Project shorty shoes

The objetive of this Project was to create na application where there were 2 types of users, admins and customers. The admins can add,update and delete products and customers can see those products and can order them. There is possible to create new accounts into the application and to login. Admin user can change his password too if he wants.

Login

The first step was to create a Login entity with all the Login data we want and the setters and getters.

On the Login entity we needed to add @Entity to make possible the connection with the JPA and to automatically create a table with the Login Data.

It was used the @Id to make emailid as a primary key to table.

It was created an LoginRepository with 3 methods:

-findLoginByEmail that is a select to verify if the emailid received is already in bd or not to allow a new signUp

-findLoginByEmailAndPassword that is a select to verify if the emailid and emailpassword match any column on the bd to make sure that the signIn is possible or not

-updateAdminPassword that is an update to make possible the admin to change is password.

It was necessary to use @Repository to make possible to use ,store or update any data

The @Query is used before our sql query.

After this, was created the LoginService that have @Service to that indicate this class have the application Logic.

It was used the @AutoWired LoginRepository to make the injection of the LoginRepository into out bean.

At the LoginService class where created 4 methods:

-init to save the admin data on BD. It was used the @PostConstruct to make sure this method is executed before the rest.

-signIn that receives an Login and verify if the Login exists and if it is admin or customer.

-signUp that receives an Login and verify if the email already exist or not to check if it is possible to create a new client.

-updateAdminPassword that receives a new password and updates on BD.

After that it was created the class LoginController with @Controller to indicate that this class will have web requests.

It was used the @AutoWired LoginService and ProductService to make the injection of the LoginService and ProductService into out bean.

It was created a @GetMapping(value=”shorty-shoes/login”) to make a rest request with method get to inicialize the login page and return the html file Login that have the html code for the login page.

It was created a @PostMapping for the sigIn to make a rest request with method post for the sigIn. Here we call the service login.signIn to verify if the Login received is from an admin or a customer and return to Admin html file or Customer html file. If the Login doesn’t exist, it will return to Login page. In this method it was used the Model to make possible to add attributes like buttons or messages and show all the products that exist in the BD.

It was created a @PostMapping for the signUp to make a rest request with method post for signUp. Here we call service loginService.signUp to verify if the email already exist or not in the bd and show a message before return to Login page.

It was created a @PostMapping for the updateAdmin to make a rest request with method post that calls loginService.updateAdminPassword that will update the admin password at the bd. This method is only called in the admin page.

Product

As said before, it was created an entity Product with all Product data and setters and getters and used @Entity to make possible the connection with the JPA and @Id to make pid our primary Key.

It was created the ProductRepository with 2 methods. The first one (findProductByNameAndModel) to verify if the product that we are trying to add already exist in the BD.

The second one ( filterProduct ) to make possible for the customer to filter the products he want to see. Both are @Query select

It was created the ProductService @autoWired ProductRepository to to make the injection of the ProductRepository into out bean and @Service to indicate that this class have application logic.

It were created 6 methods:

-storeProduct that validate if the product exist in the bd and if not, save the product.

-deleteProduct that receives a pid and verify if the pid exist in bd. If yes, the column is deleted.

-updateProduct that verifies if the pid exist in the bd and if yes, update all the data that receives.

-findAll to recturn a list of all products that exist in the bd

-searchProductById that receives a pid and verify if the pid exist in the bd. If yes, return all data from that product.

-filterProducts that will return the list of the products that match the data received.

After that it was created the ProductController with @Controller to indicate that this class will have web requests and @autoWired ProductService to inject ProductService into our bean.

It was created an @PostMapping addProduct that calls productService.storeProduct to save new products and provide success/error messages. This is only used for the user admin.

It was created @GetMapping deleteProduct that calls deleteProduct to delete a product

It was created @GetMapping udpateProductIngo that calls searchProductById to update products.

It was created @filterProducts to filter the products. Used for Customer.

Transaction

The first step was to create a Transaction entity with all the Transaction data we want and the setters and getters.

It was created the transactionService to insert a transaction with the product that customer wants to buy and a findAll to return all buys.

It was created the TransactionController with @GetMapping that inserts a Transaction when the customer clicks “Buy” on a Product.