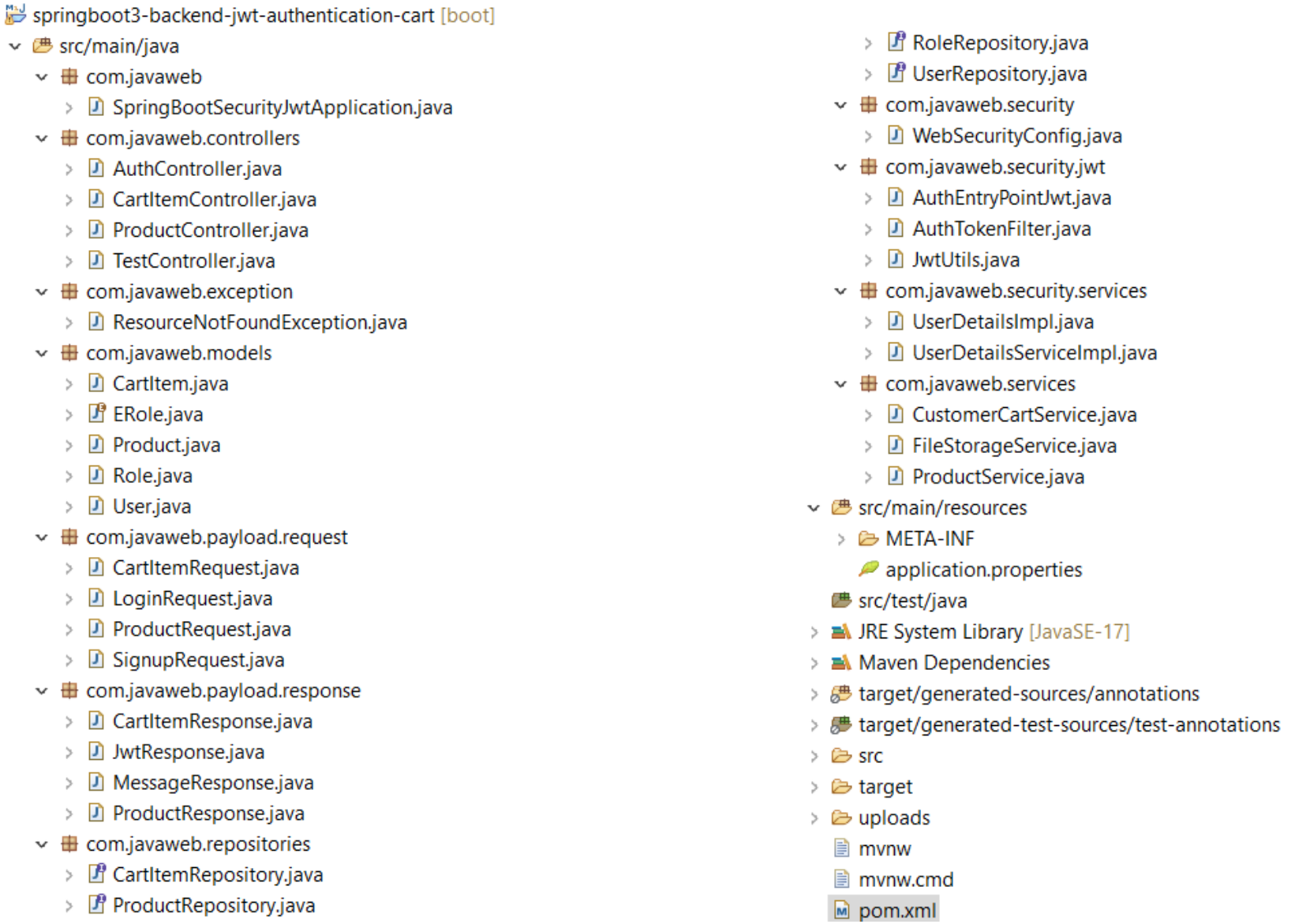
(3) JwtAuthenticationProvider decodes, verifies, and validates the Jwt using a JwtDecoder.

(4) JwtAuthenticationProvider then uses the JwtAuthenticationConverter to convert the Jwt into a Collection of granted authorities.

(5) When authentication is successful, the Authentication that is returned is of type JwtAuthenticationToken and has a principal that is the Jwt returned by the configured JwtDecoder. Ultimately, the returned JwtAuthenticationToken will be set on the SecurityContextHolder by the authentication Filter.



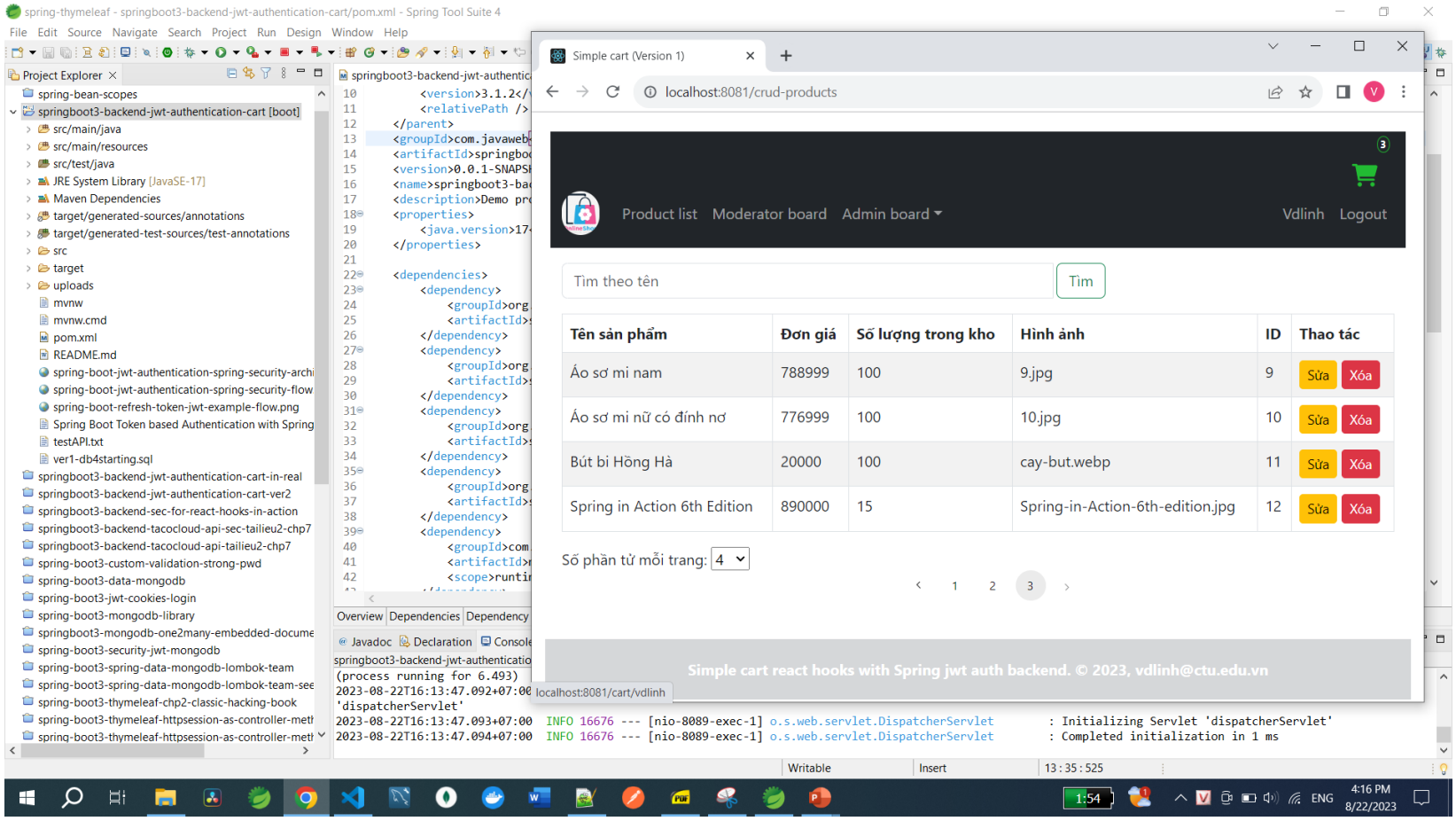

REF: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html


springboot3-backend-jwt-authentication-cart [boot]
src/main/java
com.javaweb
SpringBootSecurityJwtApplication.java
com.javaweb.controllers
AuthController.java
CartItemController.java
ProductController.java
TestController.java
com.javaweb.exception
ResourceNotFoundException.java
com.javaweb.models
CartItem.java
ERole.java
Product.java
Role.java
User.java
com.javaweb.payload.request
CartItemRequest.java
LoginRequest.java
ProductRequest.java
SignupRequest.java
com.javaweb.payload.response
CartItemResponse.java
JwtResponse.java
MessageResponse.java
ProductResponse.java
com.javaweb.repositories
CartItemRepository.java
ProductRepository.java
RoleRepository.java
UserRepository.java
com.javaweb.security
WebSecurityConfig.java
com.javaweb.security.jwt
AuthEntryPointJwt.java
AuthTokenFilter.java
JwtUtils.java
com.javaweb.security.services
UserDetailsImpl.java
UserDetailsServiceImpl.java
com.javaweb.services
CustomerCartService.java
FileStorageService.java
ProductService.java
src/main/resources
META-INF
application.properties
src/test/java
JRE System Library [JavaSE-17]
Maven Dependencies
target/generated-sources/annotations
target/generated-test-sources/test-annotations
src
target
uploads
mvnw
mvnw.cmd
pom.xml

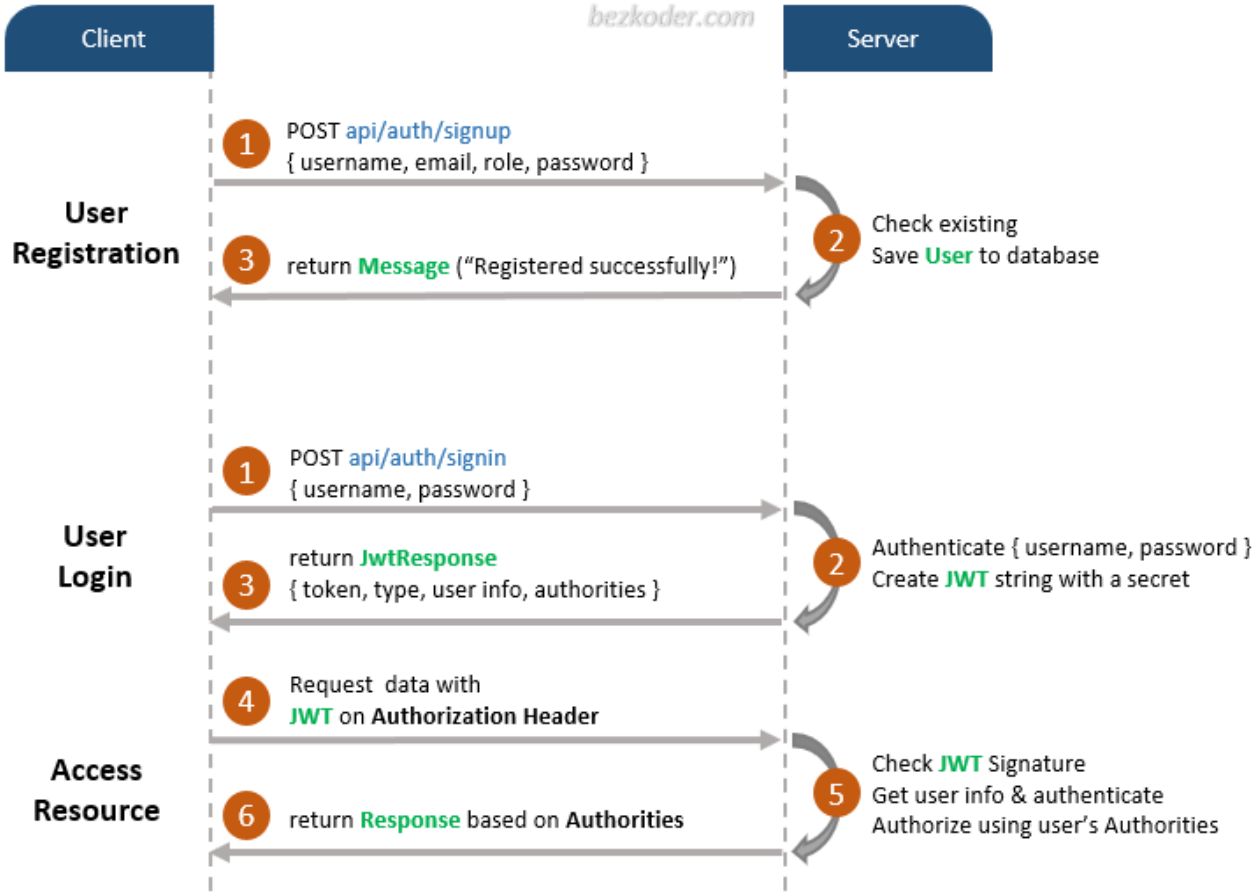
```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-
jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
security</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
validation</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt-api</artifactId>
<version>0.11.5</version>
</dependency>
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt-impl</artifactId>
<version>0.11.5</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt-jackson</artifactId>
<version>0.11.5</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>2.13.0</version>
</dependency>
<dependency>
<groupId>org.springframework.security</group
Id>
<artifactId>spring-security-
test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

# Project 3 ◆ Backend Rest API Security with Spring Boot ◆ Chap 5

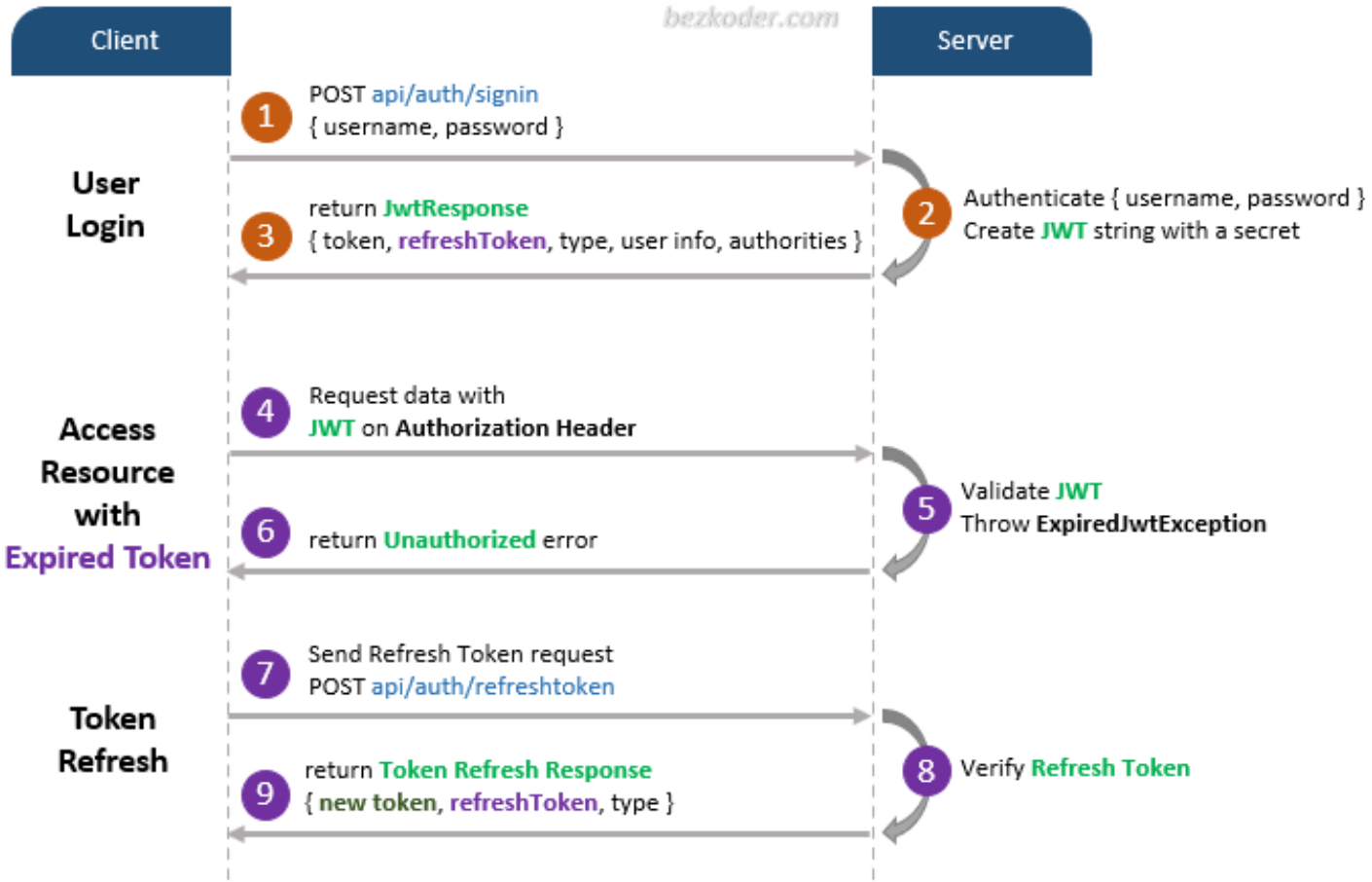
- **Demo full-stack simple cart web app (version 1)**
  - **Back-end: springboot3-backend-jwt-authentication-cart**
  - **Front-end: simple-cart-react-hooks-jwt-auth-ver1**



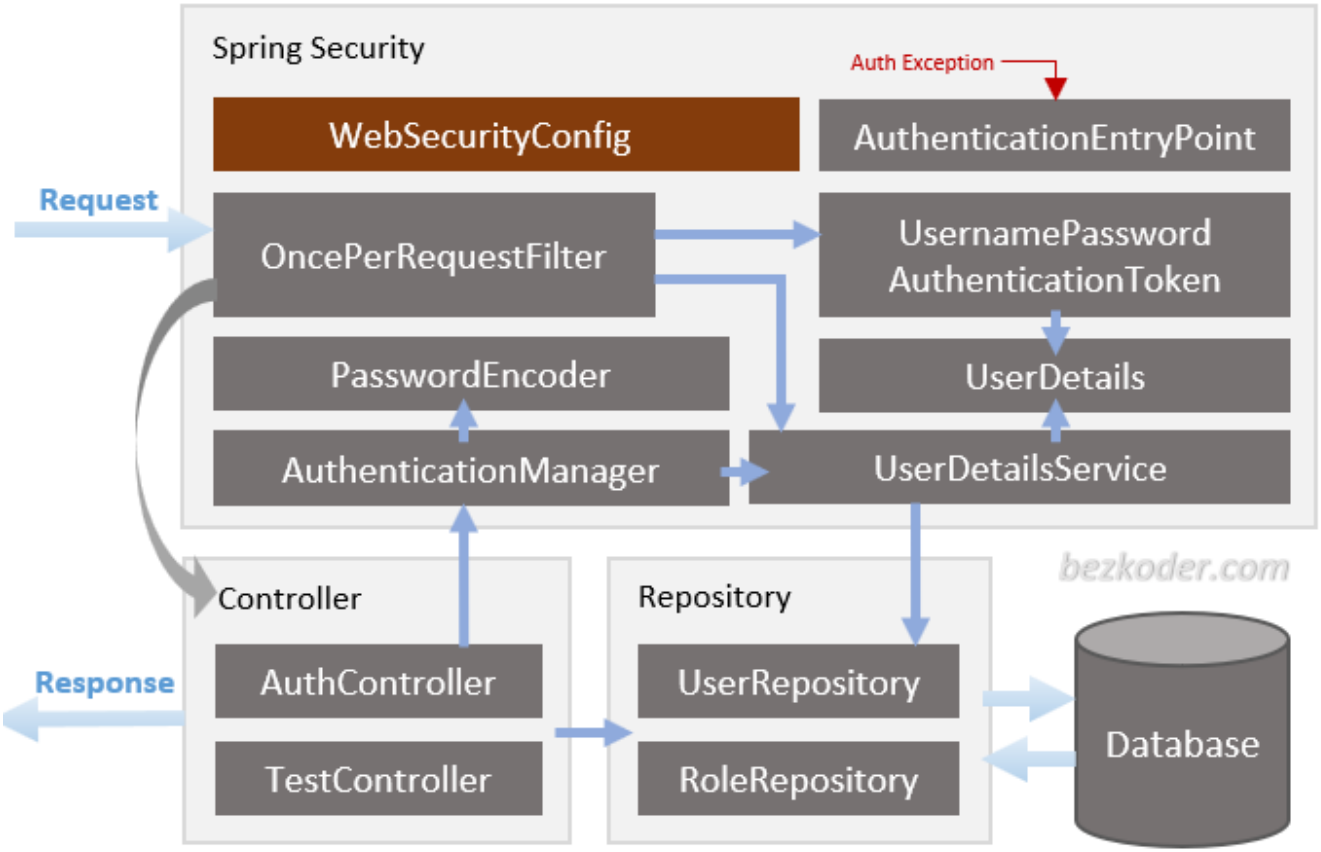
- **Spring Boot Signup & Login with JWT Authentication Flow**
  - The diagram shows flow of how we implement User Registration, User Login and Authorization process.



- A legal JWT must be added to HTTP Authorization Header if Client accesses protected resources. You will need to implement **Refresh Token**: https://www.bezkoder.com/spring-boot-refresh-token-jwt/

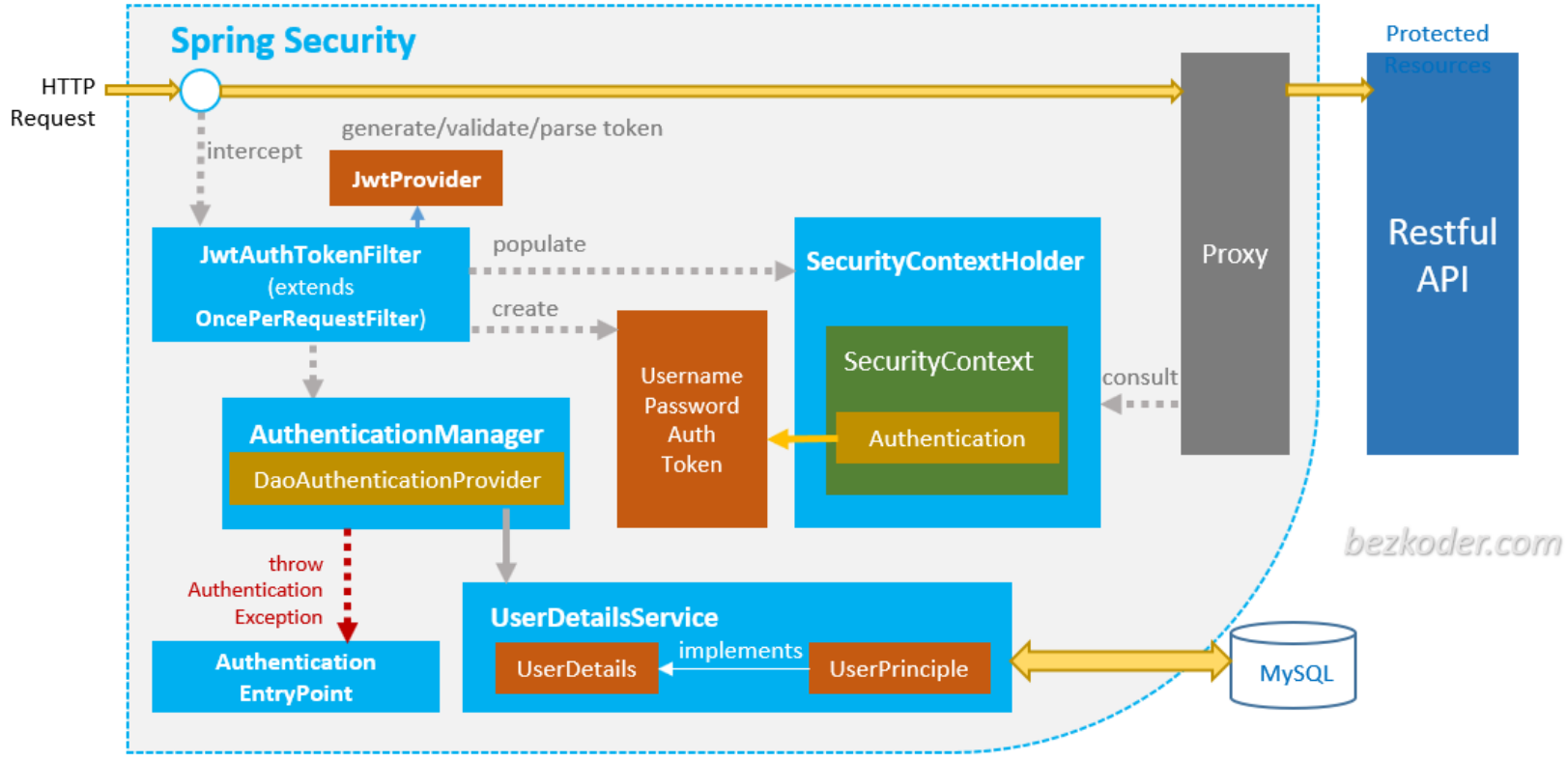


- **Spring Boot Server Architecture with Spring Security**
  - Refs: https://www.bezkoder.com/spring-boot-jwt-authentication/
    https://www.bezkoder.com/spring-boot-jwt-mysql-spring-security-architecture/
    https://www.bezkoder.com/react-hooks-jwt-auth/



- Spring Security
  - **WebSecurityConfig** is the crux of our security implementation. It configures cors, csrf, session management, rules for protected resources.
  - **UserDetailsService** interface has a method to load User by username and returns a UserDetails object that Spring Security can use for authentication and validation.
  - **UserDetails** contains necessary information (such as: username, password, authorities) to build an Authentication object.
  - **UsernamePasswordAuthenticationToken** gets {username, password} from login Request, AuthenticationManager will use it to authenticate a login account.
  - **AuthenticationManager** has **a DaoAuthenticationProvider** (with help of UserDetailsService & PasswordEncoder) to validate UsernamePasswordAuthenticationToken object. If successful, AuthenticationManager returns a fully populated Authentication object (including granted authorities).
  - **OncePerRequestFilter** makes a single execution for each request to our API. It provides a doFilterInternal() method that we will implement parsing & validating JWT, loading User details (using UserDetailsService), checking Authorizaion (using UsernamePasswordAuthenticationToken).
  - **AuthenticationEntryPoint** will catch authentication error.
- Controller receives and handles request after it was filtered by OncePerRequestFilter
  - AuthController handles signup/login requests
  - TestController has accessing protected resource methods with role based validations.

- Spring Security JWT in Spring Boot 3




Spring Security Authentication process: receive HTTP request, filter, authenticate, store Authentication data, generate token, get User details, authorize, handle exception…

WebSecurityConfig.java

```java
package com.javaweb.security;
import org.springframework.beans.factory.annotation.Autowired;

/*
 * @author Vu Duy Linh, vlinh@ctu.edu.vn
 *
 * References:
 *    https://www.bezkoder.com/spring-boot-jwt-authentication
 *    https://www.toptal.com/spring/spring-security-tutorial
 * */

@Configuration
@EnableMethodSecurity
public class WebSecurityConfig {
    @Autowired UserDetailsServiceImpl userDetailsService;

    @Autowired private AuthEntryPointJwt authEntryPointJwt;

    @Bean
    AuthTokenFilter authenticationJwtTokenFilter() {
        return new AuthTokenFilter();
    }

    @Bean
    DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Bean
    AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }
```

```
56
57      @Bean
58      PasswordEncoder passwordEncoder() {
59          return new BCryptPasswordEncoder();
60      }
61
62      @Bean
63      SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
64          http
65                  // 1. [Enable CORS and] disable CSRF
66                  .csrf(csrf -> csrf.disable())
67
68                  // 2. Set unauthorized requests exception handler
69                  .exceptionHandling(exception -> exception.authenticationEntryPoint(authEntryPointJwt))
70
71                  // 3. Set session management to stateless
72                  .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
73
74                  // 4. Set permissions on endpoints
75                  .authorizeHttpRequests(auth -> auth
76                          // Our public endpoints
77                          .requestMatchers("/api/auth/**").permitAll().requestMatchers("/api/test/**").permitAll()
78                          .requestMatchers("/api/product/**").permitAll().requestMatchers("/api/cart-item/**").permitAll()
79
80                          // Our private endpoints
81                          .anyRequest().authenticated());
82
83          http.authenticationProvider(authenticationProvider());
84
85          // 5. Add JWT token filter
86          http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
87
88          return http.build();
89      }
90
91  }
```

CartItem.java

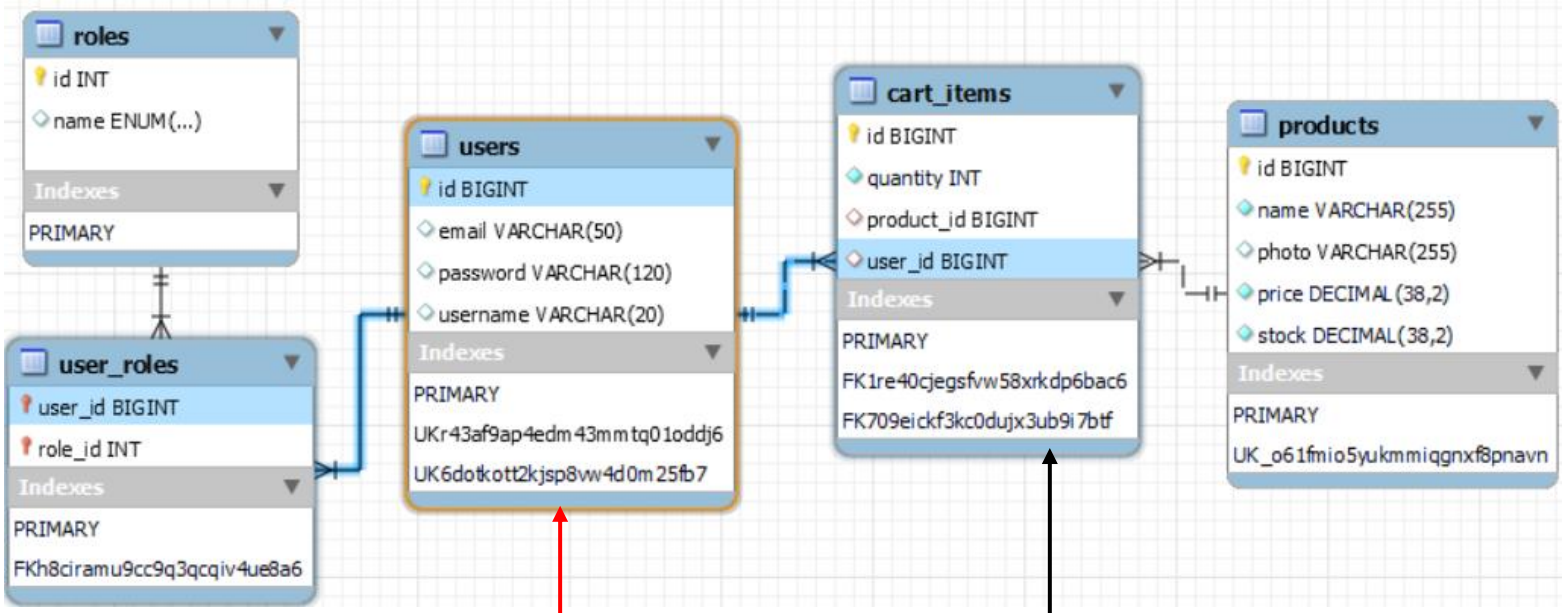
```java
package com.javaweb.springjwt.models;

import jakarta.persistence.Column;

@Entity
@Table(name = "cart_items")
public class CartItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "product_id", referencedColumnName = "id")
    private Product product;

    @Column(name = "quantity", nullable = false)
    @DecimalMin(value = "0", message = "*Quantity has to be non negative number")
    private Integer quantity;

    public CartItem() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

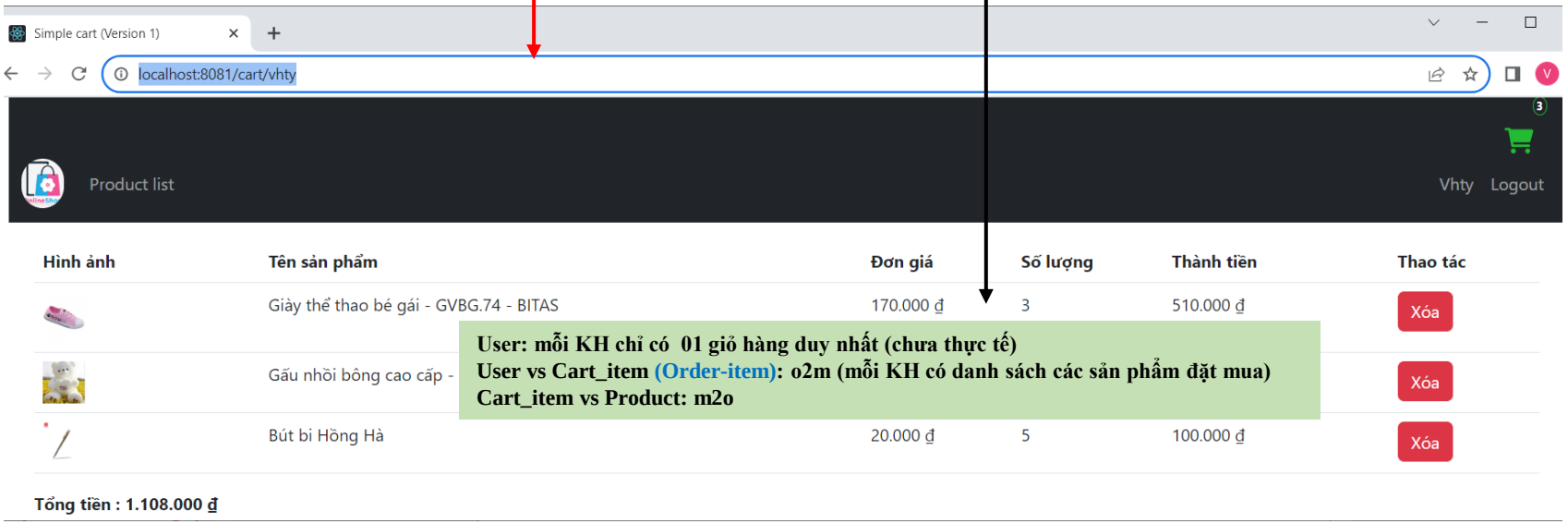
User.java

```
 2
 3 import java.util.ArrayList;
23
24 @Entity
25 @Table(name = "users", uniqueConstraints = { @UniqueConstraint(columnNames = "username"),
26         @UniqueConstraint(columnNames = "email") })
27 public class User {
28     @Id
29     @GeneratedValue(strategy = GenerationType.IDENTITY)
30     private Long id;
31
32     @NotBlank @Size(max = 20)
33     private String username;
34
35     @NotBlank @Size(max = 50) @Email
36     private String email;
37
38     @NotBlank @Size(max = 120)
39     private String password;
40
41     @ManyToMany(fetch = FetchType.LAZY)
42     @JoinTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"),
43                                     inverseJoinColumns = @JoinColumn(name = "role_id"))
44     private Set<Role> roles = new HashSet<>();
45
46     @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
47     private List<CartItem> cartItems = new ArrayList<>();
48
49     public List<CartItem> getCarts() {
50         return cartItems;
51     }
52
53     public void setCarts(List<CartItem> cartItems) {
54         this.cartItems = cartItems;
55     }
```

Product.java

```java
package com.javaweb.springjwt.models;

import org.hibernate.validator.constraints.Length;

@Entity
@Table(name = "products")
public class Product {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name", nullable = false, unique = true)
    @Length(min = 3, message = "*Name must have at least 5 characters")
    private String name;

    @Column(name = "price", nullable = false)
    @DecimalMin(value = "0.00", message = "*Price has to be non negative number")
    private BigDecimal price;

    @Column(name = "photo")
    private String photo;

    @Column(name = "stock", nullable = false)
    @DecimalMin(value = "0", message = "*Stock has to be non negative number")
    private BigDecimal stock;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```




**User: mỗi KH chỉ có 01 giỏ hàng duy nhất (chưa thực tế)**
**User vs Cart_item (Order-item): o2m (mỗi KH có danh sách các sản phẩm đặt mua)**
**Cart_item vs Product: m2o**

CustomerCartService.java

```java
package com.javaweb.services;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class CustomerCartService {
    @Autowired  private CartItemRepository cartItemRepository;
    @Autowired  private UserRepository userRepository;
    @Autowired  private ProductRepository productRepository;

    public ResponseEntity<?> getAllByUserId(Long userid) {
        List<CartItem> cartItems = cartItemRepository.findAllByUserId(userid);
        List<CartItemResponse> responses = new ArrayList<>();
        for (CartItem cartItem : cartItems) {
            ProductResponse productResponse = new ProductResponse(
                    cartItem.getProduct().getName(),
                    cartItem.getProduct().getPrice(),
                    cartItem.getProduct().getStock(),
                    cartItem.getProduct().getPhoto());

            CartItemResponse cartItemResponse = new CartItemResponse(
                    cartItem.getId(),
                    productResponse,
                    cartItem.getQuantity());

            responses.add(cartItemResponse);
        }
        return new ResponseEntity<>(responses, HttpStatus.OK);
    }

```

```
47      public ResponseEntity<?> create(Long userId, CartItemRequest request) {
48          CartItem availableCart = cartItemRepository
49                  .findAllByUserIdAndProductId(userId, request.getProductId());
50
51          if (availableCart != null) {
52              Integer quantity = availableCart.getQuantity();
53              quantity += request.getQuantity();
54              availableCart.setQuantity(quantity);
55
56              cartItemRepository.save(availableCart);
57          } else {
58              CartItem cartItem = new CartItem();
59              cartItem.setUser(userRepository.findById(userId).orElse(null));
60              cartItem.setProduct(productRepository.findById(request.getProductId()).orElse(null));
61              cartItem.setQuantity(request.getQuantity());
62
63              cartItemRepository.save(cartItem);
64          }
65
66          return new ResponseEntity<>(new MessageResponse("Logged-in User Cart is created"),
67                                      HttpStatus.CREATED);
68      }
69
70      @Transactional
71      public ResponseEntity<?> deteleById(Long id, Long userId) {
72          if (!cartItemRepository.existsById(id)) {
73              throw new RuntimeException("CartItem not existed.");
74          }
75
76          cartItemRepository.deleteByIdAndUserId(id, userId);
77          return new ResponseEntity<>(new MessageResponse("CartItem is deleted"), HttpStatus.OK);
78      }
79  }
```

CartItemController.java

```java
package com.javaweb.controllers;

import org.springframework.beans.factory.annotation.Autowired;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/cart-item")
public class CartItemController {

    @Autowired private CustomerCartService customerCartService;

    @GetMapping("/list")
//  @PreAuthorize("hasRole('USER')  or hasRole('ADMIN')")
    public ResponseEntity<?> getAllByUserId(
            @AuthenticationPrincipal UserDetailsImpl userDetails) {

        return customerCartService.getAllByUserId(userDetails.getId());
    }

    @PostMapping("/create")
    @PreAuthorize("hasRole('USER')  or hasRole('ADMIN')")
    public ResponseEntity<?> create(@AuthenticationPrincipal UserDetailsImpl userDetails,
                                    @Valid @RequestBody CartItemRequest request) {

        return customerCartService.create(userDetails.getId(), request);
    }

    @DeleteMapping("/delete/{id}")
    @PreAuthorize("hasRole('USER')  or hasRole('ADMIN')")
    public ResponseEntity<?> delete(@PathVariable Long id,
                                    @AuthenticationPrincipal UserDetailsImpl userDetails) {

        return customerCartService.deteleById(id, userDetails.getId());
    }
}
```

ProductService.java

```java
package com.javaweb.services;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class ProductService {
    @Autowired private ProductRepository productRepository;
    @Autowired private FileStorageService fileStorageService;

    public Map<String, Object> getAllPageableProducts(String name, int page, int size) {
        try {
            List<Product> products = new ArrayList<Product>();
            Pageable pagination = PageRequest.of(page, size);
            Page<Product> productPage;
            if (name == null) {
                productPage = productRepository.findAll(pagination);
            } else {
                productPage = productRepository.findByNameContaining(name, pagination);
            }

            products = productPage.getContent();

            Map<String, Object> response = new HashMap<String, Object>();
            response.put("products", products);
            response.put("totalPages", productPage.getTotalPages());

            return response;
        } catch (ResourceNotFoundException exc) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Resource Not Found", exc);
        }
    }

```

```
53    public ResponseEntity<?> create(ProductRequest request, MultipartFile file) {
54        int status = fileStorageService.save(file);
55
56        if (status != 0) {
57            Product product = new Product();
58            product.setName(request.getName().trim());
59            product.setPrice(request.getPrice());
60            product.setStock(request.getStock());
61            product.setPhoto(file.getOriginalFilename());
62            productRepository.save(product);
63
64        } else {
65            throw new RuntimeException("Upload failed");
66        }
67        return new ResponseEntity<>(new MessageResponse("Product is created"), HttpStatus.OK);
68    }
69
70    public ResponseEntity<?> updateProductById(ProductRequest request,
71                MultipartFile file, Long id) throws IOException {
72
73        Product product = productRepository.findById(id).orElseThrow(
74                                () -> new RuntimeException("Product not found"));
75
76        int status = fileStorageService.save(file);
77
78        if (status != 0) {
79            product.setPhoto(file.getOriginalFilename());
80        } else {
81            throw new RuntimeException("Upload failed");
82        }
```

```
83
84          if (!product.getName().equals(request.getName())) {
85              product.setName(request.getName());
86          }
87
88          if (!product.getPrice().equals(request.getPrice())) {
89              product.setPrice(request.getPrice());
90          }
91
92          if (!product.getStock().equals(request.getStock())) {
93              product.setStock(request.getStock());
94          }
95
96          productRepository.save(product);
97          return new ResponseEntity<>(new MessageResponse("Product is updated"), HttpStatus.OK);
98      }
99
100     public Product findProductById(Long id) {
101         Product product = productRepository.findById(id).orElseThrow(
102                                         () -> new RuntimeException("Product not found"));
103         return product;
104     }
105
106     public ResponseEntity<?> deleteProductById(Long id) {
107         if (!productRepository.existsById(id)) {
108             throw new RuntimeException("Product not found");
109         }
110
111         productRepository.deleteById(id);
112         return new ResponseEntity<>(new MessageResponse("Product is deleted"), HttpStatus.OK);
113     }
114 }
```

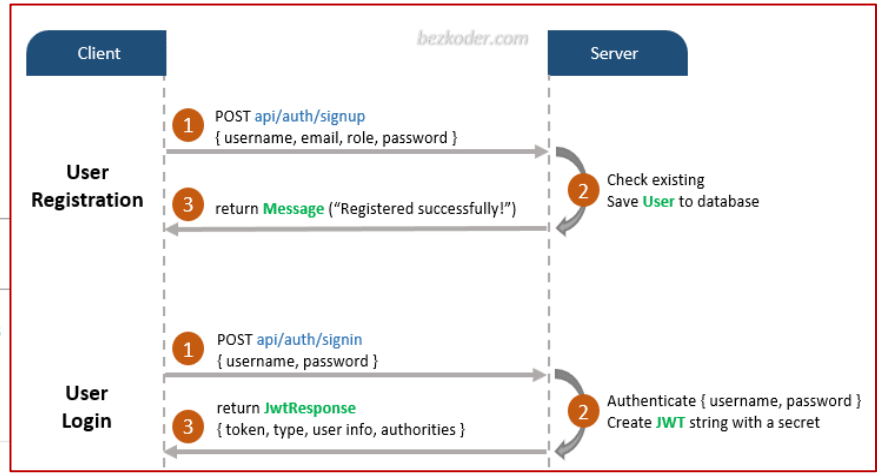
ProductController.java

```java
package com.javaweb.controllers;

import java.io.IOException;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/product")
public class ProductController {

    @Autowired  private ProductService productService;
    @Autowired  private FileStorageService fileStorageService;

    //Some CRUD's end-points for Admin role
    @GetMapping("/products")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public Map<String, Object> fetchPaginationAllProducts(
            @RequestParam(value = "name", required = false) String name,
            @RequestParam(value = "page", defaultValue = "0") int page,
            @RequestParam(value = "size", defaultValue = "4") int size) {

        return productService.getAllPageableProducts(name, page, size);
    }

    @PostMapping("/products")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ResponseEntity<?> createProduct(@Valid @RequestPart ProductRequest request,
            @RequestPart MultipartFile file) {

        return productService.create(request, file);
    }
```

**Define some end-points/api(s) with @RestController, @PreAuthorize**

```
56
57      @PutMapping("/products/{id}")
58      @PreAuthorize("hasRole('ROLE_ADMIN')")
59      public ResponseEntity<?> updateProductById(@Valid @RequestPart ProductRequest request,
60              @RequestPart MultipartFile file, @PathVariable Long id) throws IOException {
61
62          return productService.updateProductById(request, file, id);
63      }
64
65      @GetMapping("/images/{filename}")
66      public ResponseEntity<?> getImage(@PathVariable("filename") String filename) {
67          Resource resource = fileStorageService.load(filename);
68          return ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(resource);
69      }
70
71      @DeleteMapping("/products/{id}")
72      @PreAuthorize("hasRole('ROLE_ADMIN')")
73      public Boolean deleteProduct(@PathVariable("id") Long id) {
74          productService.deleteProductById(id);
75          return true;
76      }
77
78      @GetMapping("/products/{id}")
79      @PreAuthorize("hasRole('ROLE_ADMIN')")
80      public Product findById(@PathVariable("id") Long id) {
81          return productService.findProductById(id);
82      }
83  }
```

- A legal JWT must be added to HTTP Authorization Header if Client accesses protected resources.




**User login: ① ⇨ ② ⇨ ③**

POST ① http://localhost:8089/api/auth/signin

Params Auth ● Headers (8) Body ● Pre-req. Tests Settings

raw JSON

② 

```
{
    "username": "vdlinh",
    "password": "1234567"
}
```

Body Cookies Headers (14) Test Results 200 OK 435 ms

Pretty Raw Preview Visualize JSON

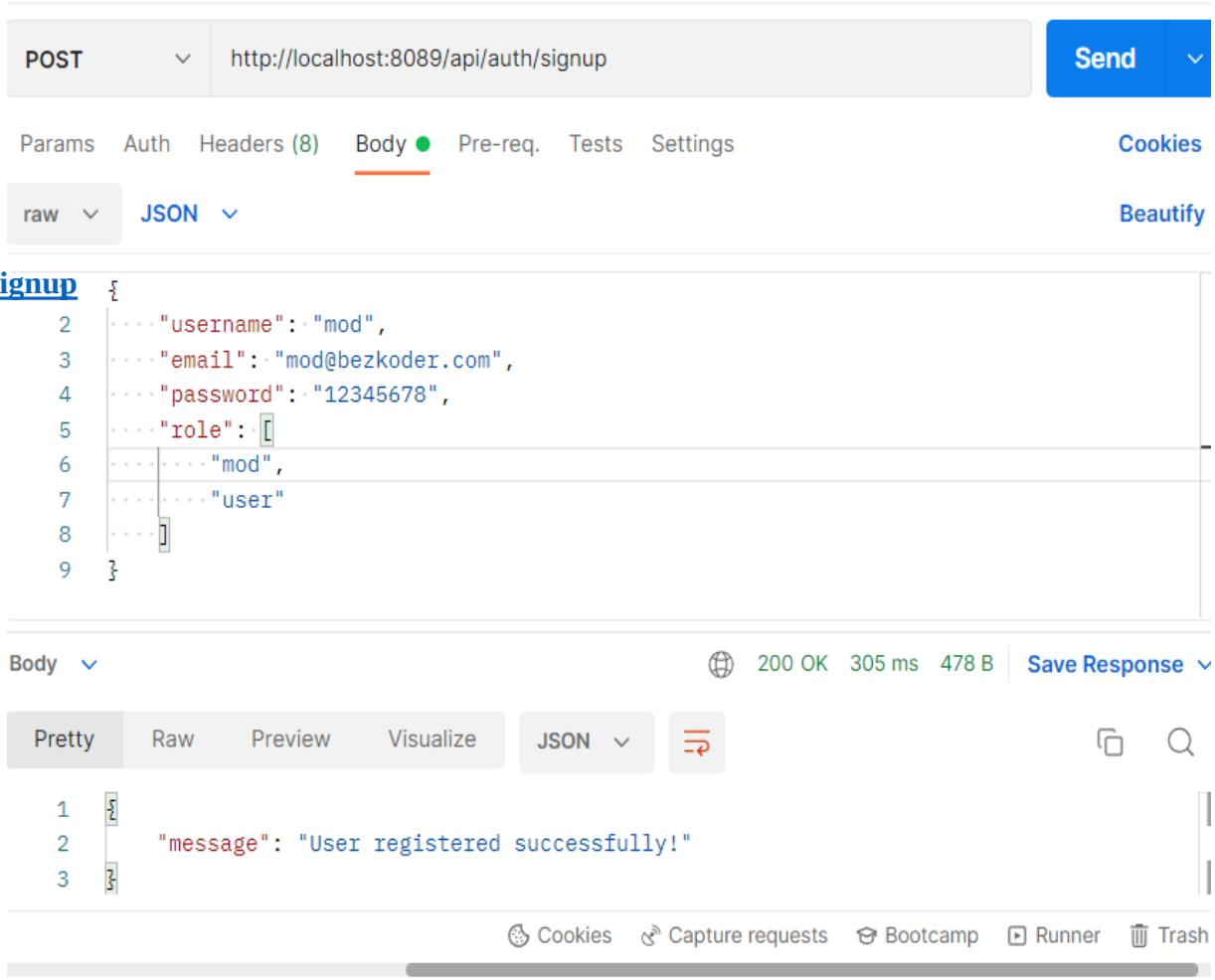
③

```
{
    "id": 1,
    "username": "vdlinh",
    "email": "vdlinh@ctu.edu.vn",
    "roles": [
        "ROLE_MODERATOR",
        "ROLE_ADMIN"
    ],
    "accessToken": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJ2ZGxpbmgiLCJpYXQiOjE2OTI4ODA2MTEsImV4cCI6MTY5Mjk2NzAxMX0.
        f4rcWRUQWBTKviS5dmOeUtVNj9v_M9gDFfsKkapThrA",
    "tokenType": "Bearer"
}
```

**Register new account**

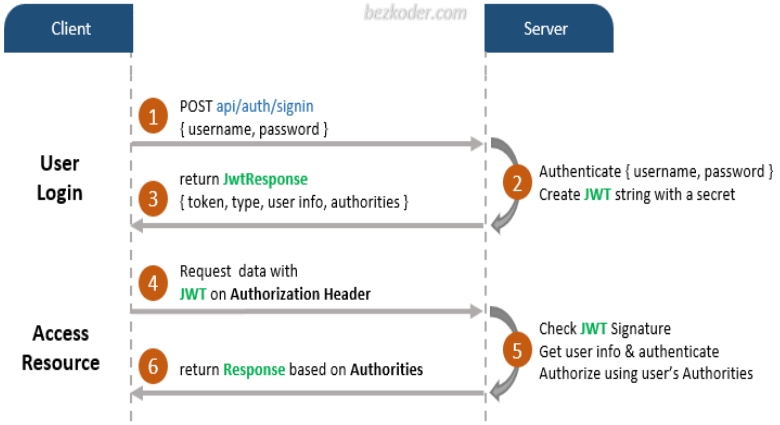
**POST**
**http://localhost:8089/api/auth/signup**

```
{
  "username": "mod",
  "email": "mod@bezkoder",
  "password": "12345678",
  "role":[
    "mod",
    "user"
  ]
}
```




**https://jwt.io/**

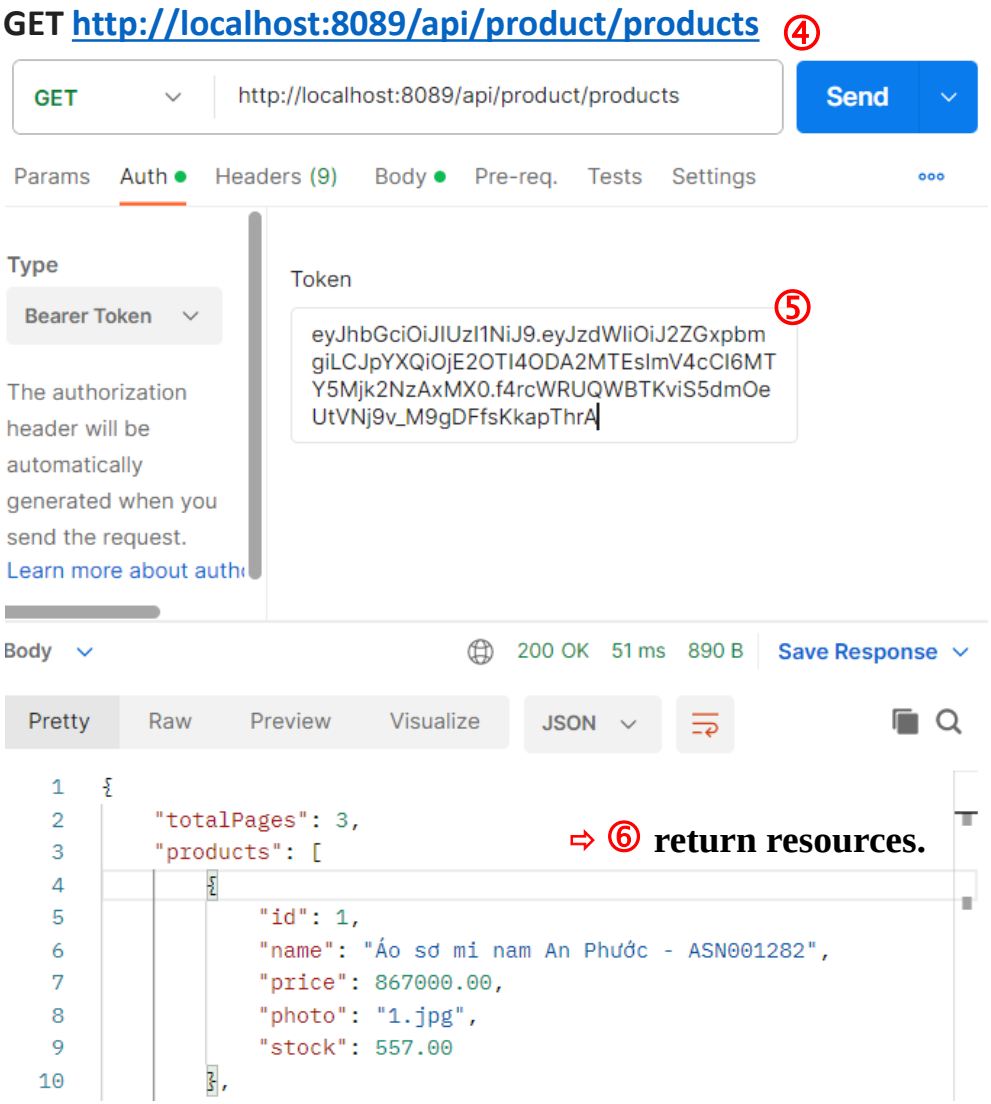
- Nếu xác thực thiếu token thì kết quả truy cập tài nguyên bị từ chối



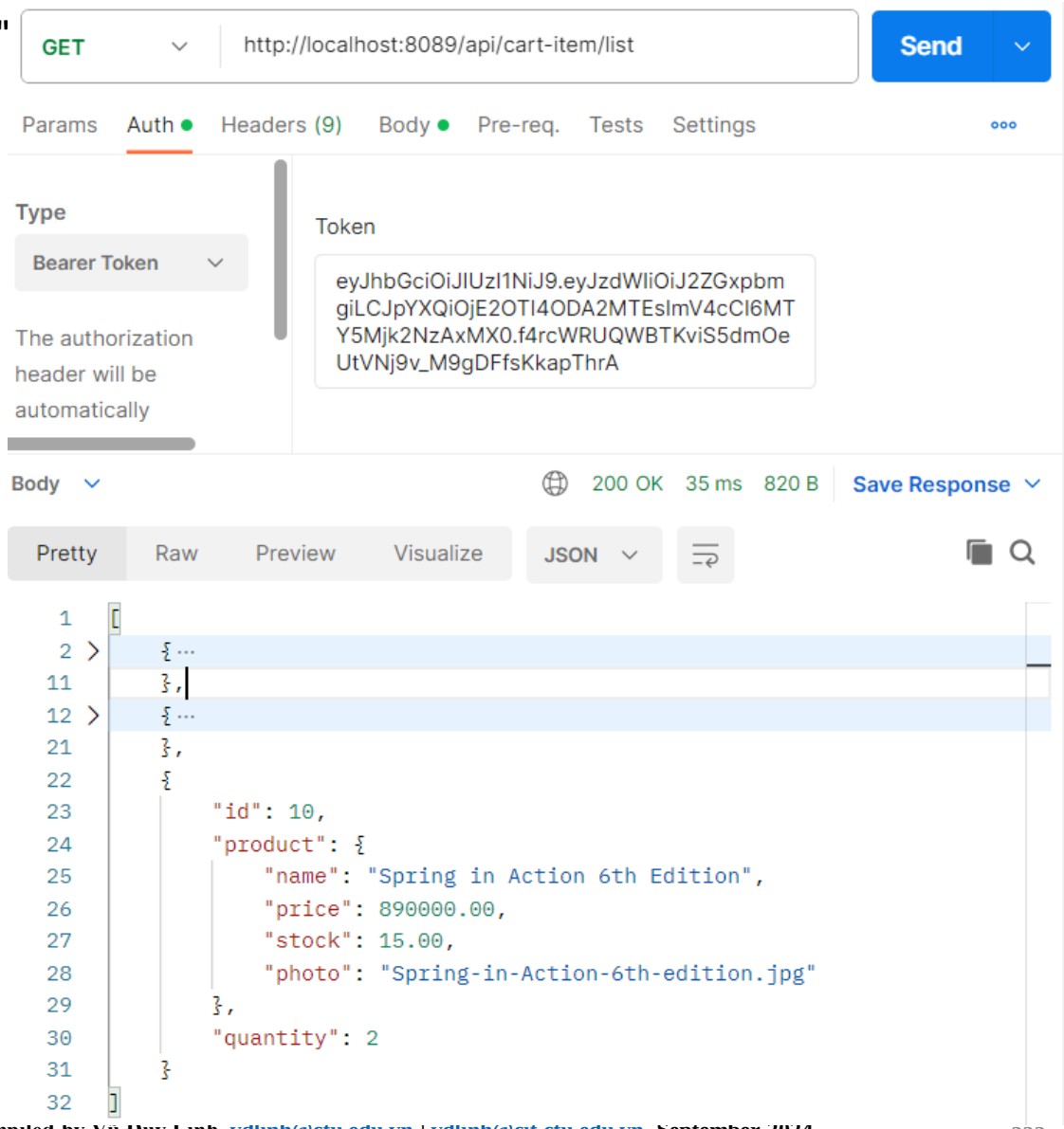
- **Nếu hết thời hạn**, **phải login lại** để server gửi lại access token mới, kết quả truy xuất tài nguyên như hình:

**Access resources:**

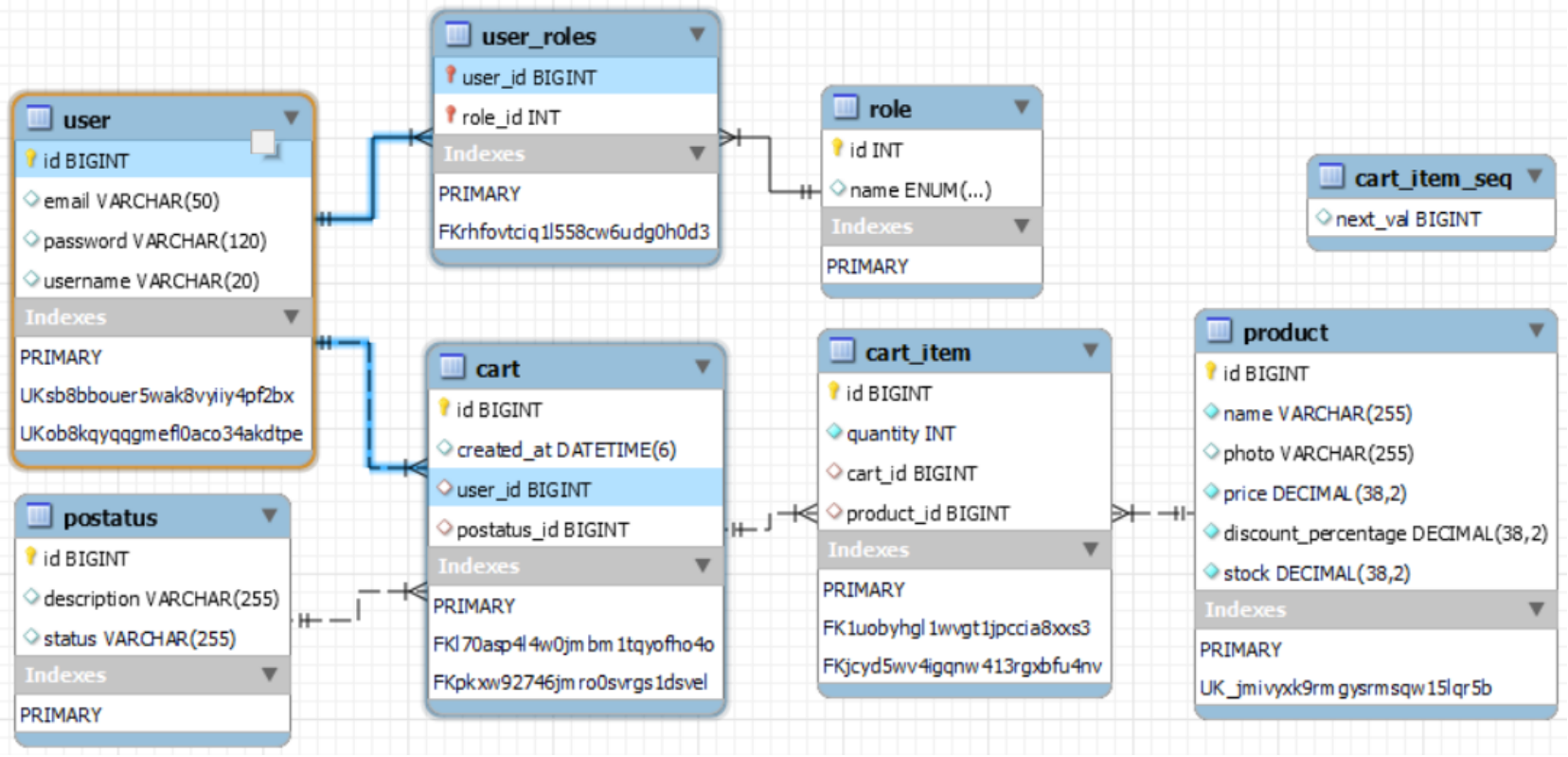
⇨ ④ ⇨ ⑤ ⇨ ⑥



- **Giỏ hàng của tài khoản "vdlinh"**

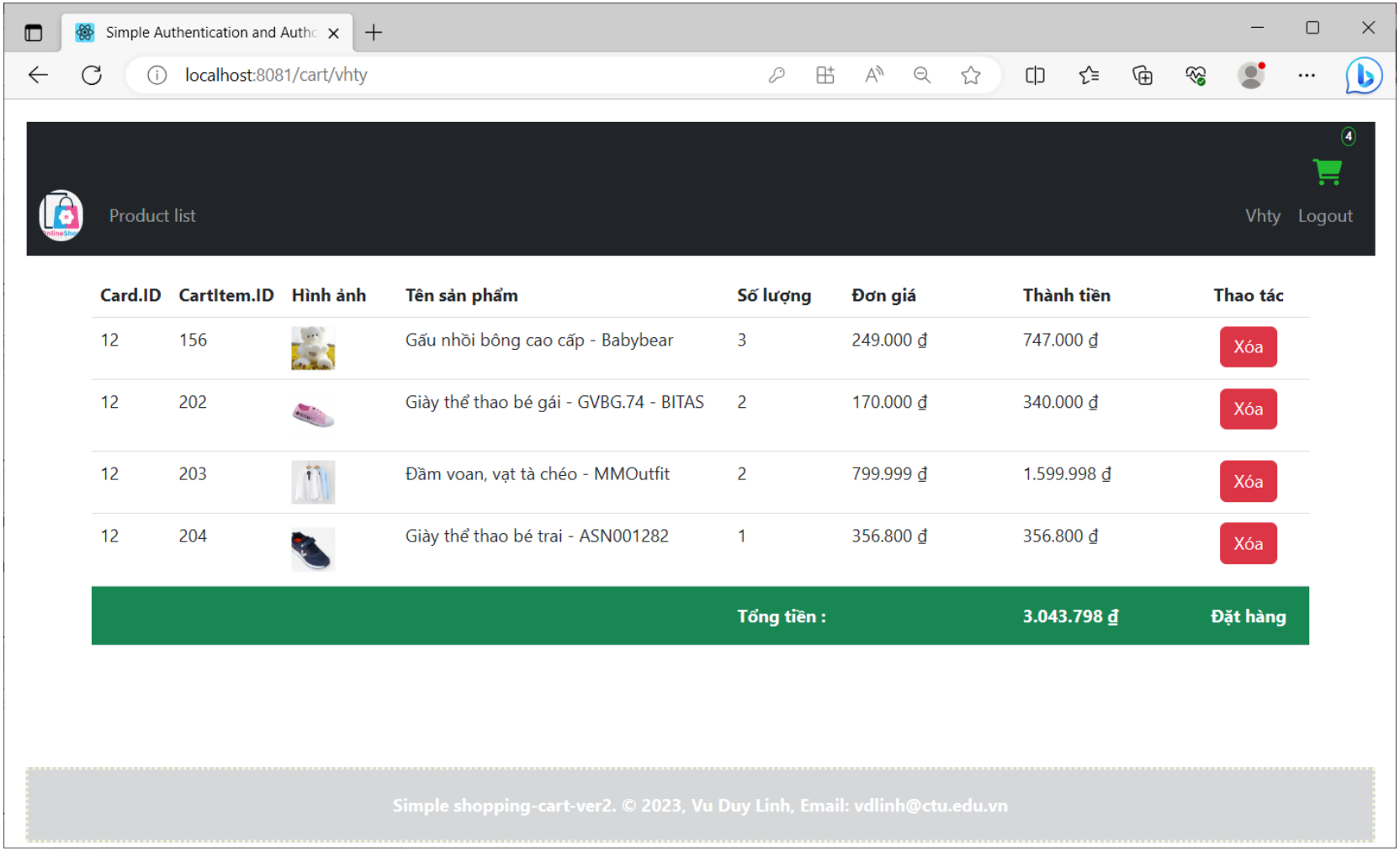


- **Project 4 : springboot3-backend-jwt-authentication-cart-ver2**




**User vs Cart (Order): o2m (mỗi KH có nhiều giỏ hàng để đặt hàng, khá sát thực tế)**
**Cart vs Cart_item (Order_item): o2m (mỗi giỏ có nhiều sản phẩm được chọn), khi giỏ hàng được đặt hàng thì toàn bộ các cart_item đều được đặt mua (khác với version_in_real: có thể chỉ chọn một số sản phẩm trong giỏ để đặt hàng)**
**Cart_item vs Product: m2o**

- Khi nhấp vào nút **Đặt hàng**, giỏ hàng mới trống sẽ được tạo cho khách.

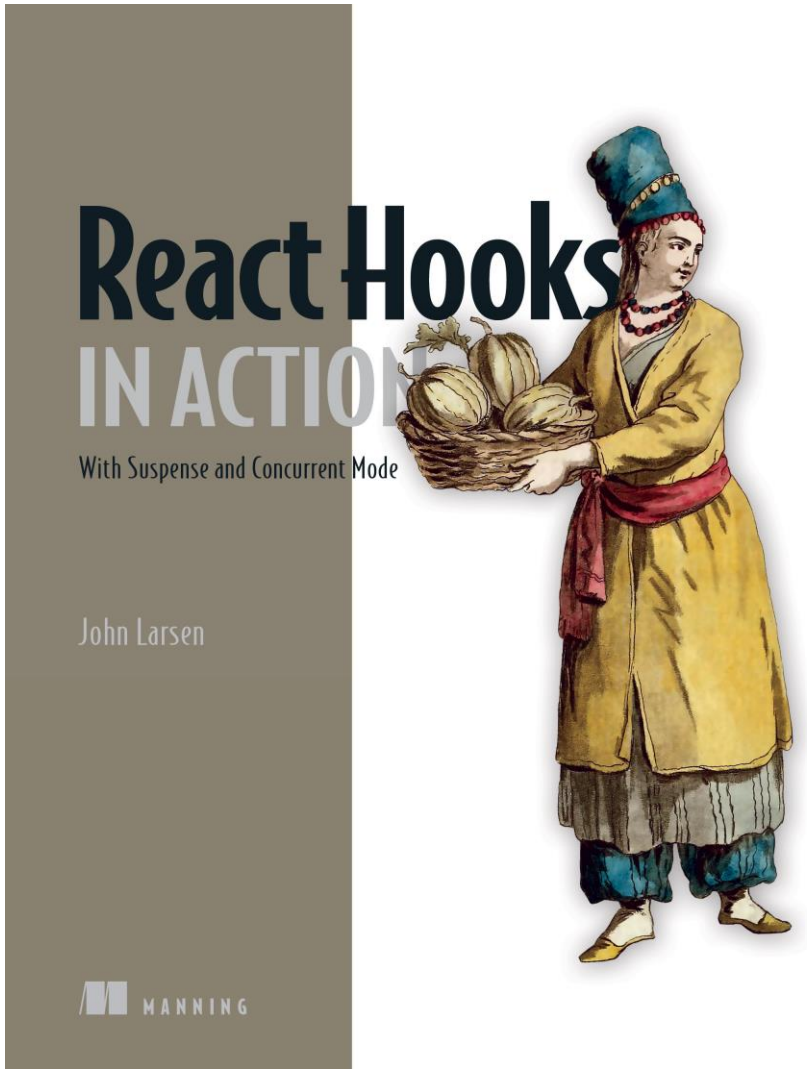


# Building web applications with Java

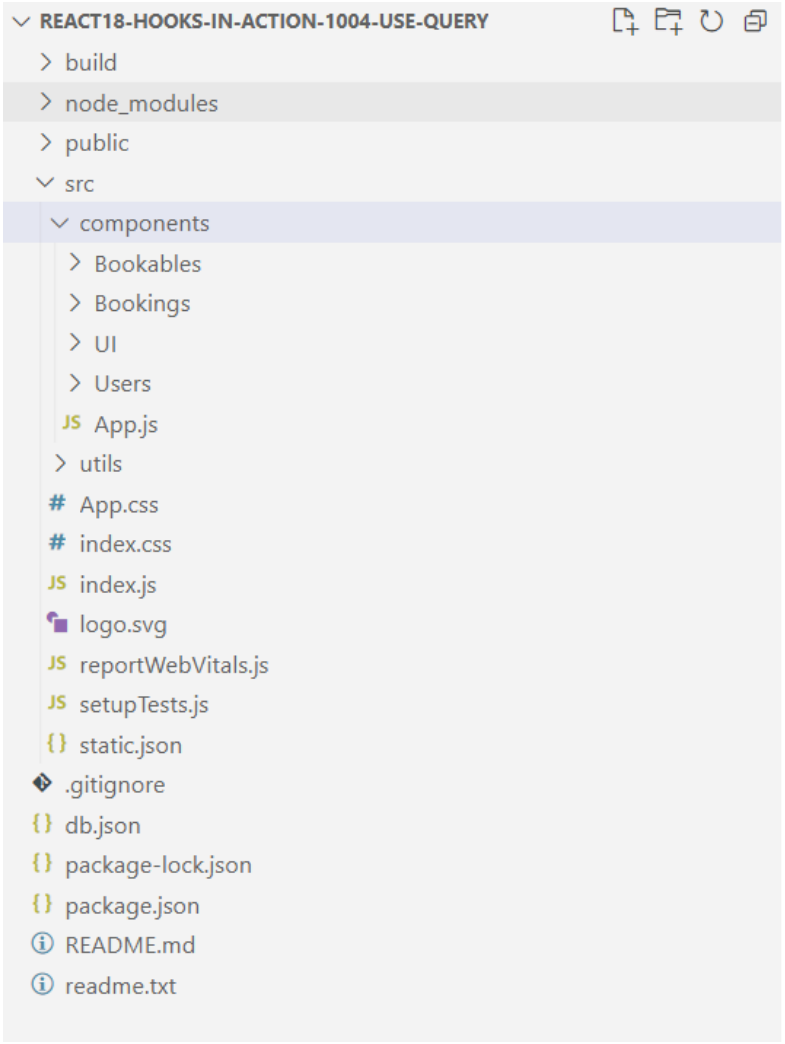
## Chap 6. Frontend using ReactJs 18.x

**Biên soạn: Vũ Duy Linh**
vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn

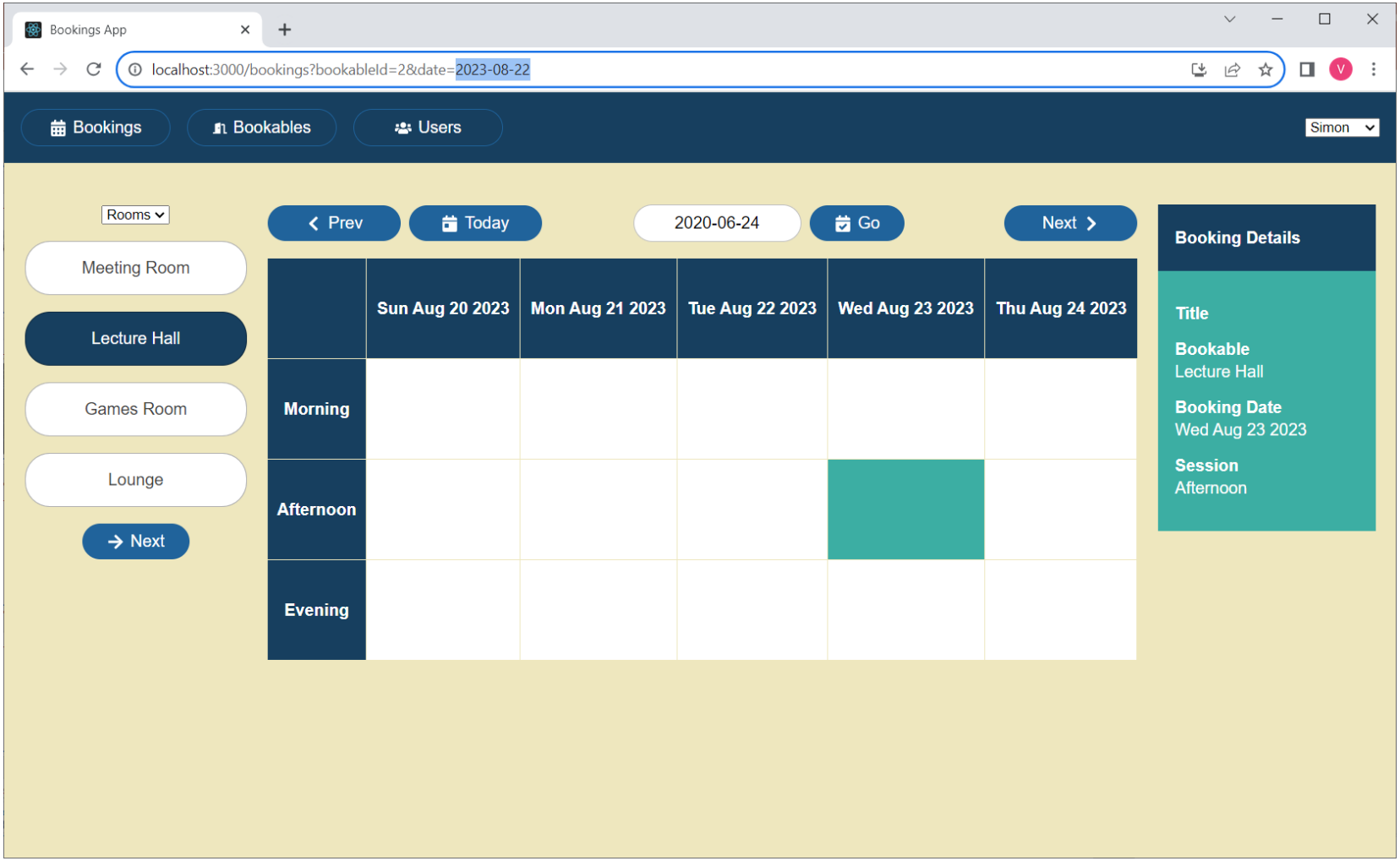
**Advice book:**


React Hooks
IN ACTION
With Suspense and Concurrent Mode
John Larsen
MANNING

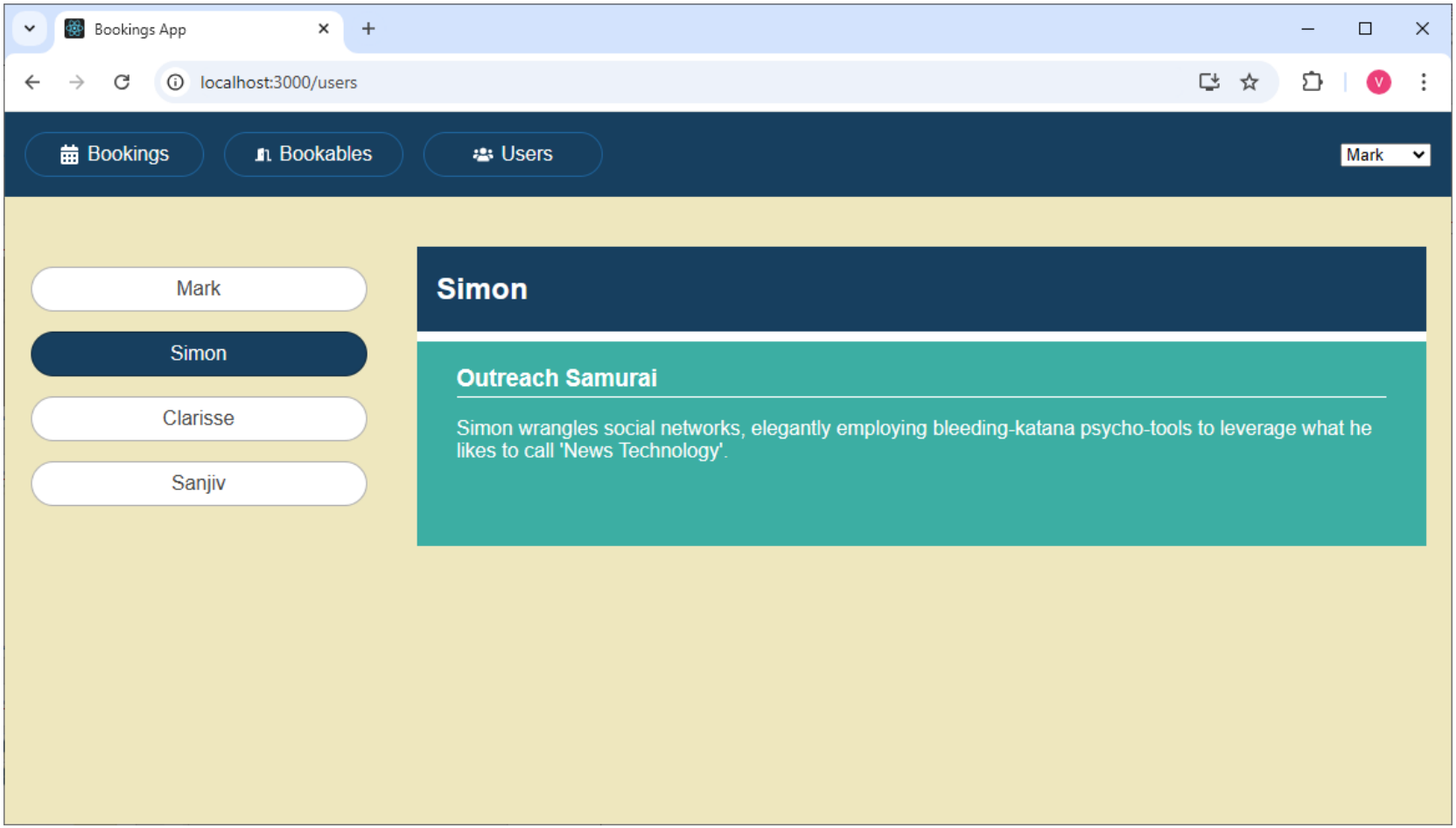
- Demo: D:\Java-web\react-fe\react18-hooks-in-action-1004-use-query




- **Booking page**

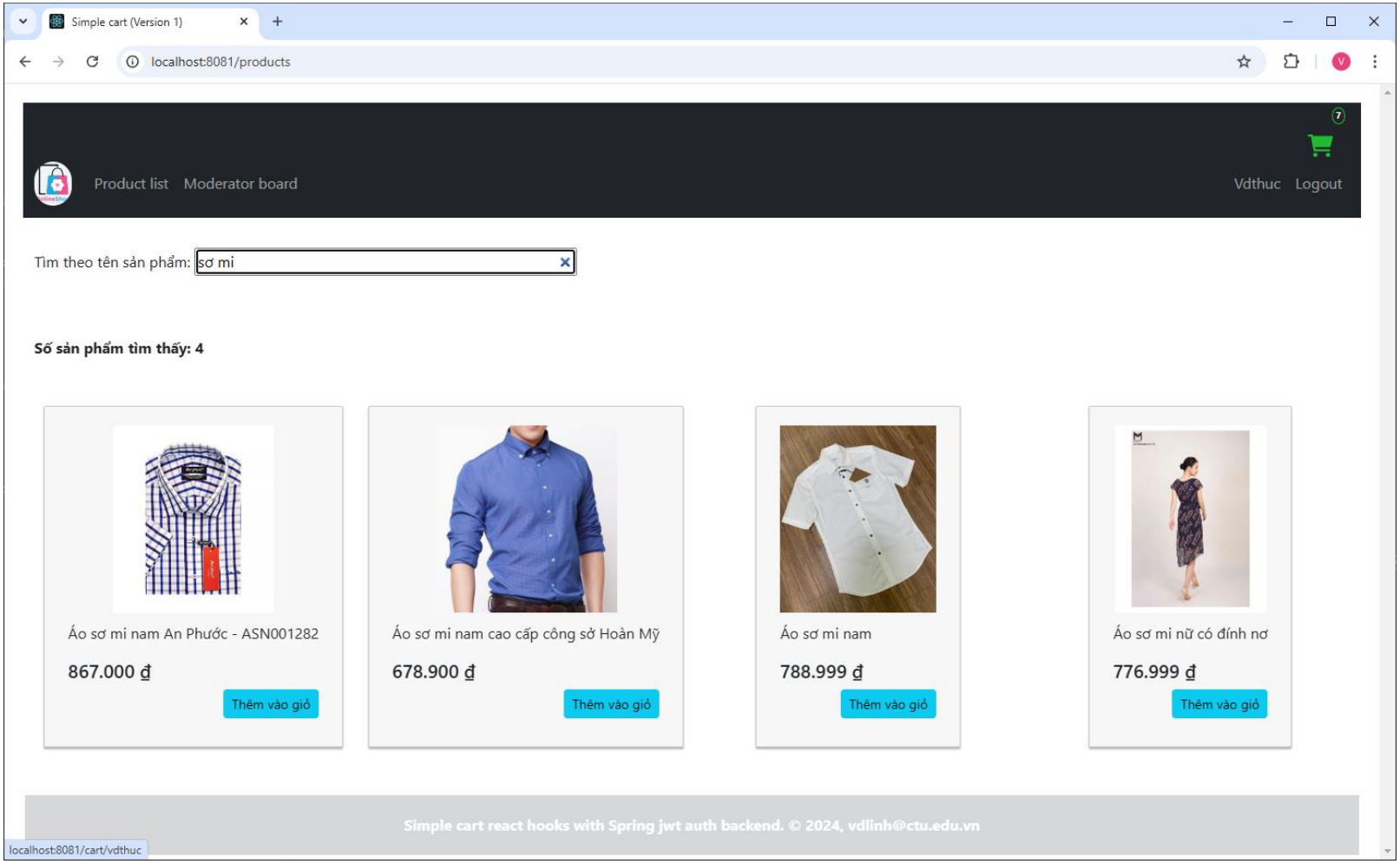


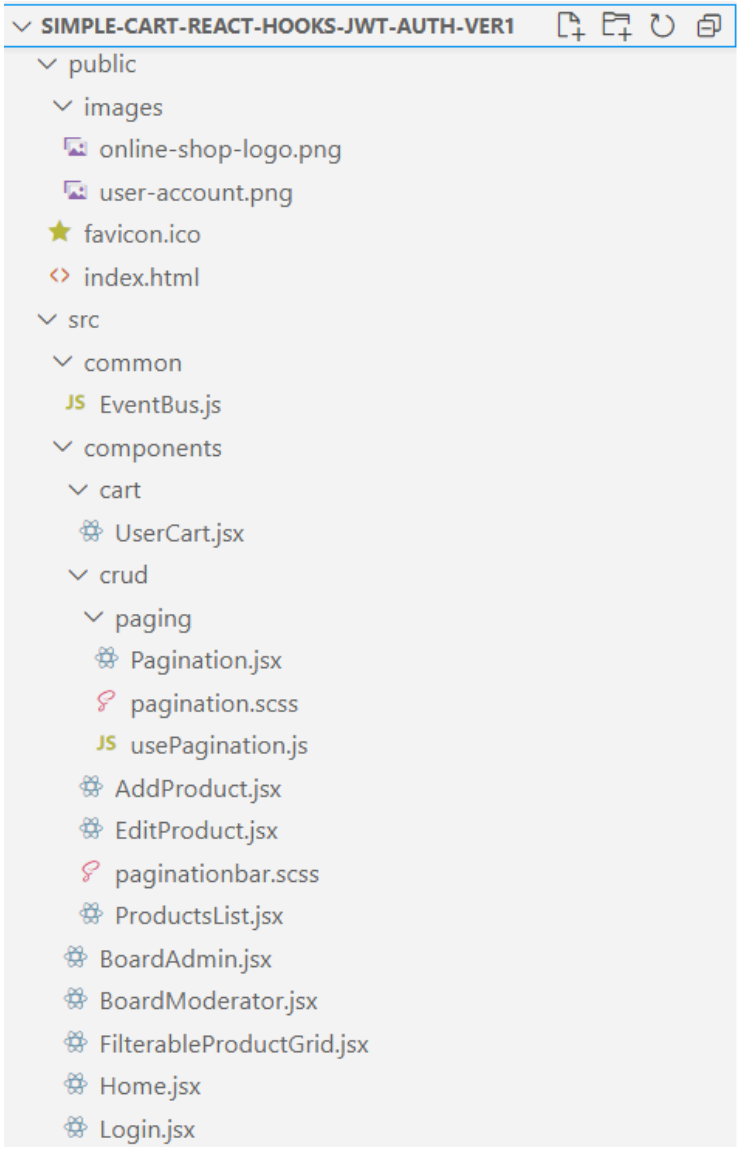
- **Users page**


Bookings App
localhost:3000/users
Bookings
Bookables
Users
Mark
Mark
Simon
Clarisse
Sanjiv
Simon
Outreach Samurai
Simon wrangles social networks, elegantly employing bleeding-katana psycho-tools to leverage what he likes to call 'News Technology'.

- **Project 2 (Front-end: simple-cart-react-hooks-jwt-auth-ver1**

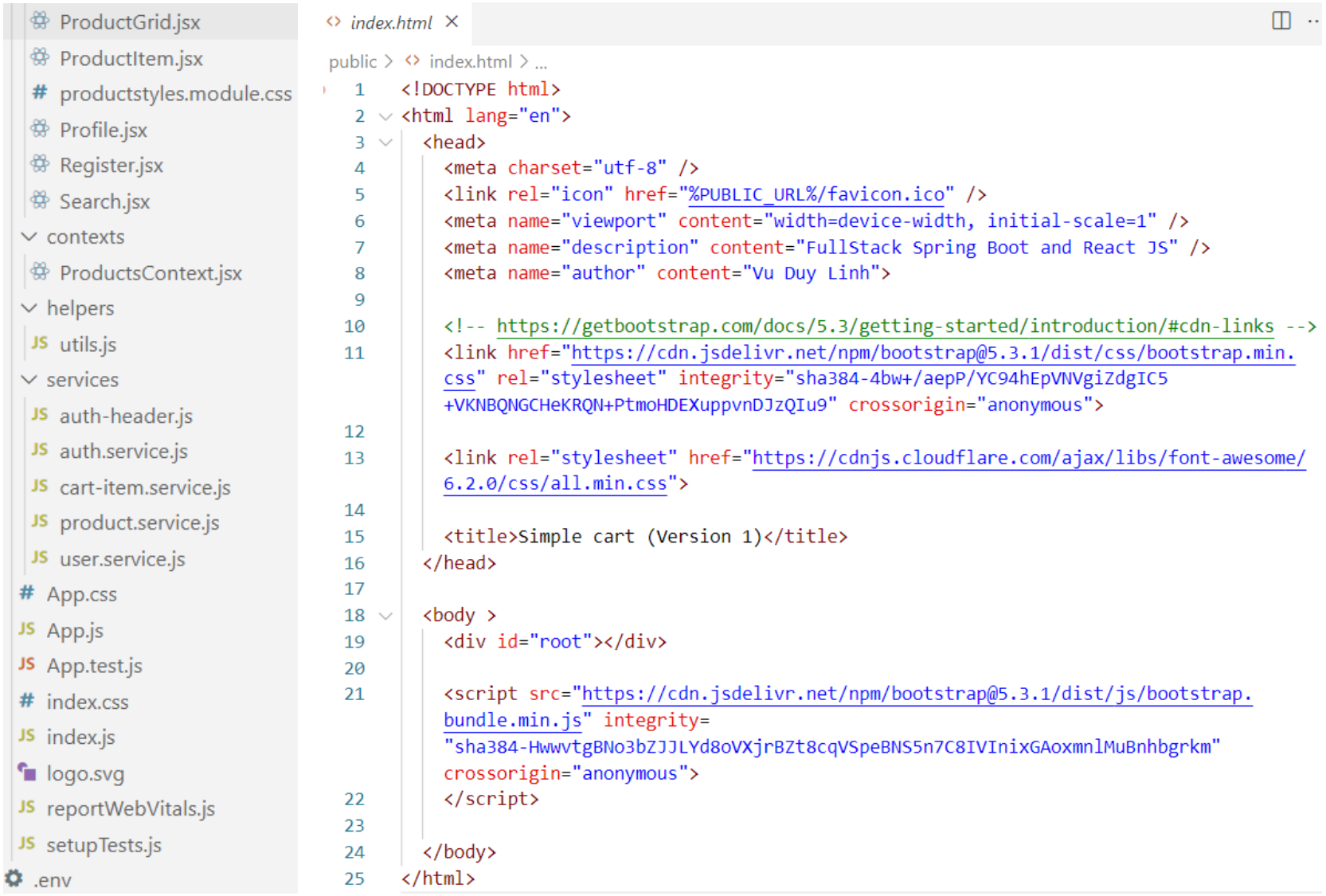
  **Back-end: springboot3-backend-jwt-authentication-cart )**



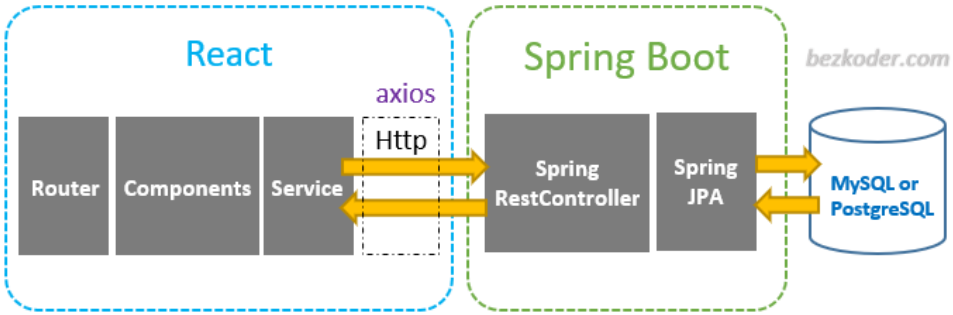



package.json > ...

```
1  {
2    "name": "simple-cart-react-hooks-jwt-auth-ver1",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.17.0",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.5.0",
9      "axios": "^1.4.0",
10     "classnames": "^2.3.2",
11     "node-sass": "^7.0.3",
12     "react": "^18.2.0",
13     "react-dom": "^18.2.0",
14     "react-router-dom": "^6.15.0",
15     "react-scripts": "5.0.1",
16     "react-table": "^7.8.0",
17     "react-validation": "^3.0.7",
18     "validator": "^13.11.0",
19     "web-vitals": "^2.1.4"
20   },
       Debug
21   "scripts": {
22     "start": "react-scripts start",
23     "build": "react-scripts build",
24     "test": "react-scripts test",
25     "eject": "react-scripts eject"
26   },
27   "eslintConfig": {
28     "extends": [
29       "react-app",
```


ProductGrid.jsx
ProductItem.jsx
productstyles.module.css
Profile.jsx
Register.jsx
Search.jsx
contexts
ProductsContext.jsx
helpers
utils.js
services
auth-header.js
auth.service.js
cart-item.service.js
product.service.js
user.service.js
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.env
index.html
public > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8" />
5 <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6 <meta name="viewport" content="width=device-width, initial-scale=1" />
7 <meta name="description" content="FullStack Spring Boot and React JS" />
8 <meta name="author" content="Vu Duy Linh">
9
10 <!-- https://getbootstrap.com/docs/5.3/getting-started/introduction/#cdn-links -->
11 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.
css" rel="stylesheet" integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5
+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9" crossorigin="anonymous">
12
13 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/
6.2.0/css/all.min.css">
14
15 <title>Simple cart (Version 1)</title>
16 </head>
17
18 <body >
19 
20
21 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.
bundle.min.js" integrity=
"sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous">
22 </script>
23
24 </body>
25 </html>

- **Architecture of Spring Boot React CRUD**



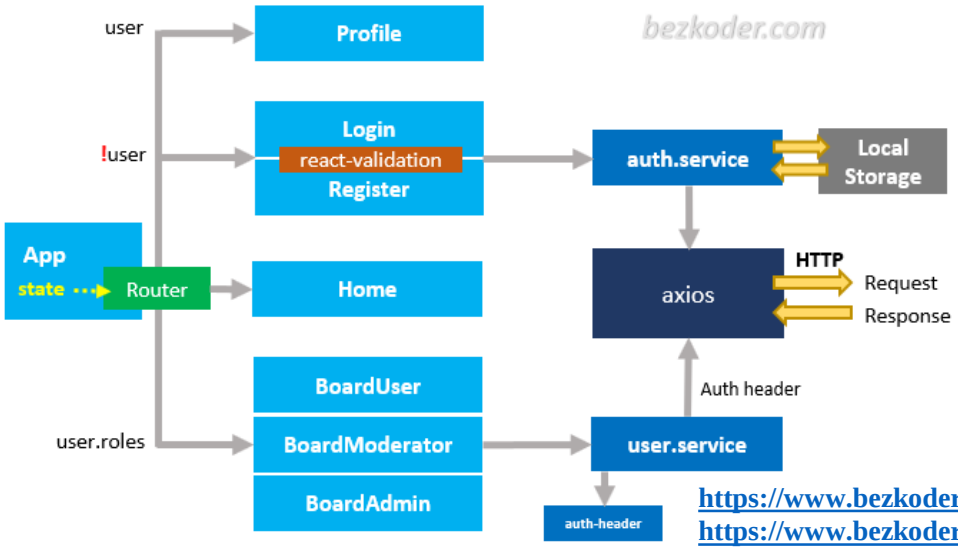

CartItemController.java

```java
package com.javaweb.controllers;

import org.springframework.beans.factory.annotation.Autowired;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/cart-item")
public class CartItemController {

    @Autowired private CustomerCartService customerCartService;

    @GetMapping("/list")
//  @PreAuthorize("hasRole('USER')  or hasRole('ADMIN')")
    public ResponseEntity<?> getAllByUserId(
            @AuthenticationPrincipal UserDetailsImpl userDetails) {

        return customerCartService.getAllByUserId(userDetails.getId());
    }

    @PostMapping("/create")
    @PreAuthorize("hasRole('USER')  or hasRole('ADMIN')")
    public ResponseEntity<?> create(@AuthenticationPrincipal UserDetailsImpl userDetails,
                                    @Valid @RequestBody CartItemRequest request) {

        return customerCartService.create(userDetails.getId(), request);
    }
```

services > JS cart-item.service.js > ...

```javascript
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8089/api/cart-item';

const getAllCartItems = () => {
    return axios.get(API_URL + '/list', {headers: authHeader()});
};

const createCartItem = (request) => {
    return axios.post(`${ API_URL}/create`, request,
                                    {headers: authHeader()});
};
```

- **React Component Diagram with Router, Axios & LocalStorage**




https://www.bezkoder.com/react-jwt-auth/
https://www.bezkoder.com/react-hooks-jwt-auth/

– **The App component** is a container with React Router (BrowserRouter). Basing on the state, the navbar can display its items.
– **Login & Register components** have form for data submission (with support of react-validation library). They call methods from auth.service to make login/register request.
– **auth.service methods** use axios to make HTTP requests. Its also store or get JWT from Browser Local Storage inside these methods.
– **Home component** is public for all visitor.
– **Profile component** displays user information after the login action is successful.
– **BoardUser, BoardModerator, BoardAdmin components** will be displayed by state user.roles. In these components, we use user.service to access protected resources from Web API.
– **user.service uses auth-header() helper function** to add JWT to HTTP header. auth-header() returns an object containing the JWT of the currently logged in user from Local Storage.

- Auth.service.js uses Local Storage

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/products"
         element={<FilterableProductGrid currentUser={currentUser}/>}/>
  {/* Cart for logged-in user  */}
  {currentUser ? (
    <Route path={`/cart/${currentUser.username}`} element={<UserCart/>}/>
    ):(
      <Route path={`/cart`} element={<UserCart/>} />
    )}

  <Route path="/login" element={<Login/>} />
  <Route path="/register" element={<Register/>} />
  <Route path="/profile" element={<Profile/>} />
  <Route path="/mod" element={<BoardModerator/>} />
  <Route path="/admin" element={<BoardAdmin/>}/>

  <Route path="/crud-products" element={<ProductsList />} />
  <Route path="/crud-products/:productId"  element={<EditProduct />}/>

  <Route path="/add-new-product"  element={<AddProduct/>} />
  <Route path="*"
    element={
      <main style={{ padding: "1rem" }}>
        <p>There's nothing here!</p>
      </main>
    } />
</Routes>
```

```
ices > auth.service.js > AuthService

const login = async (username, password) => {
  return axios
    .post(API_URL + "signin", {
      username,
      password,
    })
    .then((response) => {
      if (response.data.accessToken) {
        localStorage.setItem("user", JSON.stringify(response.data));
      }
```

```
components > Profile.jsx > Profile > currentUser
  import React from "react";
  import AuthService from "../services/auth.service";

  const Profile = () => {
    const currentUser = AuthService.getCurrentUser();

    return (
      <section className="container">
        <h3 className="mt-4 p-5 bg-primary text-whit
          <strong>{currentUser.username}</strong> Pr
        </h3>
```

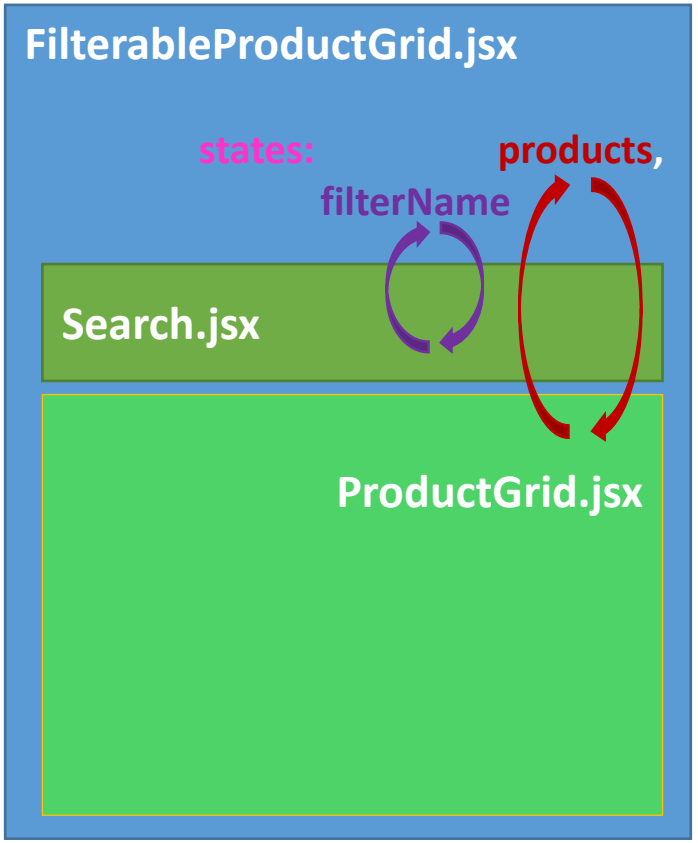
- Break The UI Into A Component Hierarchy

FilterableProductGrid.jsx

src > components > FilterableProductGrid.jsx > FilterableProductGrid > useEffect() c

```
1  import React, { useState, useEffect } from "react";
2
3  import Search from './Search';
4  import ProductGrid from './ProductGrid';
5  import styles from './productstyles.module.css';
6  import UserService from "../services/user.service";
7
8  const FilterableProductGrid = (props) => {
9    const [products, setProducts] = useState([]);
10   const [filterName, setFilterName] = useState('');
11
12   useEffect(() => {
13     UserService.getPublicContent().then(···
25     );
26   }, []);
27
28   const handleChangeFilterProductsByName = (productName) => {
29     setFilterName(productName);
30   }
31
32   return (
33     <article className={styles.article_block}>
34       <Search filterText={filterName}
35         onFilterTextChange={handleChangeFilterProductsByName}/>
36
37       <ProductGrid products={products}
38         filterText={filterName}
39         currentUser={props.currentUser}
40       />
41     </article>
42   );
43 };
44
45 export default FilterableProductGrid;
```

Ref: https://react.dev/learn/thinking-in-react



Search.jsx

src > components > Search.jsx > ...

```jsx
import React, { useRef } from "react";

const Search = (props) => {
    const inputRef = useRef();

    const handleFilterTextChange = (e) => {
        props.onFilterTextChange(e.target.value);  //from parent Component
    }

    return (
        <div className="col-sm-12 pb-5">
            {/* <form className="mb-3 mt-3"> */}
             <label>Tìm theo tên sản phẩm: {"   "}
                 <input type="search" size="50" placeholder="Search..." className="mb-3 mt-3"
                     value={props.filterText}
                     onChange={handleFilterTextChange}
                     ref={inputRef} />
             </label>
             {/* </form> */}
        </div>
      );
}

export default Search;

```

ProductGrid.jsx

src > components > ProductGrid.jsx > ...

```jsx
import React from "react";
import ProductItem from './ProductItem';
import styles from './productstyles.module.css';

const ProductsGrid = (props) => {
  const foundProducts = [];

  props.products.forEach((product) => {
    if (product.name.indexOf(props.filterText) === -1) {
      return;
    }
    // Correct! Key should be specified inside the array.
    foundProducts.push(<ProductItem key = {product.name}
                                    product = {product}
                                    currentUser = {props.currentUser} />)
  })

  return (
    <article className={styles.article_block}>
      <div style={{fontWeight: 'bold'}}> Số sản phẩm tìm thấy: {foundProducts.length}</div>

      <section className={styles.grid_container}>
        {foundProducts}
      </section>
    </article>
  );
};

export default ProductsGrid;
```

ProductItem.jsx

src > components > ProductItem.jsx > ProductItem

```jsx
import React from 'react';
import { Link, useNavigate} from 'react-router-dom';
import { formatNumber } from '../helpers/utils';
import styles from './productstyles.module.css';
import CartService from '../services/cart-item.service';

const ProductItem = (props) => {
  const navigate = useNavigate();

  const createCart = () => {
    alert("currentUser.id= " + props.currentUser.id + "; props.product.id= " + props.product.id)

    const request = {
      productId: props.product.id,
      quantity: 1,
    };

    CartService.createCartItem(request).then((res) => {
      if (res.data) {
        // navigate('/cart');
        setTimeout(()=> navigate('/cart'), 100);
        setTimeout(()=> navigate('/products'), 400); //just for fun
      }
    });
  };

```

```
27    return (
28      <div className="card card-body">
29        <img
30          style={{ display: 'block', margin: '0 auto 10px', maxHeight: '200px' }}
31          className="img-fluid"
32          src={'http://localhost:8089/api/product/images/' + props.product.photo}
33          alt=""
34        />
35
36        <p>{props.product.name}</p>
37
38        <h5 className="text-start">{formatNumber(props.product.price)}</h5>
39
40        <div className="text-end">
41
42          {props.currentUser ? (
43            <button type="button" onClick={createCart} className="btn btn-info btn-sm" >
44              Thêm vào giỏ
45            </button>
46          ) : (
47            <Link to="/login" className={styles.add_cart}>
48              Xem chi tiết
49            </Link>
50          )}
51        </div>
52      </div>
53    );
54  };
55
56  export default ProductItem;
```

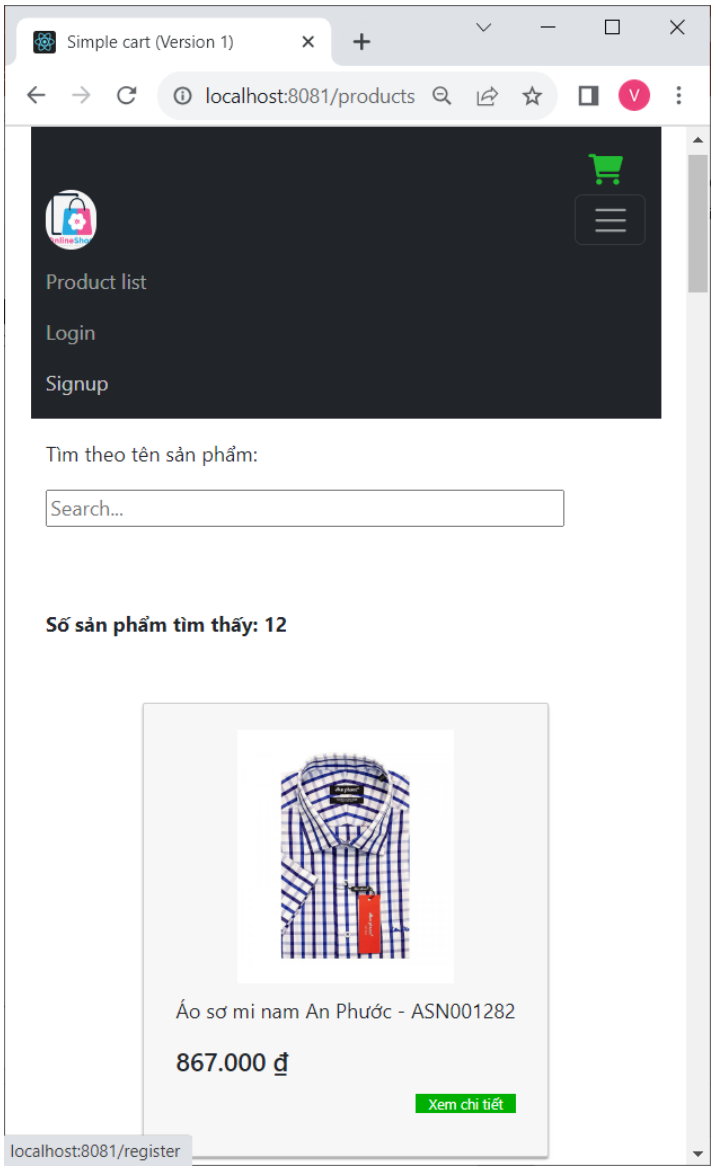
UserCart.jsx

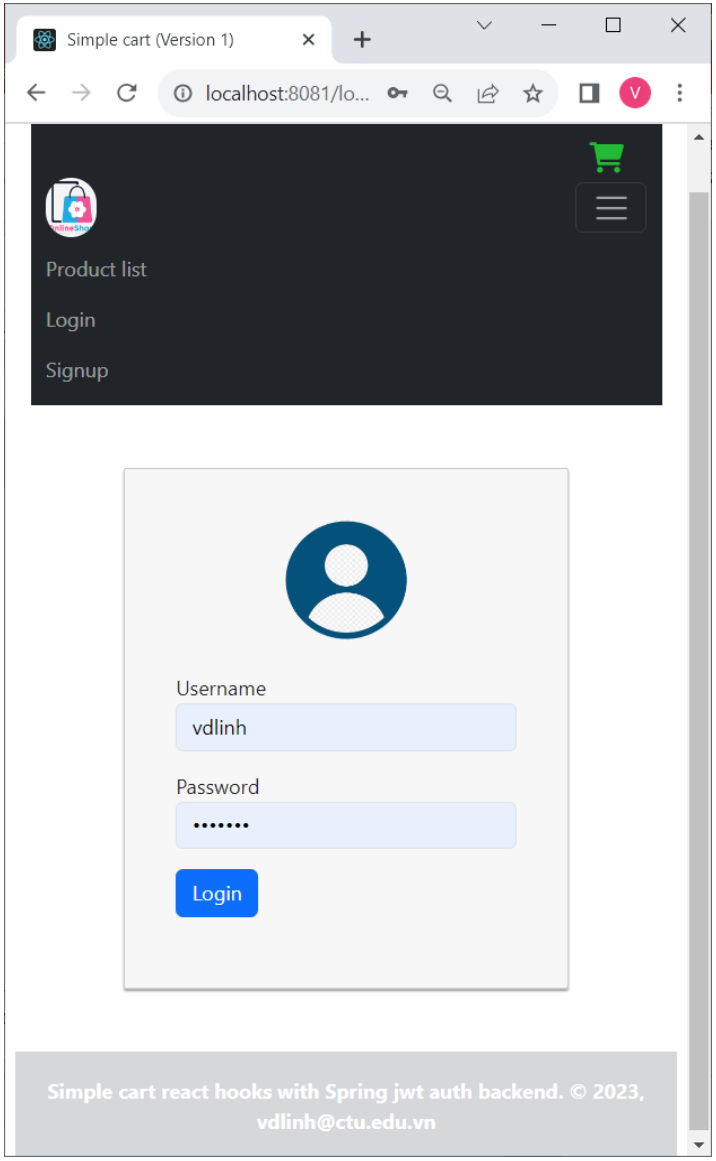
src > components > cart > UserCart.jsx > UserCart

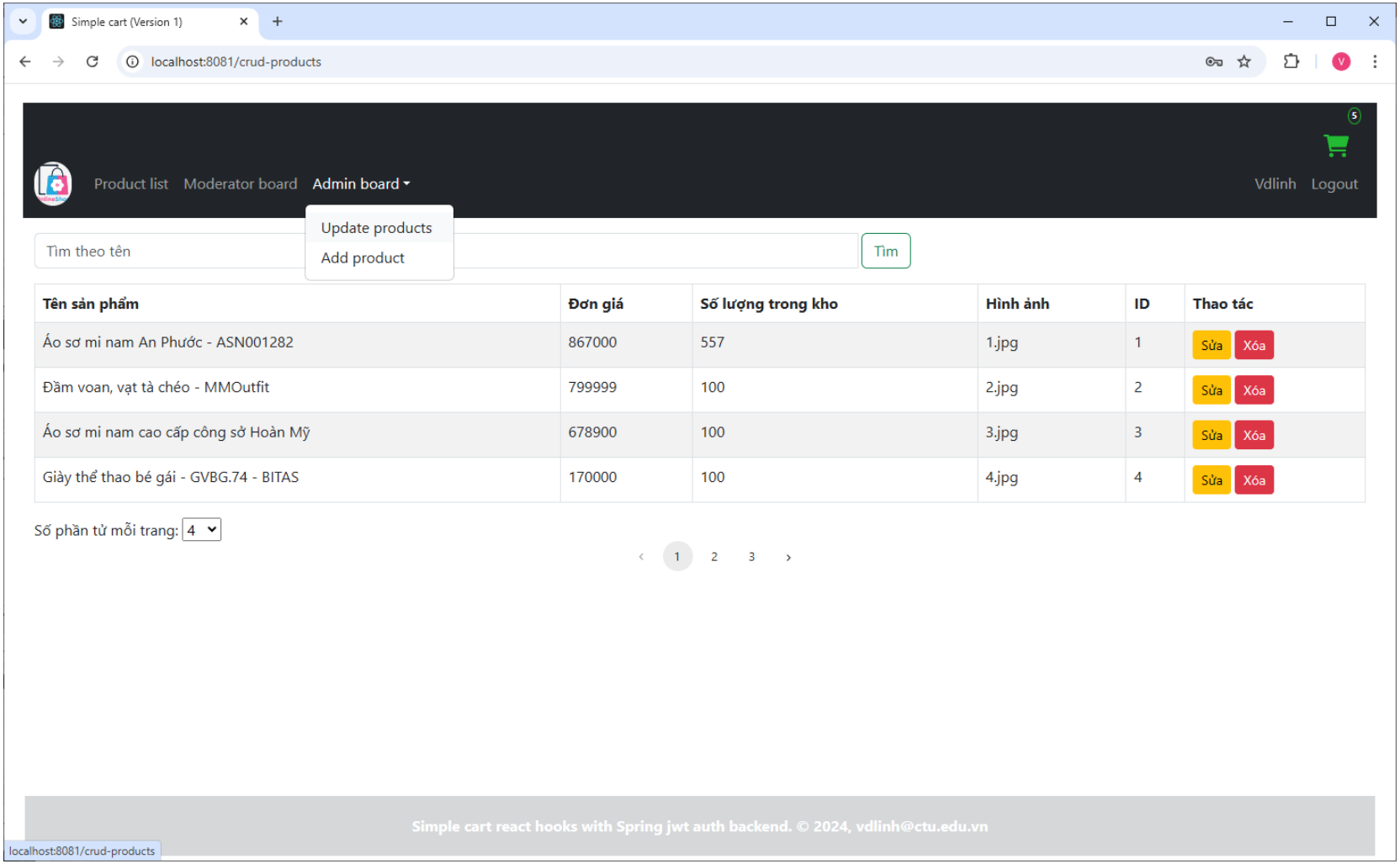
```jsx
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { formatNumber } from '../../helpers/utils';
import CartService from '../../services/cart-item.service';

const UserCart = () => {
  const [cartItems, setCartItems] = useState([]);

  useEffect(() => {
    CartService.getAllCartItems().then((res) => {
      if (res.data) {
        setCartItems(res.data);
      }
    });
  }, []);

  const deleteCartItem = (id) => {
    CartService.deleteCartItem(id).then((res) => {
      if (res.data) {
        let updatedCarts = [...cartItems].filter((i) => i.id !== id);
        setCartItems(updatedCarts);
      }
    });
  };

  const loggedUserCart = cartItems.map((cartItem, index) => {
    return (
```

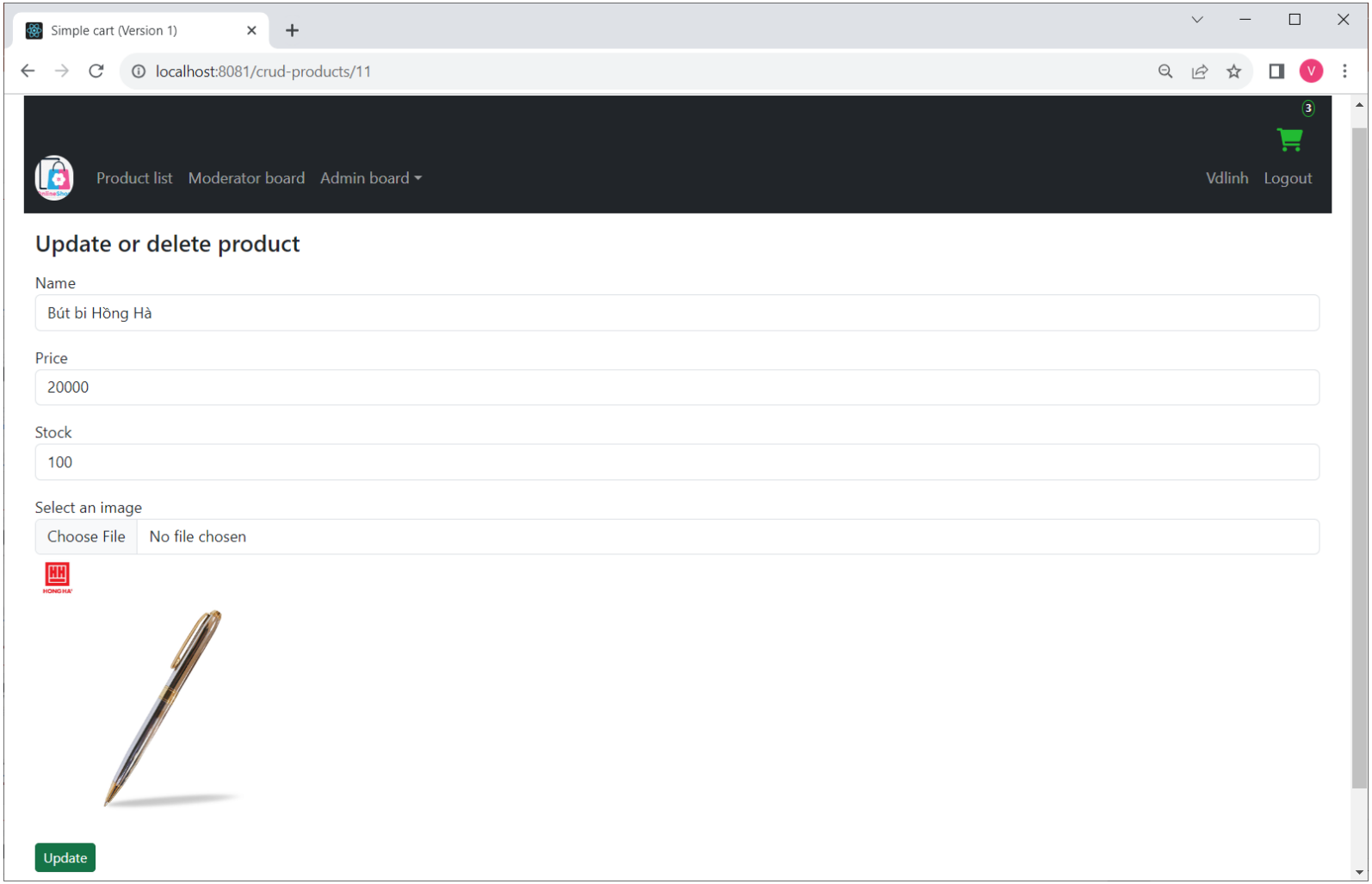
```
28          <tr key={cartItem.id} >
29            <td>
30              <img style={{ width: "40px" }}
31                src={'http://localhost:8089/api/product/images/' + cartItem.product.photo}  alt="" />
32            </td>
33            <td>{cartItem.product.name}</td>
34            <td>{formatNumber(cartItem.product.price)}</td>
35            <td>{cartItem.quantity}</td>
36            <td>{formatNumber(cartItem.product.price * cartItem.quantity)}</td>
37            <td >
38              <button className="btn btn-danger" onClick={() => deleteCartItem(cartItem.id)}>Xóa</button>
39            </td>
40          </tr>
41        );
42      });
43
44      const handleTotalPrice = () => {
45        var totalPrice = 0;
46        if (cartItems.length > 0) {
47          cartItems.forEach((item) => {
48            const price = item.product.price * item.quantity;
49            totalPrice += price;
50          });
51        }
52        return totalPrice;
53      };
54
55      return (
56        <section className="container-fluid" style={{ width: "100%" }}>
57          <table className="table">
58            <thead>
```

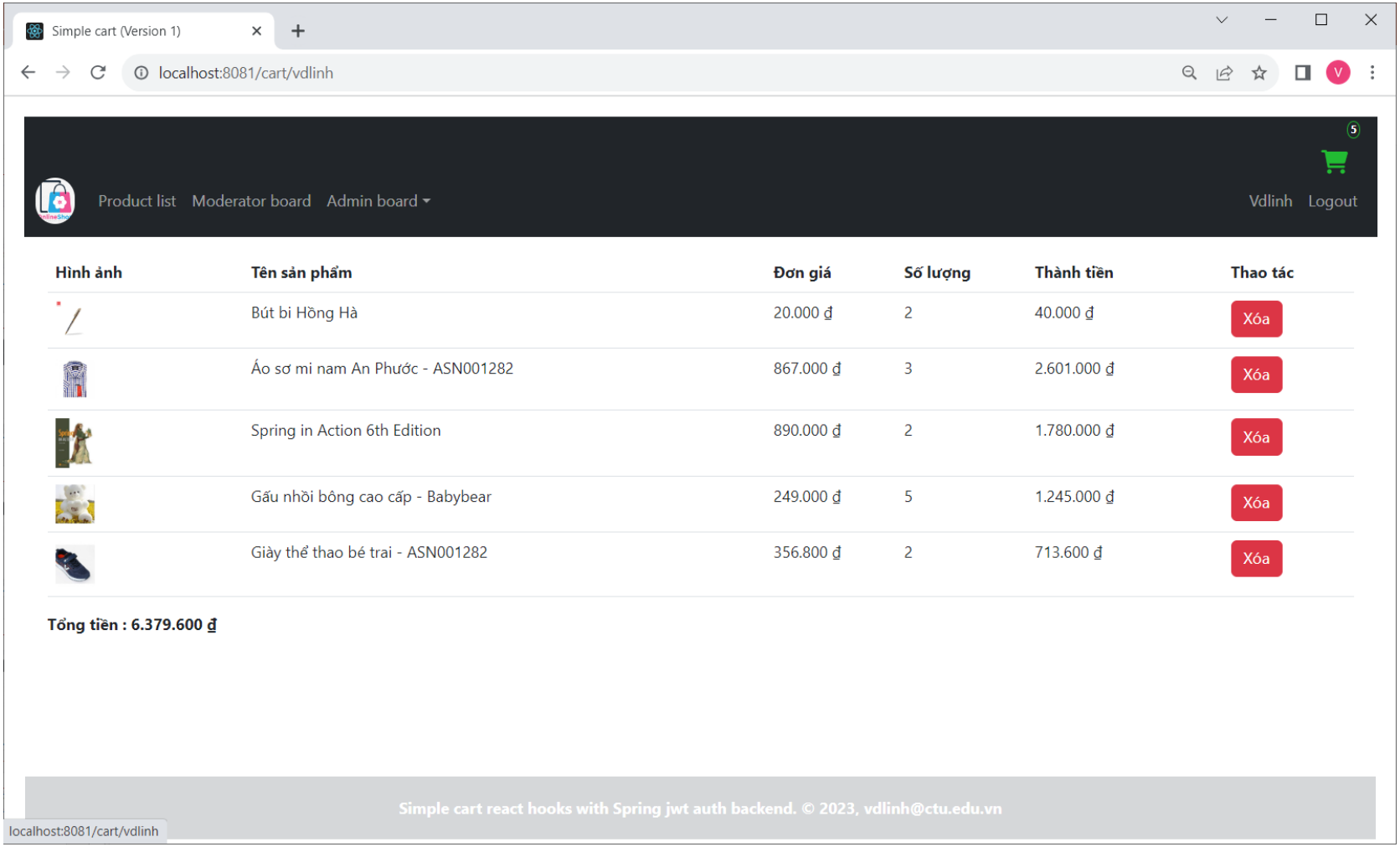
```
59            <tr>
60              <th width="15%">Hình ảnh</th>
61              <th width="40%">Tên sản phẩm</th>
62              <th width="10%">Đơn giá</th>
63              <th width="10%">Số lượng</th>
64              <th width="15%">Thành tiền</th>
65              <th width="10%">Thao tác</th>
66            </tr>
67          </thead>
68
69          <tbody>{loggedUserCart}</tbody>
70
71        </table>
72
73        { handleTotalPrice() > 0 &&
74          <p style={{ fontWeight: "bold" }}>Tổng tiền: {formatNumber(handleTotalPrice())}</p> }
75
76        { cartItems.length === 0 && (
77          <div className="no-cart">
78            <p>Giỏ hàng của bạn hiện đang trống.</p>
79            <Link to={'/products'} className="navbar-brand">
80              <button type="button" className="btn btn-primary">
81                Tiếp tục mua
82              </button>
83            </Link>
84          </div>
85        )}
86      </section>
87    );
88  };
89
90  export default UserCart;
```


Simple cart (Version 1)
localhost:8081/products
Product list
Login
Signup
Tìm theo tên sản phẩm:
Search...
Số sản phẩm tìm thấy: 12
Áo sơ mi nam An Phước - ASN001282
867.000 ₫
Xem chi tiết
localhost:8081/register

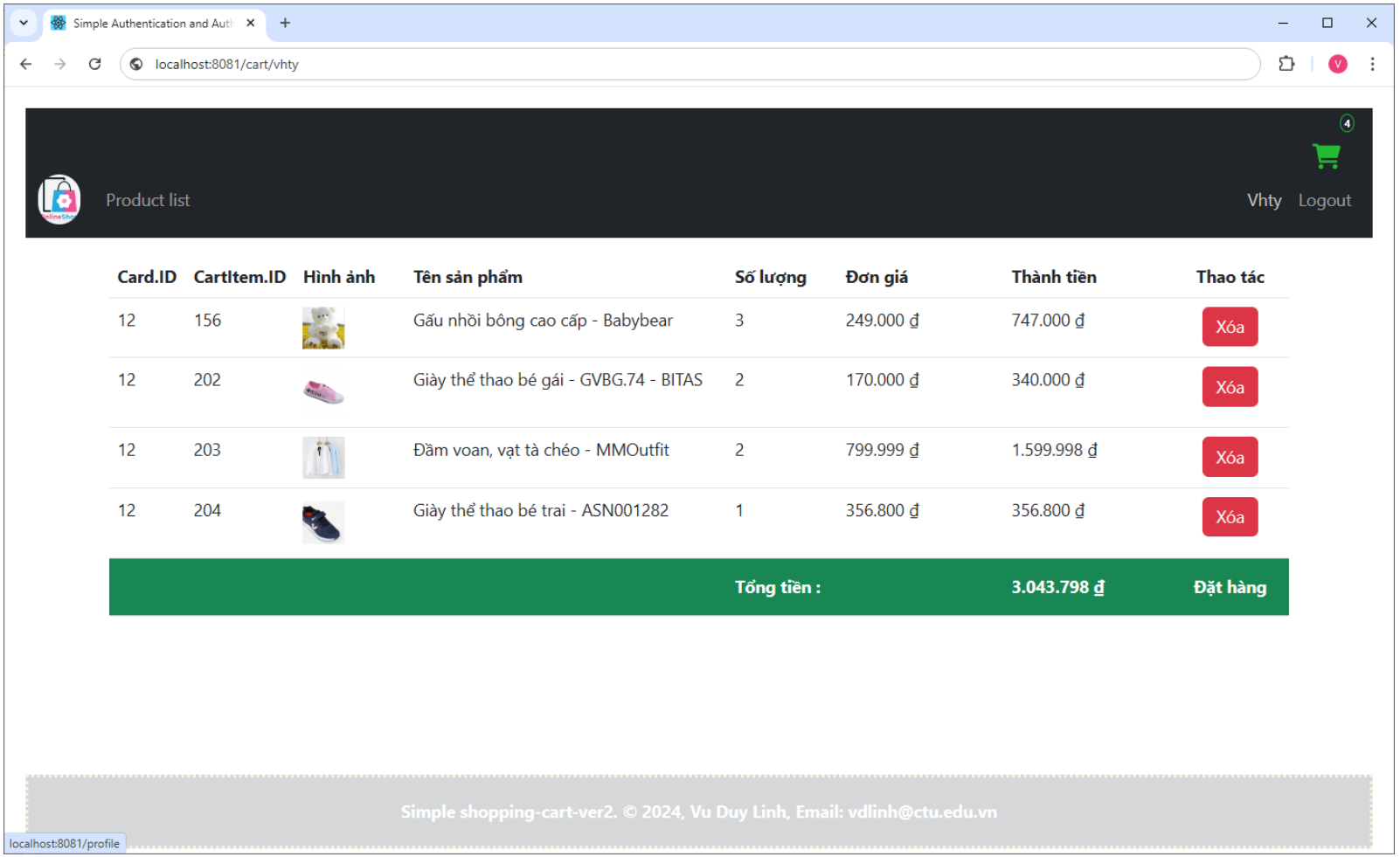

Simple cart (Version 1)
localhost:8081/lo...
Product list
Login
Signup
Username
vdlinh
Password
Login
Simple cart react hooks with Spring jwt auth backend. © 2023,
vdlinh@ctu.edu.vn


Simple cart (Version 1)
localhost:8081/crud-products
Product list
Moderator board
Admin board
Update products
Add product
Vdlinh
Logout
Tìm theo tên
Tìm
Tên sản phẩm
Đơn giá
Số lượng trong kho
Hình ảnh
ID
Thao tác
Áo sơ mi nam An Phước - ASN001282
867000
557
1.jpg
1
Sửa
Xóa
Đầm voan, vạt tà chéo - MMOutfit
799999
100
2.jpg
2
Sửa
Xóa
Áo sơ mi nam cao cấp công sở Hoàn Mỹ
678900
100
3.jpg
3
Sửa
Xóa
Giày thể thao bé gái - GVBG.74 - BITAS
170000
100
4.jpg
4
Sửa
Xóa
Số phần tử mỗi trang: 4
1
2
3
Simple cart react hooks with Spring jwt auth backend. © 2024, vdlinh@ctu.edu.vn
localhost:8081/crud-products


Simple cart (Version 1)
localhost:8081/crud-products/11
Product list
Moderator board
Admin board
Vdlinh
Logout
Update or delete product
Name
Bút bi Hồng Hà
Price
20000
Stock
100
Select an image
Choose File
No file chosen
Update


Simple cart (Version 1)
localhost:8081/cart/vdlinh
Product list
Moderator board
Admin board
Vdlinh
Logout
Hình ảnh
Tên sản phẩm
Đơn giá
Số lượng
Thành tiền
Thao tác
Bút bi Hồng Hà
20.000 ₫
2
40.000 ₫
Xóa
Áo sơ mi nam An Phước - ASN001282
867.000 ₫
3
2.601.000 ₫
Xóa
Spring in Action 6th Edition
890.000 ₫
2
1.780.000 ₫
Xóa
Gấu nhồi bông cao cấp - Babybear
249.000 ₫
5
1.245.000 ₫
Xóa
Giày thể thao bé trai - ASN001282
356.800 ₫
2
713.600 ₫
Xóa
Tổng tiền : 6.379.600 ₫
Simple cart react hooks with Spring jwt auth backend. © 2023, vdlinh@ctu.edu.vn
localhost:8081/cart/vdlinh

- B.E: Project 4 ◆ Chapter 5 (**springboot3-backend-jwt-authentication-cart-ver2)**
- F.E: simple-cart-react-hooks-jwt-auth-ver2



# References

## Ebook


1. Amuthan Ganeshan, Spring MVC: Beginner's Guide - Second Edition, 2016, Packt Publishing.
2. Anghel Leonard, Spring Boot Persistence Best Practices, 2020, APress.
3. Craig Walls, Spring in Action, Sixth Edition, 2022, Manning.
4. John Larsen, React hooks in action with suspense and concurrent mode, 2021, Manning.
5. Marten Deinum and Iuliana Cosmina, Pro Spring MVC with WebFlux: Web Development in Spring Framework 5 and Spring Boot 2, 2021, Apress.
6. Robin Wieruch, The Road to React, 2023, Learnpub.


## Website


1. BezKoder, the website for programming languages & technique, https://www.bezkoder.com/, visited 2024.
2. Javaguides, the Java/Java EE technologies and Full-Stack Java development, https://www.javaguides.net/, visited 2024.
3. React, the library for web and native user interfaces, https://react.dev/, visited 2024.
4. Spring IO is a cohesive, versioned platform for building modern applications, https://spring.io/, visited 2024.

# Xây dựng ứng dụng web với Java

## Module: Building web applications with Java

&  18

**Biên soạn: Vũ Duy Linh**

vdlinh@ctu.edu.vn | vdlinh@cit.ctu.edu.vn