**Full Stack Ecommerce Website Development Using React, Spring Boot, MySql And Payment Gatway**

**Technologies Used:** **React, Spring Boot, MySQL, And Payment Gateway (RazorPay)**

**frontEnd : react , Backend : SpringBoot , Payment gateway : Razor Pay**

**Customer**

* Carousel
* sorting feature,
* pagination,
* filter,
* add to cart,
* buy,
* payment gateway,
* track order,
* review and rate product

**Admin**

* admin panel,
* add product,
* manage order,
* change order status

**Create React App Using :** *npx create-react-app project\_name or we can use npx create-react-app . (. Is used to current folder makes our project ) if we want to create new projet folder we can use npx create-react-app project\_name this fomat*

**Downloads :**

**react alice carousel (2.7.1)**

**Mui :** *npm install @mui/material @emotion/react @emotion/styled*

**Mui – icon :** *npm install @mui/icons-material @mui/material @emotion/styled @emotion/react*

**Tailwand css :**

* + - *npm install -D tailwindcss@3*
    - *npx tailwindcss init :* ***Created Tailwind CSS config file: tailwind.config.js***

**tailwind.config.js**

*/\*\* @type {import('tailwindcss').Config} \*/*

*module.exports = {*

*content: [*

*"./src/\*\*/\*.{js,jsx,ts,tsx}",*

*],*

*theme: {*

*extend: {},*

*},*

*plugins: [],*

*}*

Add This code In **tailwind.config.js** file

**index.css**

*@tailwind base;*

*@tailwind components;*

*@tailwind utilities;*

Add This code In **index.css** file

**Now You can Start Your Project Using :** *npm start*

*30/4/25*

**\src\customer\components\navigation\Navigation.jsx**

**Navigation.jsx**

**Navigation Bar :** [*https://tailwindcss.com/plus/ui-blocks/ecommerce/components/store-navigation*](https://tailwindcss.com/plus/ui-blocks/ecommerce/components/store-navigation)

(Use This link To get readyMade Navbar From Tailwind Doc )

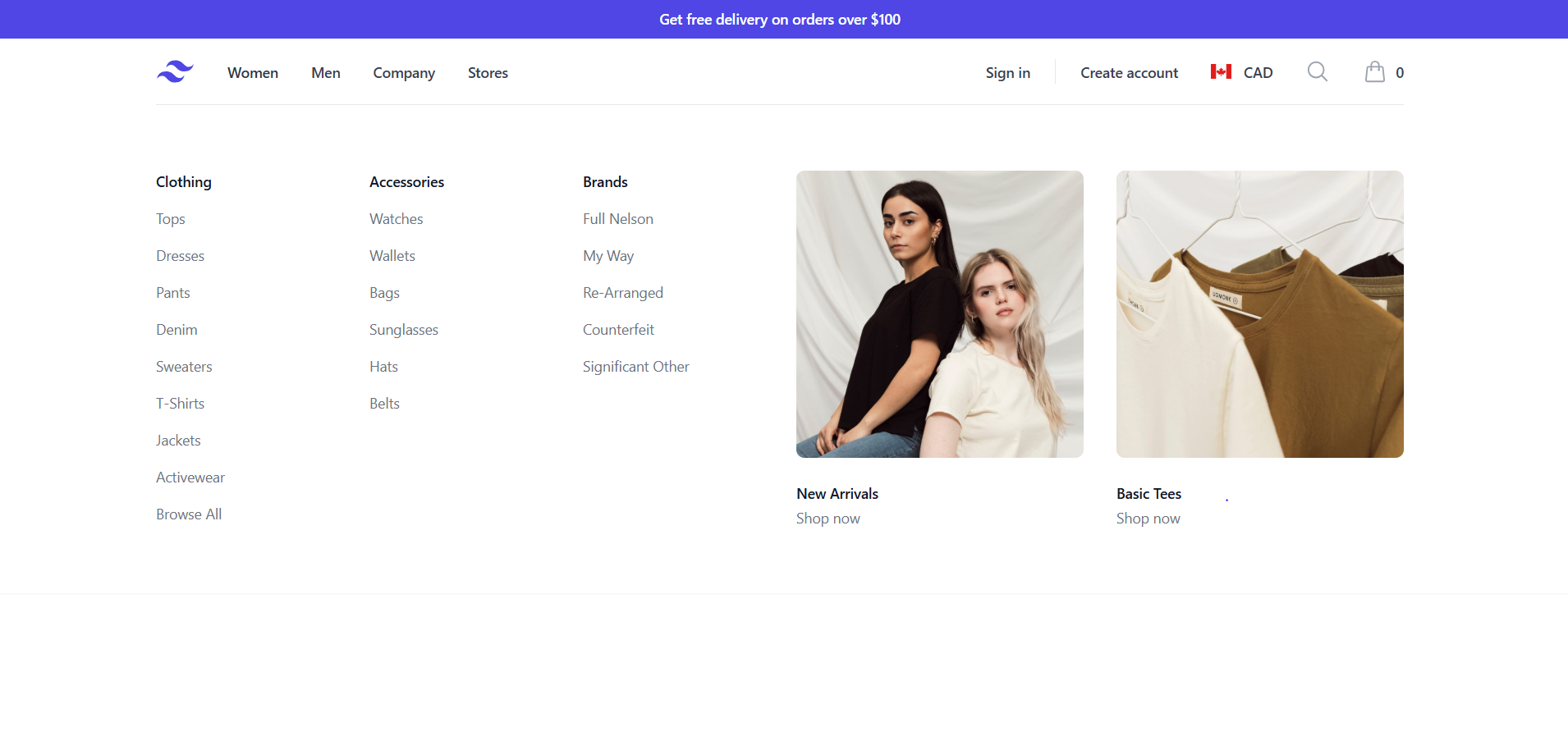
### *npm install @heroicons/react :* ( Icons for UI This library provides SVG-based icons (like shopping carts, chevrons, user profiles, etc.) that you can use directly in React components.)

### *npm install @headlessui/react :* ***(***Accessible UI components (without styles)

This library gives you interactive components like:

* Dialogs (modals)
* Dropdowns
* Tabs
* Menus
* Disclosures (accordions)

But without any built-in styling, you can fully customize them (usually with Tailwind CSS).



So Navbar look like this

Home Section :

*react-alice-carousel* : is a React carousel/slider library used to create image sliders, carousels, or galleries in React applications.

**Key Features :**

Responsive — works across devices and screen sizes

* Touch-friendly — supports swipe/drag gestures on mobile
* Autoplay support
* Infinite loop
* Custom buttons/dots
* Lightweight and easy to integrate

**Download Templete :** <https://maxmarinich.github.io/react-alice-carousel/#basic>

\src\customer\components\HomeCarousel\MainCarousel.jsx

Add this template code in -> **MainCarousel.jsx**

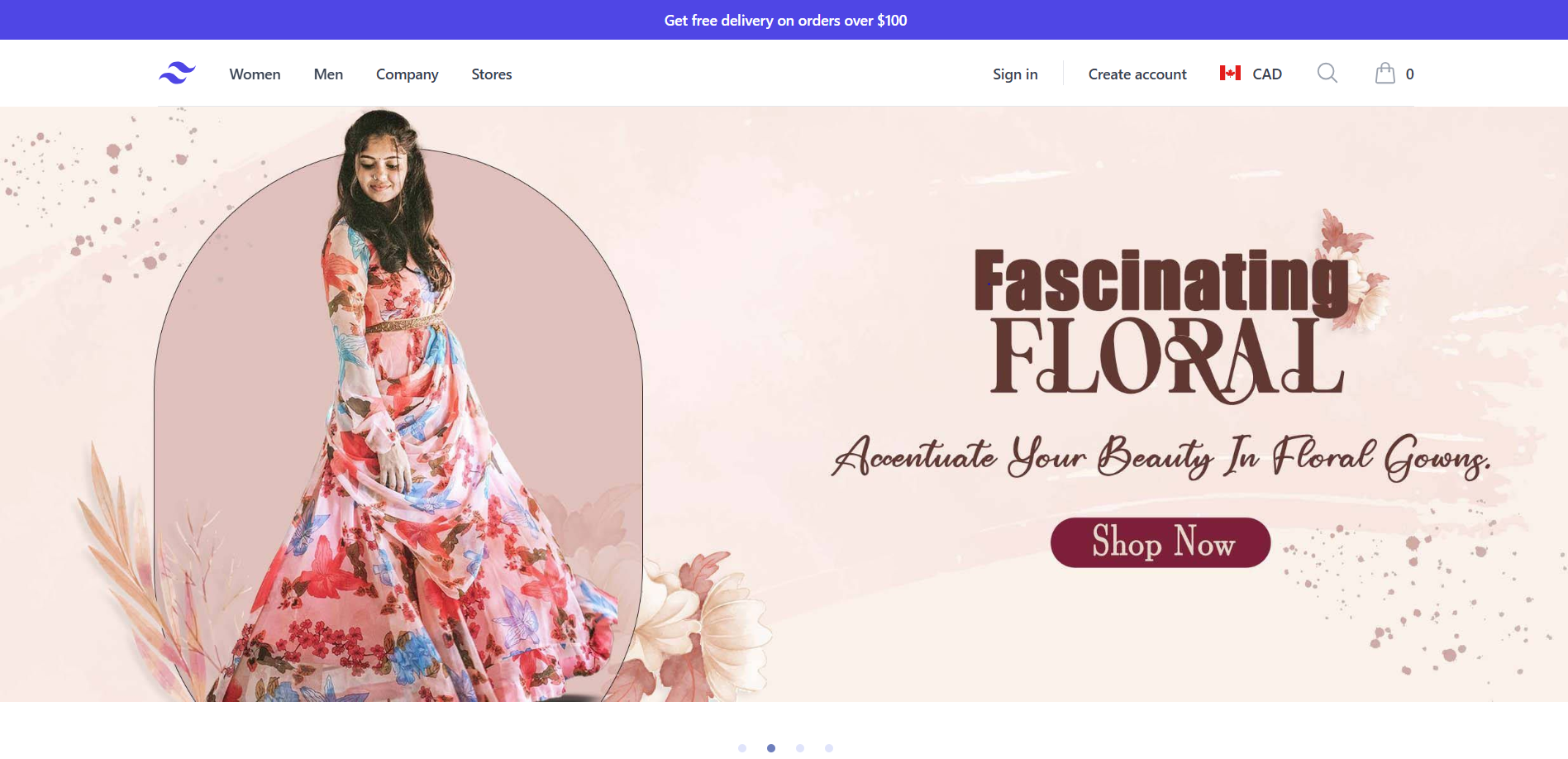
\src\customer\Pages\HomePage\HomePage.jsx

**HomePage.jsx :** importing **MainCarousel.jsx** in **HomePage.jsx** AndAfter import **HomePage.jsx** in **App.js**

\src\customer\components\HomeCarousel\ MainCarouselData.js

const items = mainCarouselData.map((item) => <img className='cursor-pointer' role='presentation' src = {item.image} alt = "" />)

Iterating in **MainCarousel.jsx** so The images present in that file will display on Home Page And Using Carousel it will Auomatically slide.When We Set some properties in this .like autoplay , autoplayInterval etc.



**Creating Home Section Card :**

\src\customer\components\HomeSectionCard\**HomeSectionCard.jsx**

**Creating Carousel**

\src\customer\components\HomeSectionCarousel\**HomeSectionCarousel.jsx**

Importing **HomeSectionCard.jsx** in **HomeSectionCarousel.jsx** and Then It will imported in Main **HomePage.jsx**

Some Problem with **HomeSectionCarousel** The slide is not working properly

**Data Files**

\src\Data\**mens\_kurta.js**

\src\customer\components\navigation\**NavigationData.js**

**Creating Product Page**

\src\customer\components\Product

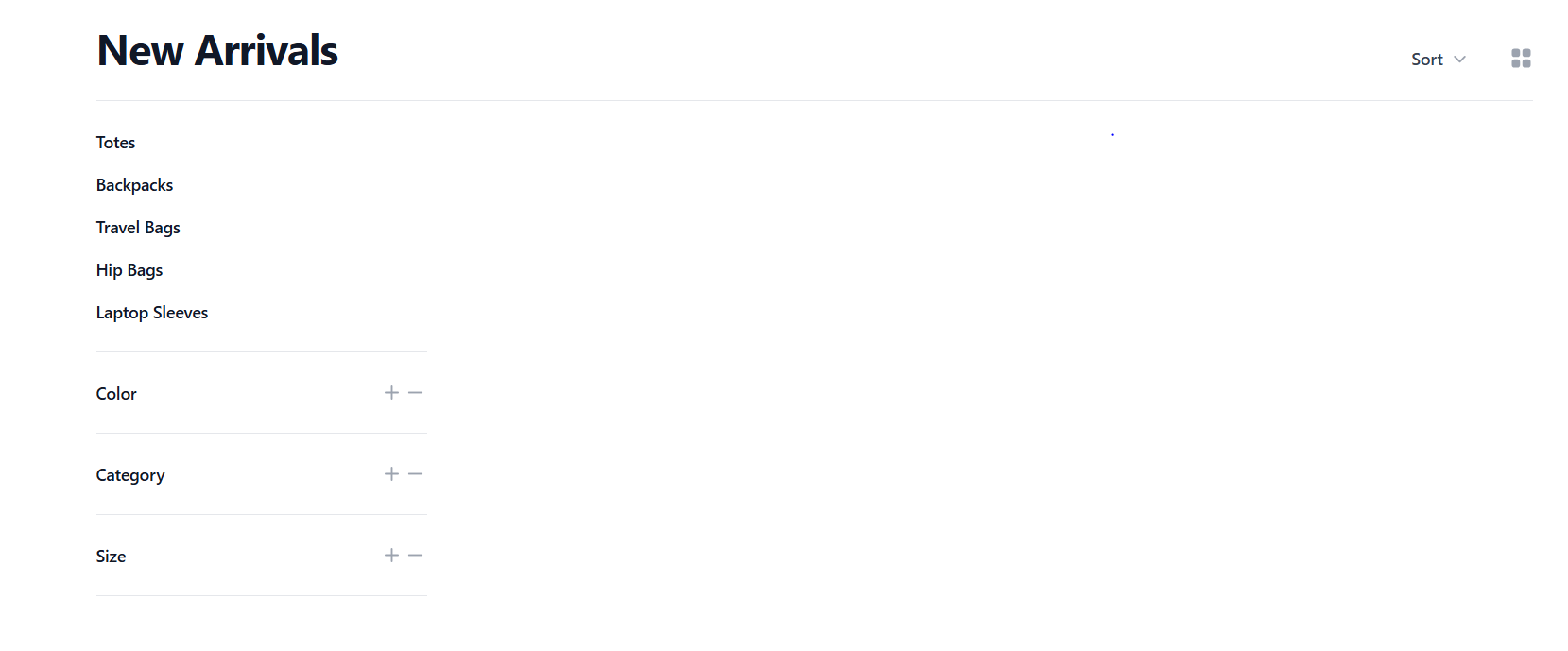
\src\customer\components\Product\**Product.jsx**

\src\customer\components\Product\**ProductCard.jsx**

\src\customer\components\Product\**ProductCardCSS.css**

Calling ProductCard.jsx in Product.jsx and after Product.jsx Add in App.js

Adding tailwind component in product.jsx : <https://tailwindcss.com/plus/ui-blocks/ecommerce/components/category-filters>



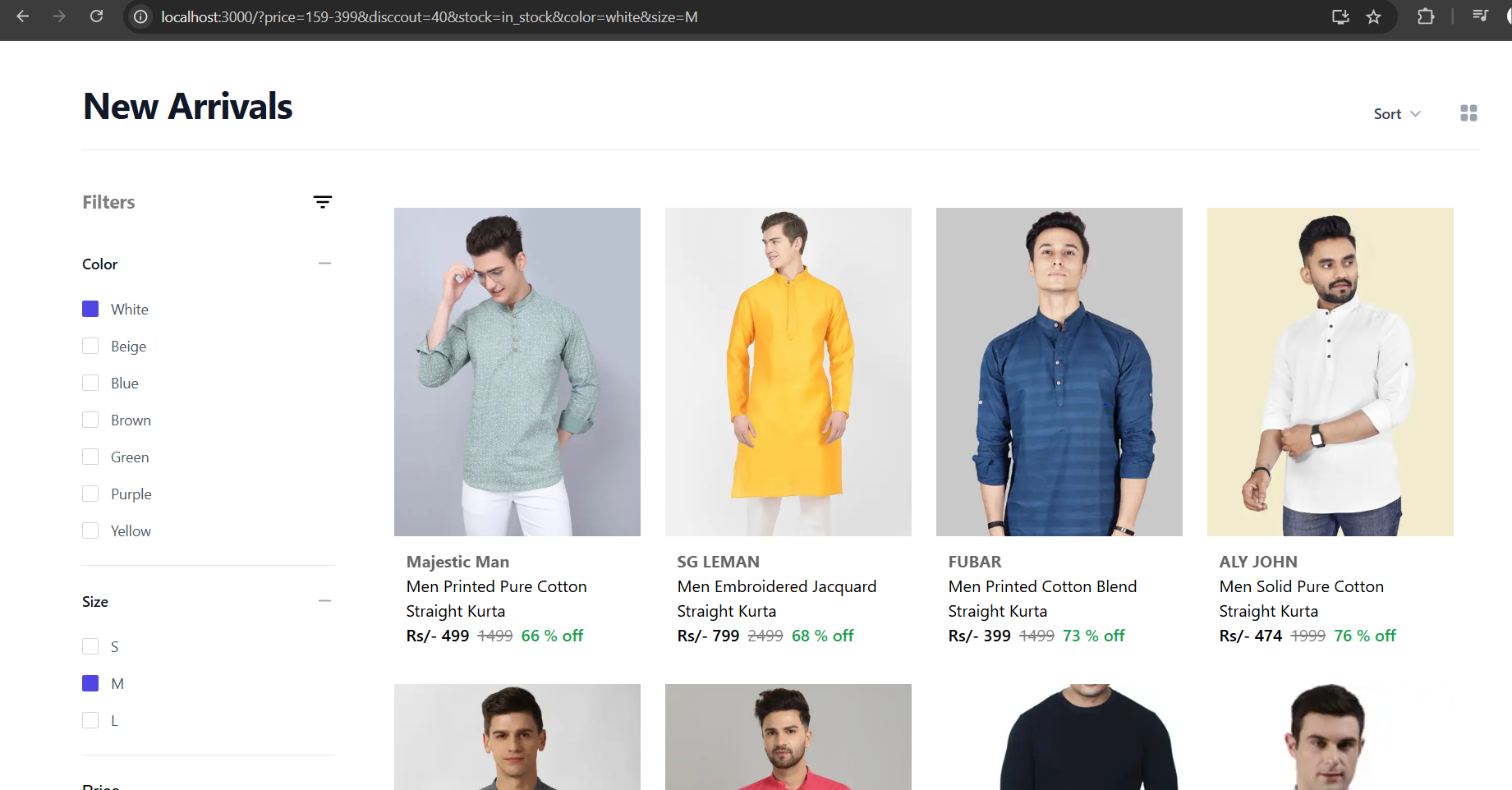
It Look Like This above image

Install : *npm i react-router-dom*

**What is react-router-dom?**

react-router-dom is a library that enables client-side routing in React applications. It allows you to:

* Navigate between different components/pages without refreshing the page.
* Create single-page applications (SPAs).
* Define routes and link components to different URLs.



Now The final look Product Page Looks Like This

Product Details Page

\src\customer\components\ProductDetails Creating Folder

\src\customer\components\ProductDetails\**ProductDetails.jsx** Creating jsx file

Import **ProductDetails.jsx** in **App.jsx**

**ProductDetails.jsx** code *:* [*https://tailwindcss.com/plus/ui-blocks/ecommerce/components/product-overviews*](https://tailwindcss.com/plus/ui-blocks/ecommerce/components/product-overviews) *and done some modification by own*

1.Image Gallery

2. Product Info

* Option (Info About Product )
* Review
* Sizes
* Rating And Review
* Similir Product

\src\customer\components\ProductDetails\**ProductReviewCard.jsx**

Cart Page

\src\customer\components\Cart

\src\customer\components\**Cart.jsx**

\src\customer\components\**CartItems.jsx**

Add cart.jsx in App.js

CheckOut Page

\src\customer\components\Checkout Main Folder

**SubFiles:**

\src\customer\components\Checkout\**Checkout.jsx :**

Code: https://mui.com/material-ui/react-stepper/

\src\customer\components\Checkout\**DeliveryAddressForm.jsx**

\src\customer\components\Checkout\**OrderSummary.jsx**

\src\customer\components\AddressCard **Main Folder**

\src\customer\components\AddressCard**\AddressCard.jsx**

**\src\customer\components\Order Main Folder**

**\**src\customer\components\Order\Order.jsx

\src\customer\components\Order\OrderCard.jsx

\src\customer\components\Order\OrderDetails.jsx

\src\customer\components\Order\OrderTracking.jsx

**\src\Routers\ Main Folder**

\src\Routers\**CustomerRouters.jsx**

**Back-End**

Spring Security

JWT

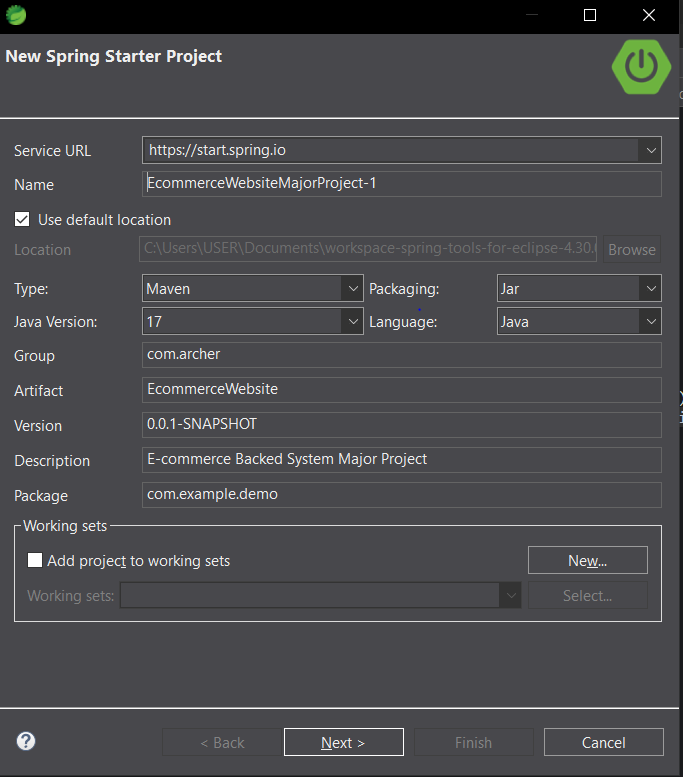
Sign up

Sign ins

Install Spring tool Zip From Google : [*https://spring.io/tools*](https://spring.io/tools)

Extract that zip and run *SpringToolSuite4*

Then it will open STS IDE Create a one project in IDE Using Spring Starter Project



This window will be displayed. Fill in all project details and click on next, and then finish

**application.properties :**

* + Database configuration
* spring.datasource.url=jdbc:mysql://localhost:3306/mydb
* spring.datasource.username=root
* spring.datasource.password=Root
* spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
  + Server port
* server.port=5455

**Appconfig.java :**

**@Bean**

**SecurityFilterChain() :**

In a Spring Security-based Java application, a SecurityFilterChain is a core component used to define the security rules and filters app

You're telling Spring Security how to secure your web application. This bean runs when your app starts, and it controls:

* Who can access which URLs
* How JWT tokens are validated
* How cross-origin requests (CORS) are handled
* Whether sessions or login forms are usedlied to incoming HTTP requests**.**

1. **Stateless Session – No Server Memoryry**

*http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)*

You tell Spring:

"Don't use server sessions. I’ll use JWT tokens instead."

2. **Which Routes Need Login?**

**.***authorizeHttpRequests(auth ->*

*auth.requestMatchers("/api/\*\*").authenticated() // 🔐 These need JWT*

*.anyRequest().permitAll() // 🌐 Everything else is public*

*)*

"/api/\*\*" – Protected routes. User must have a valid token.

All other routes (like /login, /about, /home) are open to everyone.

3. **Use Your JWT Filter**

*.addFilterBefore(new JwtValidator(), BasicAuthenticationFilter.class)*

You add a custom filter called JwtValidator that:

Reads the token from headers

* Validates it
* If it’s valid, tells Spring "Yes, this user is logged in"
* This runs before Spring’s built-in filters.

**4. Disable CSRF**

*.csrf().disable()*

"I’m building an API with JWT, not a form-based login. I don’t need CSRF protection."

5. **Allow Frontend to Call Your API (CORS)**

*.cors().configurationSource(...) // a big block*

This part tells Spring:

"Let my React app (localhost:3000) and Angular app (localhost:4200) talk to my backend, even though they are on different ports."

It allows:

* All HTTP methods (GET, POST, etc.)
* All headers (especially Authorization)
* Credentials (cookies, tokens, etc.)

**6. Form Login and Basic Auth (Not Needed)**

*.and().httpBasic().and().formLogin()*

You're saying:

"Also allow login using the default Spring login page and Basic Auth."

But:

You don’t need these if you're using only JWT.

**Final Summary**

|  |  |  |
| --- | --- | --- |
| Feature | What it does | Keep it? |
| Stateless | No sessions; use tokens | ✅ Yes |
| URL rules | Protect /api/\*\*; open others | ✅ Yes |
| JWT filter | Validates token from header | ✅ Yes |
| CSRF disabled | Fine for JWT-based APIs | ✅ Yes |
| CORS config | Lets frontend call your backend | ✅ Yes |
| Form login / Basic | For username/password or Spring login page | ❌ No (for pure JWT apps) |

*@Bean*

*public PasswordEncoder passwordEncoder() {*

*return new BCryptPasswordEncoder();*

*}*

**What is BCryptPasswordEncoder?**

BCryptPasswordEncoder is a built-in encoder in Spring Security that:

* Hashes (encrypts) passwords securely
* Adds a random salt to each hash (protects against rainbow table attacks)
* Uses a work factor (CPU cost) to slow down brute-force attempts

It's the recommended way to store user passwords in modern web applications.

**Why Do You Need This?**

* Storing raw passwords is dangerous and insecure. You should:
* Encrypt a password when the user registers
* Verify the hash when the user logs in
* Spring uses this PasswordEncoder to do both.

**Example: Register and Login Flow**

**1. Registration**

*String rawPassword = "hello123";*

*String hashedPassword = passwordEncoder.encode(rawPassword);*

// Save hashedPassword to the database

**2. Login**

*boolean match = passwordEncoder.matches(rawPassword, storedHashedPassword);*

**Why Use @Bean**?

*@Bean*

*public PasswordEncoder passwordEncoder() {*

*return new BCryptPasswordEncoder();*

*}*

You define this as a bean, so Spring can automatically inject it anywhere you need it, like in your:

* UserService
* Authentication logic
* Custom UserDetailsService

Spring will reuse the same secure encoder everywhere.

**Without This**

If you don’t define a PasswordEncoder, Spring will either:

Use no encoding (unsafe), or

Throw an error when it tries to compare passwords

\src\main\java\com\example\model Main Folder

\src\main\java\com\example\model\User.java

Annoations Used To Define The User Entity/class

Why Each One Matters

**@Entity**

Tells Spring Boot this class is a persistent entity.

It will map to a table named user by default.

**@Id and @GeneratedValue(...)**

**@Id: The field is the primary key.**

**@GeneratedValue:** Auto-generates IDs using your database’s preferred method (identity/sequence/table).

**@OneToMany**

Used for relationships like:

* A user can have many addresses
* A user can post many reviews or ratings

You’re also using:

**mappedBy = "user" →** This means the Address, Rating, or Review entity has a user field annotated with @ManyToOne.

**cascade = CascadeType.ALL** → Any operation (persist, delete, etc.) done on User will cascade to child entities.

**@Embedded, @ElementCollection, and @CollectionTable**

This combination is used to store a list of value objects (PaymentInformation) that are not full JPA entities.

* **@Embedded:** It’s a class whose fields will be part of another entity’s table or a separate table.
* **@ElementCollection:** It's a collection of embeddables, not a relation to another entity.
* **@CollectionTable:** Defines the table name for this list (paymet\_information) and sets up the foreign key (user\_id).

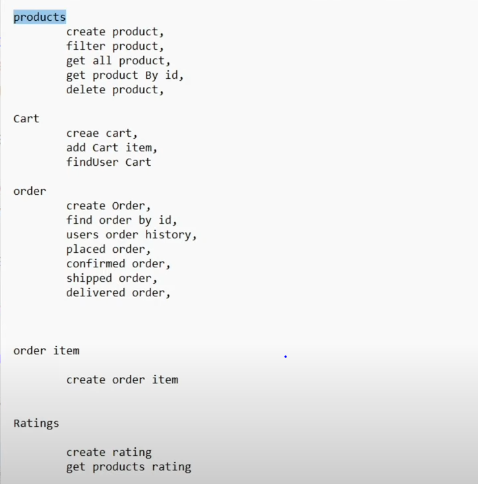
Typo alert: paymet\_information → should probably be payment\_information.

**@JsonIgnore**

* Prevents Jackson from serializing these fields into JSON (e.g., when returning the user as an API response).
* Helps avoid circular references and large, nested JSON structures.



Creating All This Entities.



UserController : users/profiles request

AdminProductController: api/admin/product

ApiResponse :