Online Shopping Website

Backend : Spring boot 3.0, Spring security 6.0
FrontEnd : HTML,Bootstrap, CSS, Thymeleaf
DB : MySQL
Tools : STS
HOST : AWS, HEROKU

1. creating project
STS --> file --> new --> spring starter project --> give name, maven, jdk 22 ---> Spring web, Mysql Driver, Spring data jpa, Thymeleaf --> finish
add spring-boot-devtools dependency
2. Create controller, service, repository under scr/main/java
3. create dynamic pages like base, index under templates under scr/main/resources
4. create css, js, img folder under static under scr/main/resources
5. create style.css, script.js under css and js folder respectively
6. go to bootstrap in google, go to first link, docs , include bootstrap's css and js copy the link : copy only links (both links) and paste it in base.html page

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Bootstrap demo</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
</head>
<body>
<h1>Hello, world!</h1>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
</body>
</html>
```

7. to get icons : search font awesome cdn, go to first link, copy latest link </> and paste it in base.html

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-
SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A==" crossorigin="anonymous"
referrerpolicy="no-referrer" />
```

1. Base.html

I need these in navbar : left : Ecom Store, Home, Products, Category(dropdown) right : login, resister
Go to bootstrap, search nav bar , scroll and copy first one
search for font awesome cdn ❼ go to second link (www.fontawesome.com) -> icon
❼ search : shopping ❼ select any cart icon and click and copy tag and paste it I base.html

❼ search : home❼ select any home icon and click and copy tag and paste it I base.html

❼ search : login❼ select any login icon and click and copy tag and paste it I base.html

Header
<nav class="navbar navbar-expand-lg bg-body-tertiary **bg-primary fixed-top navbar-dark**">

****Navbar **Ecom Store**
<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

</button>

<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
Home

<li class="nav-item">
Product

<li class="nav-item">
Link

<li class="nav-item dropdown">

Dropdown **Category**

<ul class="dropdown-menu">
Action
Another action
<hr class="dropdown-divider">
Something else here

<li class="nav-item">
Disabled

<form class="d-flex" role="search">
<input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
<button class="btn btn-outline-success" type="submit">Search</button>
</form>
Copy above ul tag to below and remove not required code
<ul class="navbar-nav mems-auto mb-2 mb-lg-0">
<li class="nav-item">
****Login

<li class="nav-item">
Register

<li class="nav-item">
Admin

</nav>
Dynamic content will ne loaded here
** **
Footer
ecom.com

2. Index.html

I need a big image under nav bar
Go to bootstrap and search for carousel copy first one and paste in index.html in section tag
Note : add all the images to img folder
Slider

```
<div id="carouselExample" class="carousel slide">
<div class="carousel-inner">
<div class="carousel-item active">
<img src="img/ecom1.png" class="d-block w-100" alt="..." height="350px">
</div>
<div class="carousel-item">
<img src=" img/ecom2.png " class="d-block w-100" alt="..." height="350px">
</div>
<div class="carousel-item">
<img src=" img/ecom3.png " class="d-block w-100" alt="..." height="350px">
</div>
</div>
<button class="carousel-control-prev" type="button" data-bs-target="#carouselExample" data-bs-slide="prev">
<span class="carousel-control-prev-icon" aria-hidden="true"></span>
<span class="visually-hidden">Previous</span>
</button>
<button class="carousel-control-next" type="button" data-bs-target="#carouselExample" data-bs-slide="next">
<span class="carousel-control-next-icon" aria-hidden="true"></span>
<span class="visually-hidden">Next</span>
</button>
</div>
```

Category

```
<div class="container">
<div class="row">
<p class="text-center fs-4">Category</p>
<div class="col-md-2">
<div class="card rounded-circle shadow-sm p-3 mb-5 bg-body-tertiary rounded ">
<div class="card-body text-center"><img src="img/category_img/appli.png" width="65%" height="140px"> <p>Eelectronics</p></div>
</div>
</div>
```

shadow-sm p-3 mb-5 bg-body-tertiary rounded ❼ search for shadow in bootstrap and copy small shadow code
Added these images in img folder ❼ category_img folder
Same code goes for
Beauty : img/category_img/beuty.png Laptop : img/category_img/laptop.jpg Grocery : img/category_img/groccery.jpg
Clothes : img/category_img/pant.png Mobile : img/category_img/mobile.png

```
</div>
</div>
```

Latest product

```
<div class="container-fluid bg-light p-3">
<div class="row"><p class="text-center fs-4">Latest Product</p>
<div class="col-md-3">
<div class="card shadow-sm p-3 mb-5 bg-body-tertiary rounded">
<div class="card-body text-center"><img alt="" src="img/product_img/laptop.jpg" width="65%"height="140px"><p class="text-center">HP Lap</p>
</div></div>
</div>
```

shadow-sm p-3 mb-5 bg-body-tertiary rounded ❼ search for shadow in bootstrap and copy small shadow code
Added these images in img folder ❼ product _img folder
Copy paste same code 3 times again

```
</div>
</div>
```

Thymeleaf :

-used to build dynamic web applications with minimal configuration.
-used to build web pages in Java-based projects
-used to show dynamic data to users
-Thymeleaf templates are valid HTML (or XML) documents that can be opened and viewed in any web browser
-used to add dynamic contents to the web pages like product price, user name, bg image changes
-it is commonly used with spring framework
-Thymeleaf supports template layouts and fragments, enabling developers to
-create reusable components and layouts for web applications. This promotes code reusability and helps maintain consistency across the application.

1. th:text : Replaces the content of an HTML element with the result of an expression.
John Doe ----> If user.name is "Alice", the resulting HTML will be: Alice
<th:block th:text="${@commnServiceImpl.removeSessionMessage()}"></th:block>
2. th:if / th:unless : Conditionally includes or excludes elements based on the evaluation of the expression.
Welcome back! ---> If user.loggedIn is true, this element will be included
Please log in. ---> If user.loggedIn is false, this element will be included
3. th:each : Iterates over collections or arrays and repeats the content for each item.

<li th:each="item : ${items}" th:text="${item.name}">Item

If items is a list of objects with a name property, the resulting HTML will be:

Item 1
Item 2
Item 3

4. th:href : Dynamically sets the href attribute of an a (anchor) element.
<a th:href="@{/users/${user.id}}">Profile ---> If user.id is 5, the resulting HTML will be: Profile
5. th:src : Dynamically sets the src attribute of an img element.
<img th:src="@{/images/${image.name}}" alt="Image">
If image.name is "photo.jpg", the resulting HTML will be: <img src="/images/photo.jpg" alt="Image">
6. th:classappend / th:class : Appends a class to an element or replaces the class attribute dynamically.
Content
If active is true, the resulting HTML will be: Content
7. th:value : Sets the value attribute of an input element dynamically.
<input type="text" th:value="${user.email}" />
If user.email is "alice@example.com", the resulting HTML will be: <input type="text" value="alice@example.com" />
8. th:utext : Similar to th:text, but does not escape HTML tags.
Default bio
If user.bio contains HTML tags like "Bio", they will be rendered as HTML. Bio
9. th:attr : Sets one or more attributes of an element dynamically.
<a th:attr="href=@{/contact}, title=${contactTitle}">Contact us
If contactTitle is "Get in touch", the resulting HTML will be: Contact us
10. **th:inline** : Enables inlining of JavaScript or CSS expressions.

<script th:inline="javascript"> var userName = /*[[${user.name}]]*/ 'John Doe'; </script >
If user.name is "Alice", the resulting JavaScript will be:
<script> var userName = 'Alice'; </script>

<style th:inline="css">
.user-profile {
background-image: url(/images/[[${user.profilePic}]]);
}
</style>
If user.profilePic is "profile.jpg", the resulting CSS will be:
<style>
.user-profile {
background-image: url(/images/profile.jpg);
}
</style>

11. th:remove : Removes elements from the final HTML.
This will be removed ---> This element and its content will not appear in the resulting HTML

Lambok

- Is a Java library that helps to reduce boilerplate code in your Java projects.
- It achieves this by using annotations to automatically generate commonly used methods such as getters, setters, constructors, toString, equals, and hashCode.
- This can significantly simplify the code and make it more readable and maintainable.

Why Lombok is Needed
Reduce Boilerplate Code: Lombok reduces the amount of repetitive code you need to write, such as getters, setters, and constructors.
Improve Readability: By reducing boilerplate, the code becomes more concise and easier to read.
Maintainability: With less boilerplate, the code is easier to maintain and less prone to errors that can occur from manually writing repetitive methods.

Example simple Java class without using Lombok

```
public class User {
    private String name;
    private int age;

    // Constructor
    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter for name
    public String getName() {
        return name;
    }

    // Setter for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter for age
    public int getAge() {
        return age;
    }

    // Setter for age
    public void setAge(int age) {
        this.age = age;
    }

    // toString method
    @Override
    public String toString() {
        return "User{name='" + name + "', age=" + age + '}';
    }

    // equals and hashCode methods
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return age == user.age && Objects.equals(name, user.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

Example using Lombok annotations to reduce boilerplate code:

```
import lombok.Data;

@Data
public class User {
   private String name;
   private int age;

   // Lombok will automatically generate:
   // - Getters for all fields
   // - Setters for all fields
   // - A constructor with all fields
   // - toString method
   // - equals and hashCode methods
}
```

@Data: This annotation is a shortcut for several Lombok annotations: @Getter, @Setter, @RequiredArgsConstructor, @ToString, and @EqualsAndHashCode.
No Need to Manually Write Methods: Lombok automatically generates the constructor, getters, setters, toString, equals, and hashCode methods, significantly reducing the amount of code.

Application.properties

```
spring.application.name=OnlineShopping

spring.datasource.url=jdbc:mysql://localhost:3306/online_shopping
#This property specifies the JDBC URL used to connect to the MySQL database.
#it connects to a database named "online_shopping" running on the local machine (localhost) on port 3306.

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#Specifies the JDBC driver class name

spring.datasource.username=root
spring.datasource.password=password
#These properties provide the username and password for authenticating to the MySQL database.

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
#Specifies the Hibernate dialect to be used.
#The Hibernate dialect determines the specific SQL syntax and configuration options that Hibernate will use for interacting with the database
#Here it is MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update
#it automatically creates a table if it is not exist
```

Run as → spring boot app → localhost:8080/login

Spring Data JPA

JPA stands for Java Persistence API.
It's a Java specification for accessing, persisting, and managing data between Java objects/classes and a relational database.
JPA provides a standardized way to interact with databases using object-oriented paradigms,
reducing the need for developers to write SQL queries manually.**/

Spring Data JPA :
It allows you to define query methods simply by declaring methods in your repository interfaces.
The method names follow a specific pattern which is parsed by Spring Data JPA to generate the corresponding SQL queries.
It generates corresponding SQL queries

findByIsActiveTrue() :
 --findBy : prefix indicates we are performing query to retrieve data from db
 --IsActive : name of the field in the Category entity
 --True : isActive field should be true
When the method findByIsActiveTrue is called, Spring Data JPA translates it into a SQL query :SELECT * FROM category WHERE is_active = true;
This query retrieves all records from the category table where the is_active column has the value true

existsByName(String name) :
 --existsBy: This prefix indicates that we are performing an existence check.
 --name: This is the name of the field in the Category entity.
checks if Category entity with a given name exists in the database

Translation : SELECT
CASE WHEN COUNT(c) > 0
THEN TRUE ELSE FALSE
END FROM category c WHERE c.name = ?1;

save(category) : save if new else update if old
findAll() : SELECT * FROM category; This query retrieves all records from the category table.
findById(id) : SELECT * FROM category WHERE id = ?;
delete(category) : Spring Data JPA will translate to DELETE FROM category WHERE id = :id

Spring Security

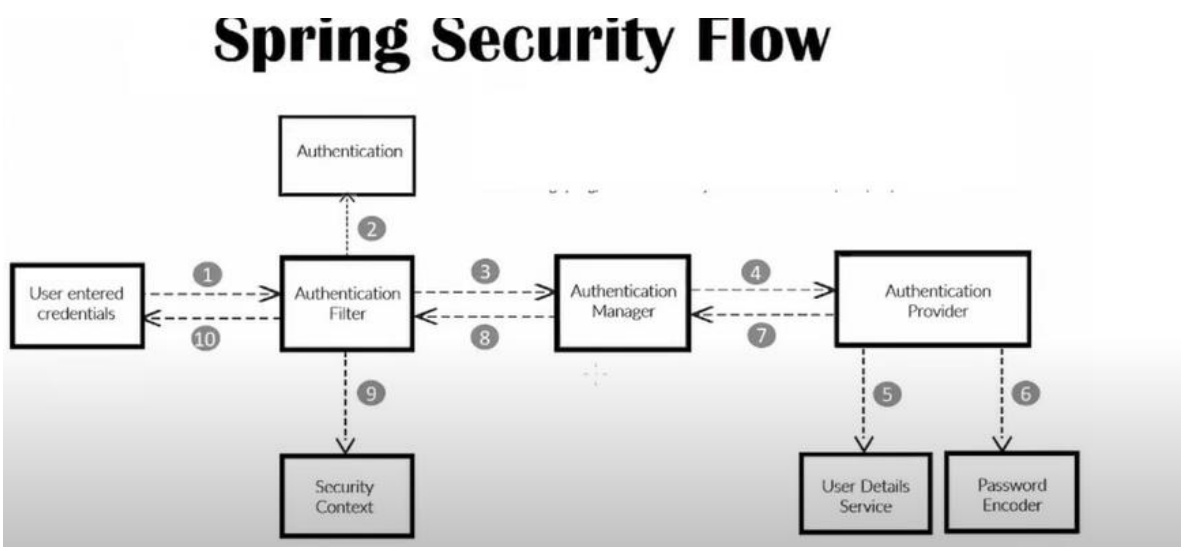
Spring Security is a powerful and highly customizable authentication and access control framework for the Spring framework

- **Authentication**: This is the process of verifying the identity of a user. (if username and password matches with the db)
  In Spring Security, this is represented by the Authentication interface. A common implementation is UsernamePasswordAuthenticationToken.
- **Authorization**: This is the process of deciding whether a user is allowed to perform an action or access a resource. It is handled by the AccessDecisionManager.
- **Security Context**: Holds the authentication information of the current user. It is stored in a SecurityContextHolder.
- **UserDetailsService**: An interface that provides a method to load user-specific data. It is used to fetch user details like username, password, and authorities.
- **PasswordEncoder**: An interface used for encoding passwords. BCryptPasswordEncoder is a commonly used implementation.

1. **Add Spring Security Dependency**
   ```
   <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
   ```
   Google → Spring io → initializer → choose maven → click on add dependency , search security, choose Spring Security → explore → copy the dependency → add in pom.xml → maven update project

2. **Add role in User model**
3. **Create custom user and implement UserDetails : set role, email, password**
   Create com.shopping.config package → create CustomUser.class → implements UserDetails (it is interface) → override methods
   Source → generate constructor using field → change this
   ```
   Override
   public String getPassword() {
       return user.getPassword();
   }

   @Override
   public String getUsername() {
       return user.getEmail();
   }
   @Override
   public Collection<? extends GrantedAuthority> getAuthorities() {
       SimpleGrantedAuthority authority = new SimpleGrantedAuthority(user.getRole());
       return Arrays.asList(authority);
   }
   ```
4. **Create UserDetailsServiceImpl and implement UserDetailsService**
   ```
   @Service
   public class UserDetailsServiceImpl implements UserDetailsService {
       @Autowired
       private UserRepository userRepository;
       @Override
       public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
           UserDtls user = userRepository.findByEmail(username);
           if (user == null) {throw new UsernameNotFoundException("user not found");}
           return new CustomUser(user);
       }
   }
   ```
5. **Create SecurityConfig - Create a Security Configuration Class**
   In Spring Security 6.0, WebSecurityConfigurerAdapter is deprecated. Instead, you should use a SecurityFilterChain bean and configure HTTP security directly
6. **Create AuthSuccessHandler**



SecurityFilterChain: Intercepts requests and applies security filters.
AuthenticationManager: Delegates authentication requests to AuthenticationProvider implementations.
AuthenticationProvider: Performs the actual authentication logic.
UserDetailsService: Loads user-specific data.
PasswordEncoder: Encodes and verifies passwords.
SecurityContextHolder: Stores the security context for the current request.
AccessDecisionManager: Makes authorization decisions based on votes from AccessDecisionVoter.

Client Request: A user requests a protected resource, e.g., /profile.
Security Filter Chain: The request passes through the security filters.
Authentication Filter: If the user needs to log in, the UsernamePasswordAuthenticationFilter processes the login form.
AuthenticationManager: The credentials are passed to the AuthenticationManager.
AuthenticationProvider: The DaoAuthenticationProvider uses the UserDetailsService to fetch user data and PasswordEncoder to verify the password.
Successful Authentication: On success, an Authentication object is stored in the SecurityContextHolder.
Authorization: The AccessDecisionManager checks if the authenticated user has access to the resource.
Resource Access: If authorized, the request is processed and the resource is returned to the user.
Post-Processing: The security context is cleared, and any other necessary cleanup is performed.