Make an E-commerce Website for Sporty Shoes

This document contains the following.

- Project and developer details
- Sprint planning and tasks achieved
- Core concepts used in the project
- Flowchart of the application
- Links to the GitHub repository

Project objective:

As a Full Stack Developer, complete the features of the application by planning the development and pushing the source code to the GitHub repository.

Background of the problem statement:

Sporty Shoes is a company that manufactures and sells sports shoes. They have a walk-in store, and now, they wish to launch their e-commerce portal sportyshoes.com.

You're asked to develop a prototype of the application. It will be then presented to the relevant stakeholders for budget approval. Your manager has set up a meeting where you're asked to do the following:

- Presenting the specification document which has the product's capabilities, appearance, and user interactions
- Setting up Git and GitHub account to store and track your enhancements of the prototype
- Explaining the Java concepts used in the project
- Discussing the generic features of the product:
- There will be an admin to manage the website. An administrator login will be required to access the admin page.

The admin should be able to change his password if he wants, he should be able to:

- Manage the products in the store including categorizing them
- Browse the list of users who have signed up and be able to search users
- See purchase reports filtered by date and category

Sprint planning and tasks achieved

Sprint 1:

I. Create a welcome page or home page displaying SportyShoes and name of developer.
II. Take user input to select one of the options from the navbar.
III. Clicking on Admin login tab will redirect to admin login page for asking input as username and password.

Release product for feedback over the frontend part of home page and admin login page.

Sprint 2:

After successful admin login -

I. Add options to admin login page for further activities like
    a. Change password and either continue on page or logout and login again with new credentials.
    b. Managing products available in store, create/remove categories, create/remove products from existing categories.
    c. Check list of signed up clients of website and search any particular client by name or id.
    d. See purchase report filtered by date and category.
II. Create webpage for the above-mentioned activities and interlink them as necessary.

Release product for feedback with all added webpages.

Sprint 3:

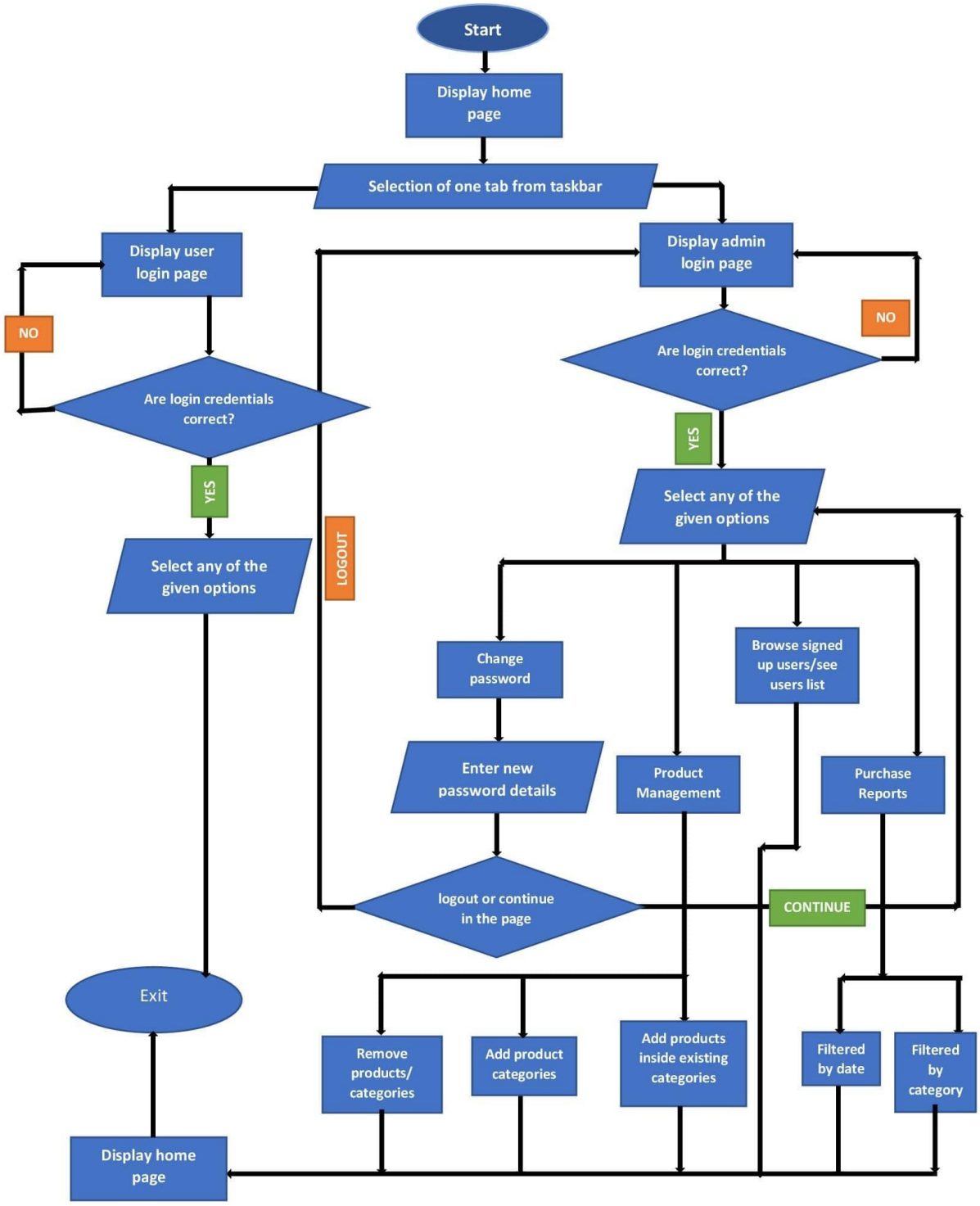
Create a program to store all the inputs given by the admin to a database (assigned to the SportyShoes). Develop relation between frontend developed in sprint2 with the database to store and retrieve data dynamically.

Release the product for feedback with all required database activities.

Flow of Application:

1. Welcome screen with name of website and developer.
2. Ask for user input to select admin login or user login.
3. For admin login, allow user to select one of the following options.
    a. Change password and either continue on page or logout and login again with new credentials.
    b. Managing products available in store, create/remove categories, create/remove products from existing categories.
    c. Check list of signed up clients of website and search any particular client by name or id.
    d. See purchase report filtered by date and category.
4. For client login, allow client to surf for available options.
    a. Select options to choose categories/products and add them to cart.
    b. Make checkout option to buy products added in cart.
5. Display home page or exit the website.

Flow Chart


Start
Display home page
Selection of one tab from taskbar
Display user login page
Display admin login page
NO
Are login credentials correct?
NO
Are login credentials correct?
YES
Select any of the given options
LOGOUT
Select any of the given options
Change password
Browse signed up users/see users list
Enter new password details
Product Management
Purchase Reports
logout or continue in the page
CONTINUE
Exit
Remove products/ categories
Add product categories
Add products inside existing categories
Filtered by date
Filtered by category
Display home page

Pushing the code to GitHub Repository:

- Open Git Bash and navigate to the folder where you have created your files.

  cd E-commerce \MahendraPhase3-SecondProject\E-commerce website
- Initialize the repository using the below command **git init**
- Add all the files to your git repository using the below command **git add .**
- Commit the changes using the below command **git commit -m "Initial commit"**
- Add the URL for the remote repository where your local repository will be pushed **git remote add origin https://github.com/Mahendra1272/E-commerce .git**
- Push the files to the folder you initially created using below command **git push -u origin master**

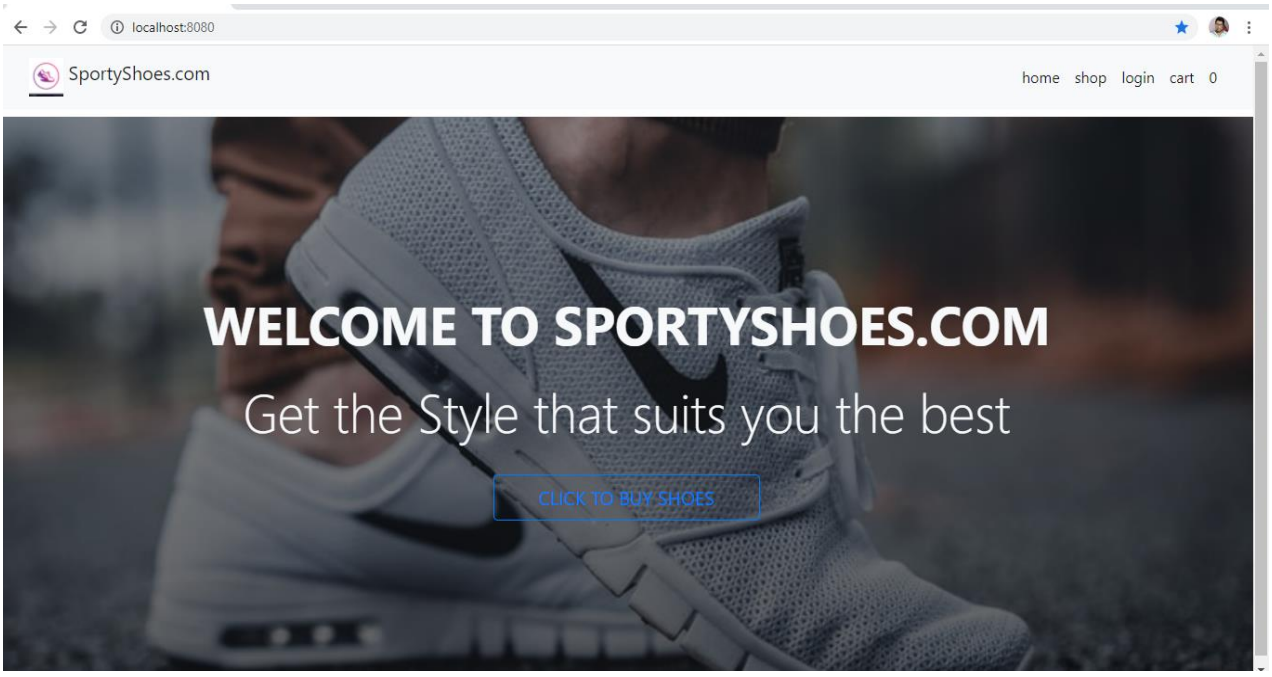
Links to the GitHub repository:

https://github.com/Mahendra1272/E-commerce.git

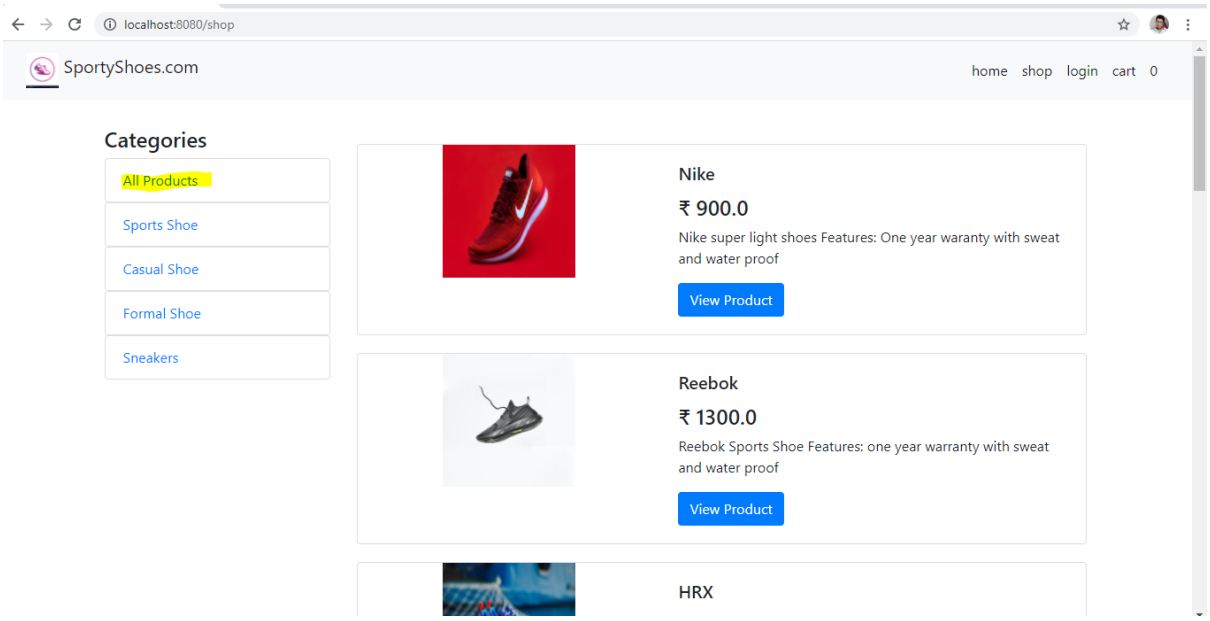
Screenshots of all Scenarios in the Application:

1. **Customer Related Scenarios**

Homepage displays options to users where they can login (if they already have an account or Sign Up if they are new users) , users also have an option to shop where they can shop for shoes and add to cart, they can also click on Click to Buy button which will redirect them to shop page to buy products.



In Shop page user will see list of All products if the category is not selected from the sidebar, User can also select individual categories and purchase products from there



Sneakers

Reebok

₹ 1300.0

Reebok Sports Shoe Features: one year warranty with sweat and water proof

View Product

HRX

₹ 1900.0

HRX Sweat Proof Shoe Features: Features: one year warranty with sweat and water proof

View Product

WoodLand

₹ 3000.0

WoodLand Casual Shoe Features: one year warranty with sweat and water proof

localhost:8080/shop

Reebok Air

₹ 3000.0

Super Light Reebok Air Collection Features: one year warranty with sweat and water proof

View Product

Lakhani Office Shoe

₹ 800.0

Lakhani Office Collection Features: one year warranty with sweat and water proof

View Product

Puma Formal Shoe

₹ 1900.0

Puma Formal Collection Features: one year warranty with sweat and water proof

localhost:8080/shop

Puma Formal Shoe

₹ 1900.0

Puma Formal Collection Features: one year warranty with sweat and water proof

View Product

Reebok

₹ 2900.0

Reebok Formal Collection Features: one year warranty with sweat and water proof

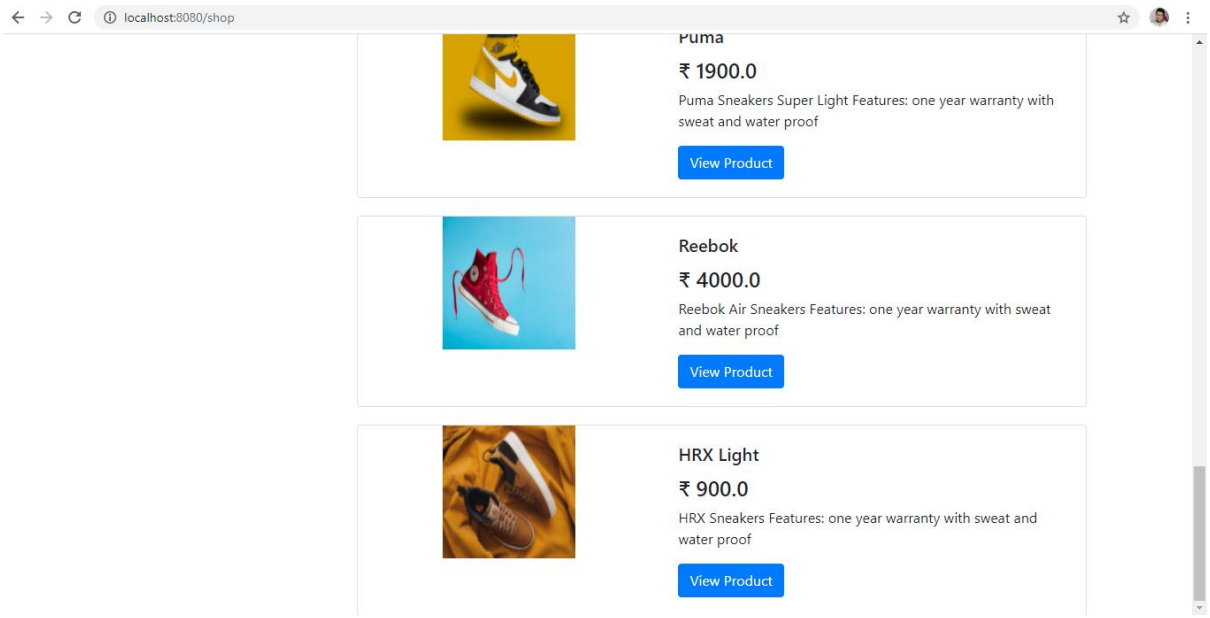
View Product

Puma

₹ 1900.0

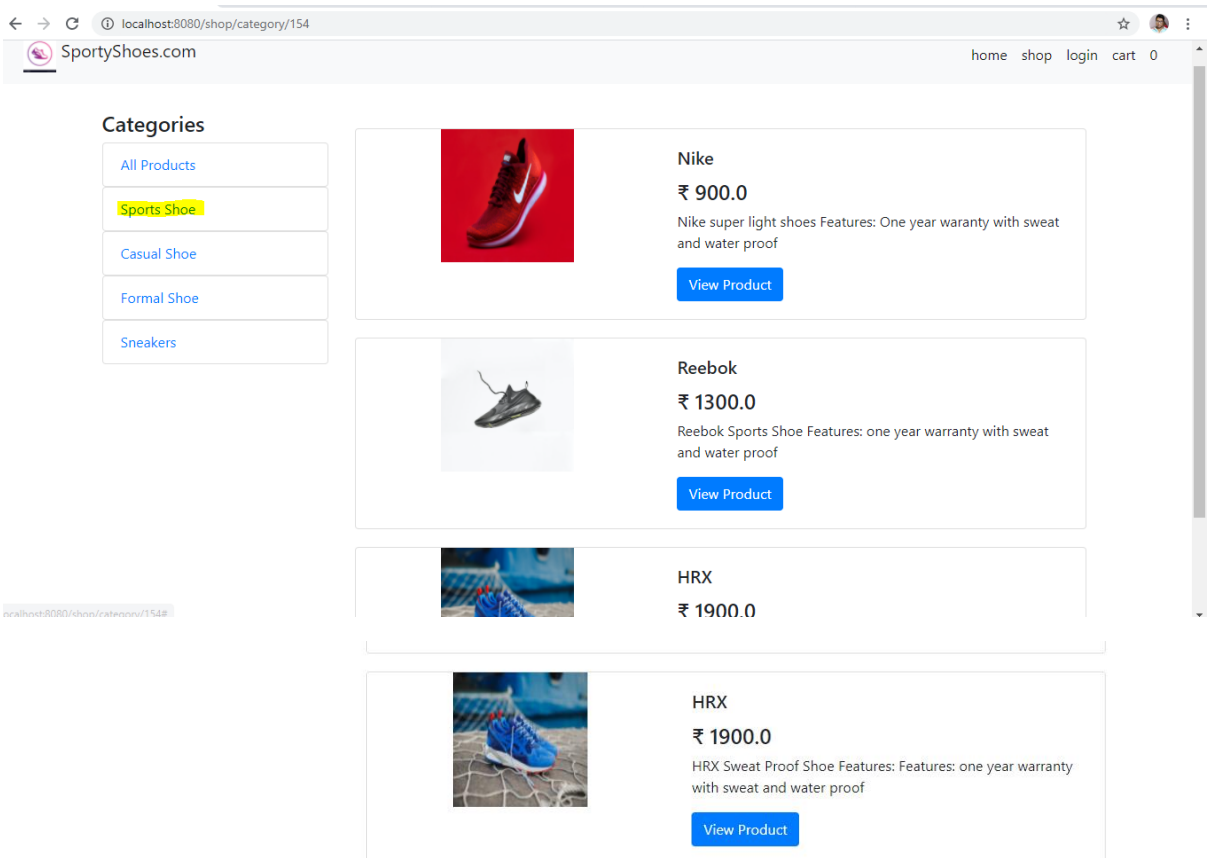
Puma Sneakers Super Light Features: one year warranty with sweat and water proof

View Product

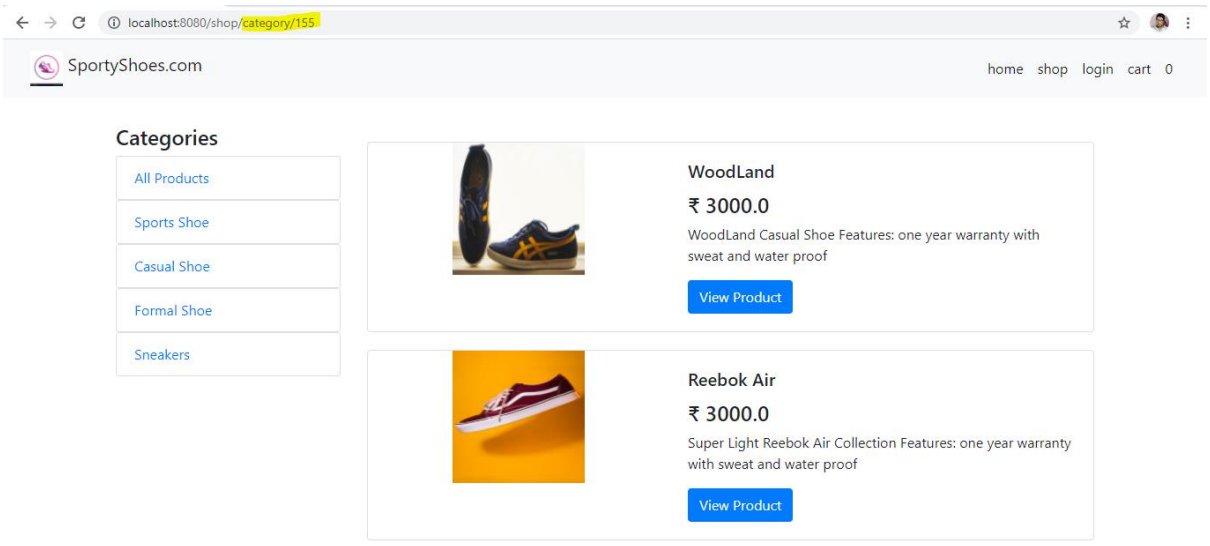


User can also select individual categories to view products only from that category (view products by category)

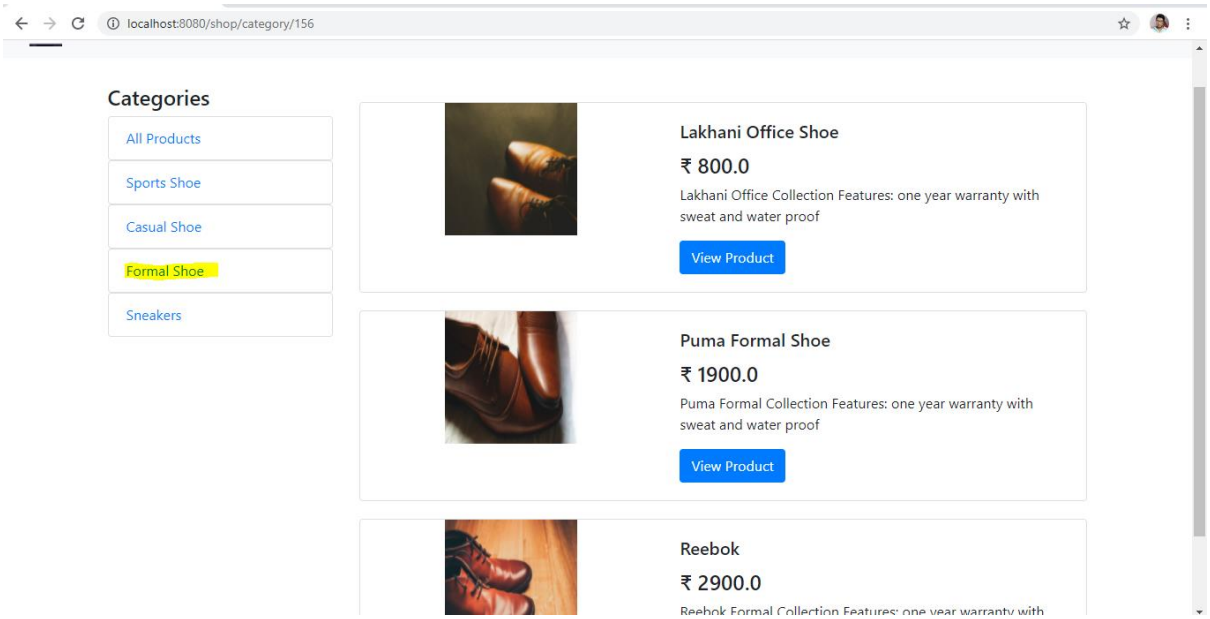
If user selects sports shoe category they can see products listed in that category



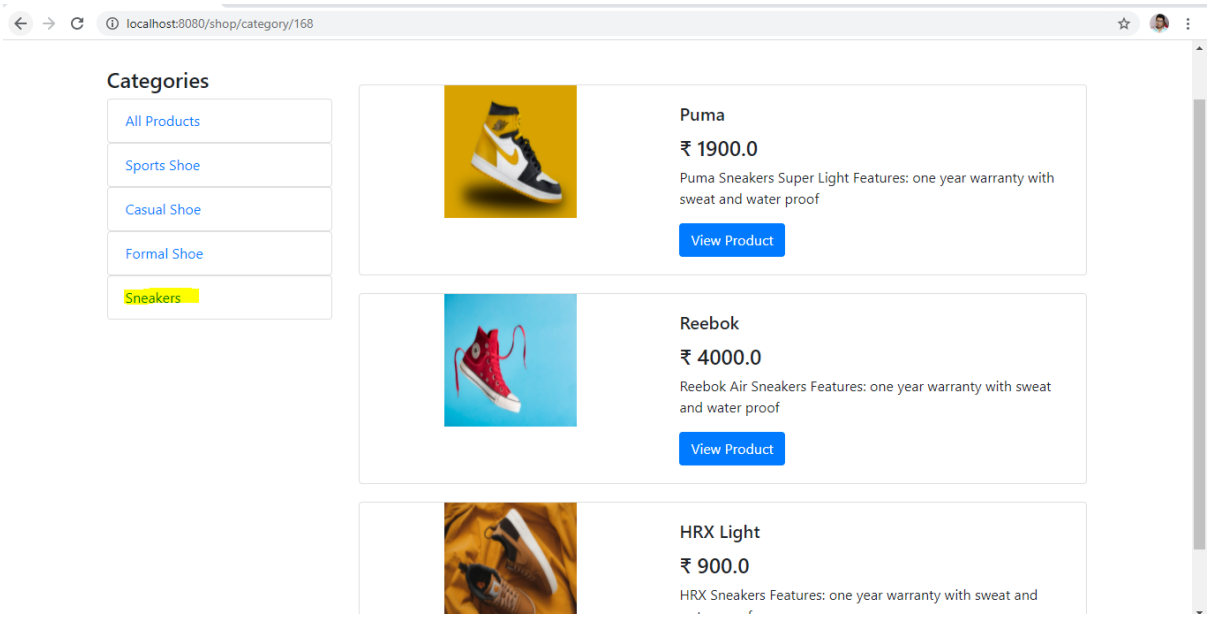
If user selects casual shoe category they can see products listed in that category



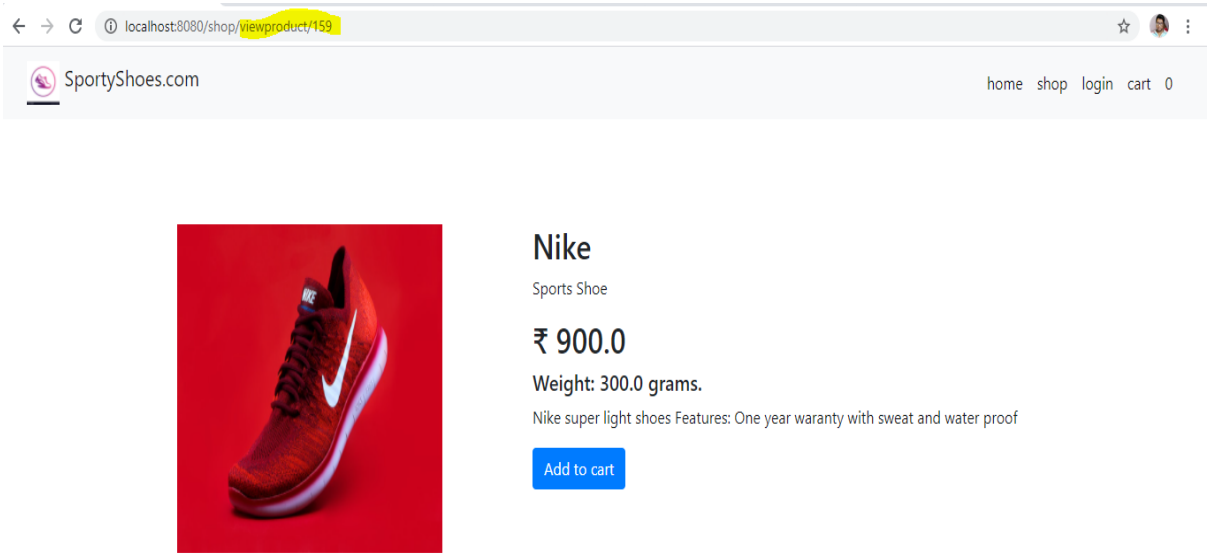
If user selects Formal shoe category they can see products listed in that category



If user selects Sneakers category they can see products listed in that category

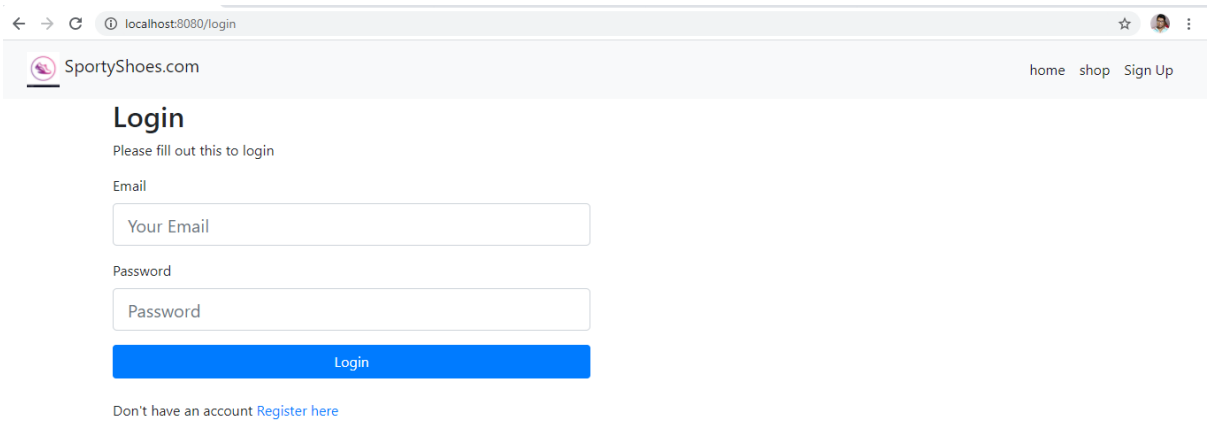


When user clicks on View Product they are redirected to specific product description page

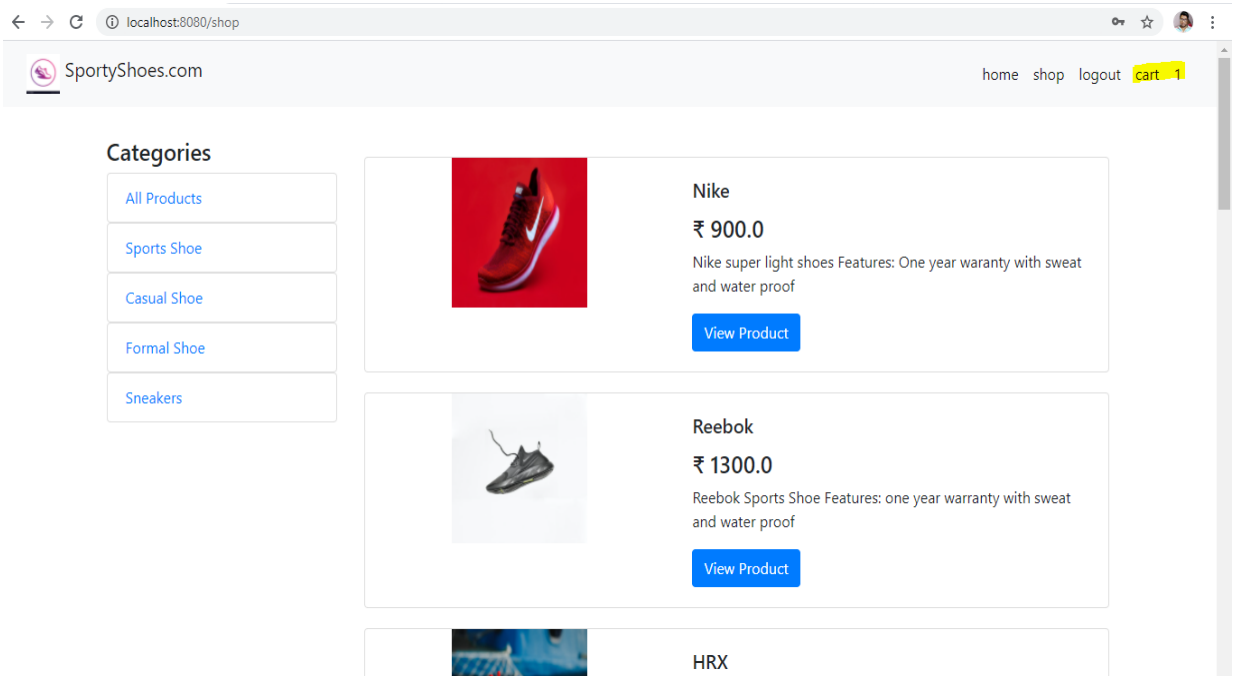


In the product description page User have option to Add the product to cart, But In order to add the product to cart user must be signed in (implemented with Spring Security).

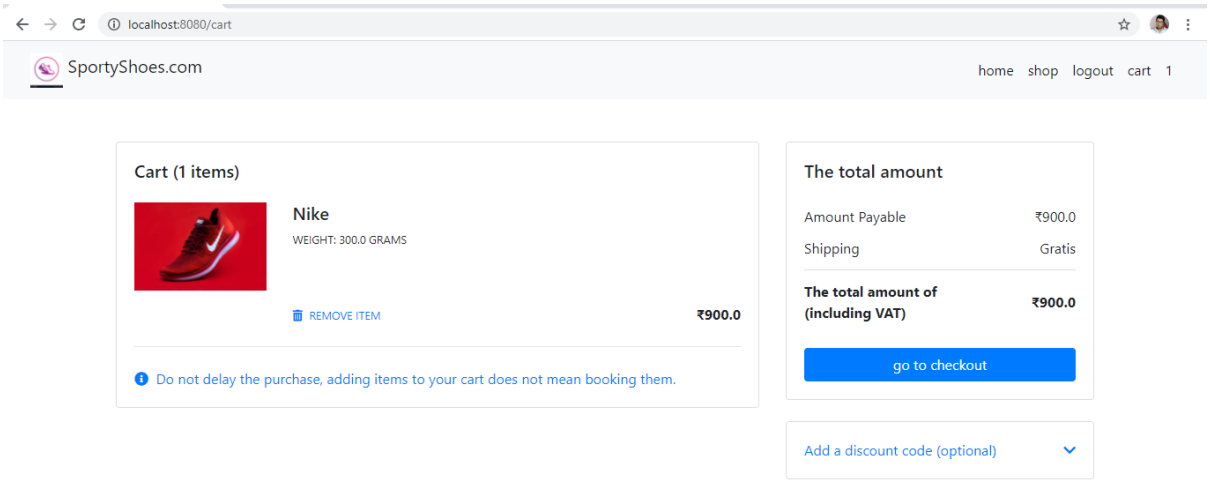
Users are redirected to Login page if they want to add products to cart



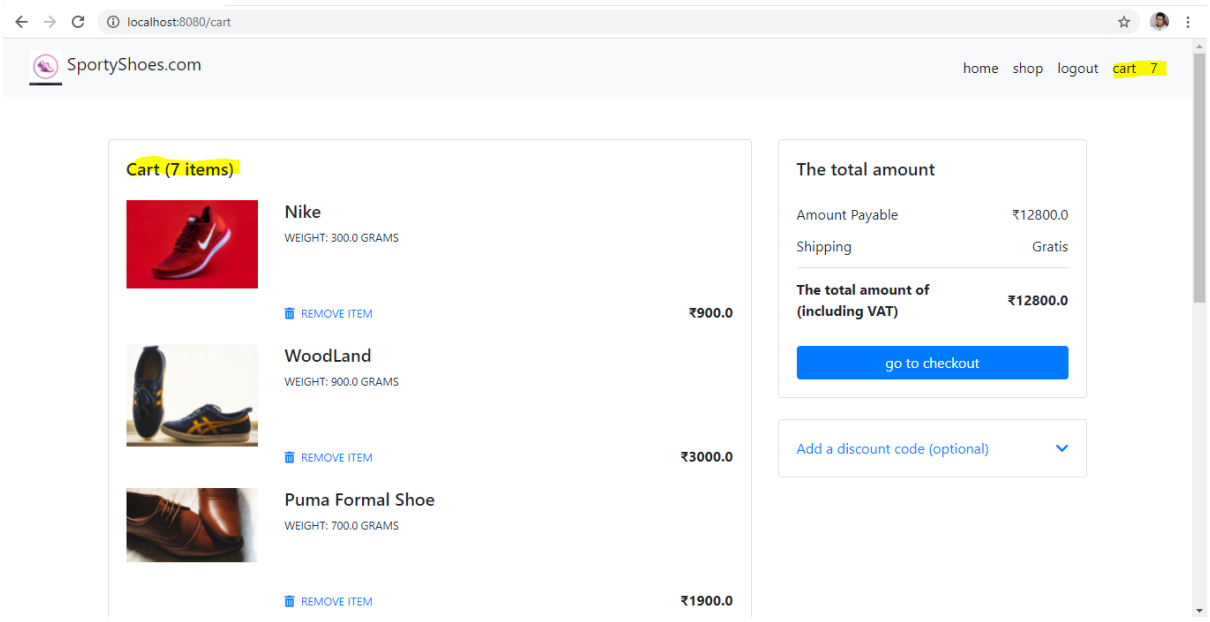
After successful login we can see the product is added to users cart and also we can see logout option now in the navbar for logged in user

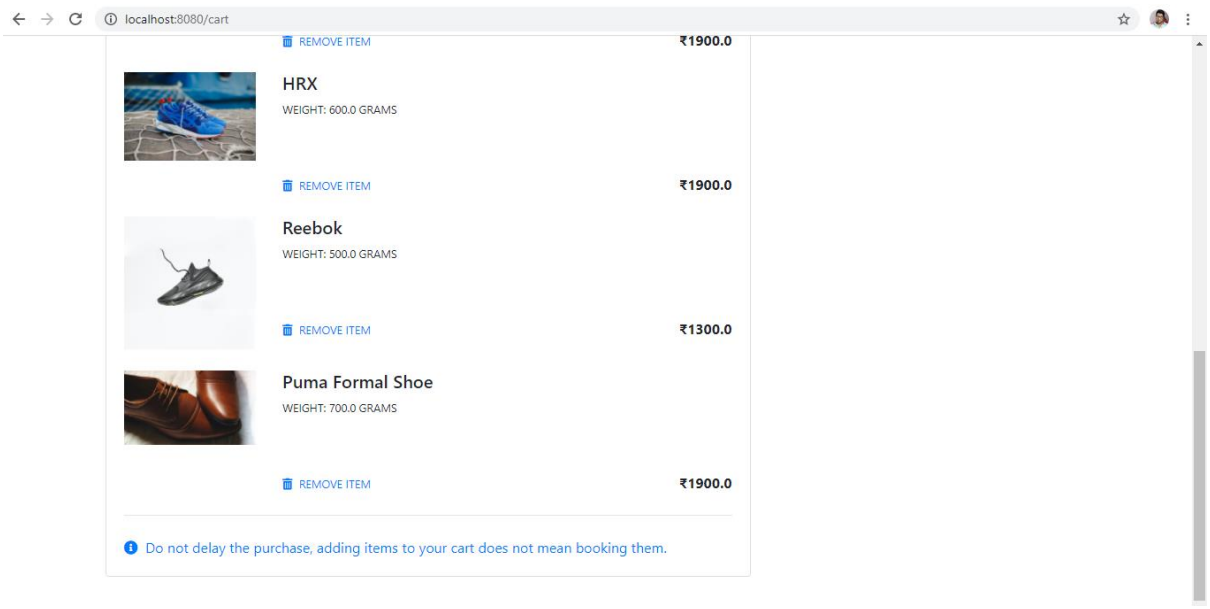


Users can view cart items by clicking on cart button in the navbar

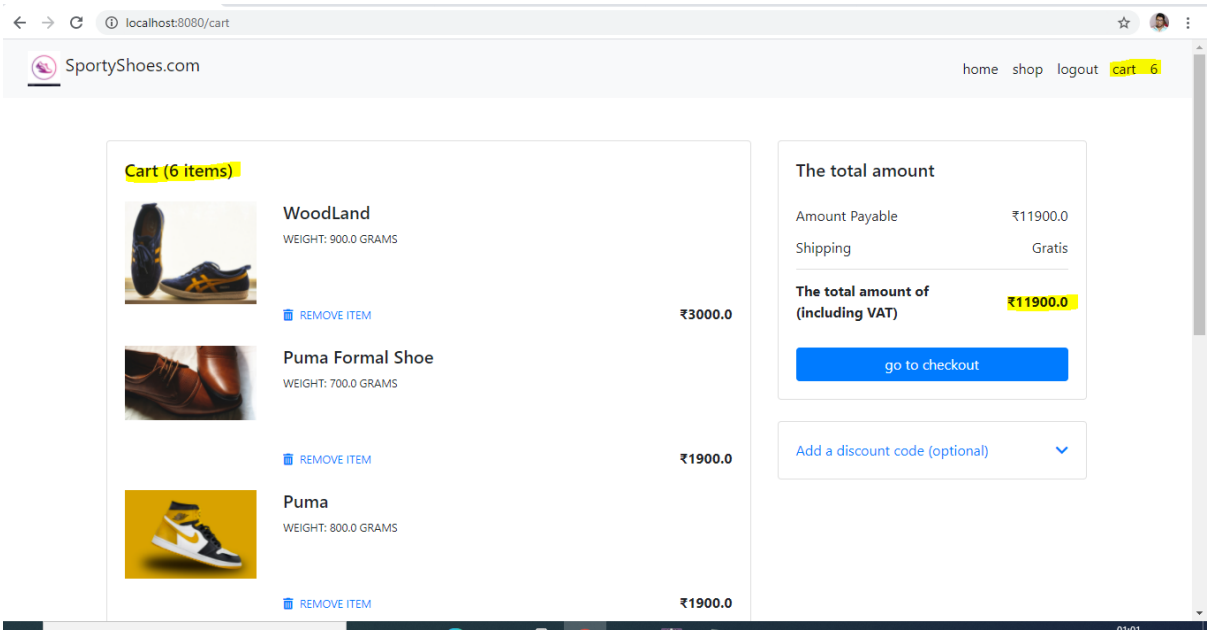


Users can also add multiple Items to their cart (Subtotal of each product, And the total price of all the products is shown)



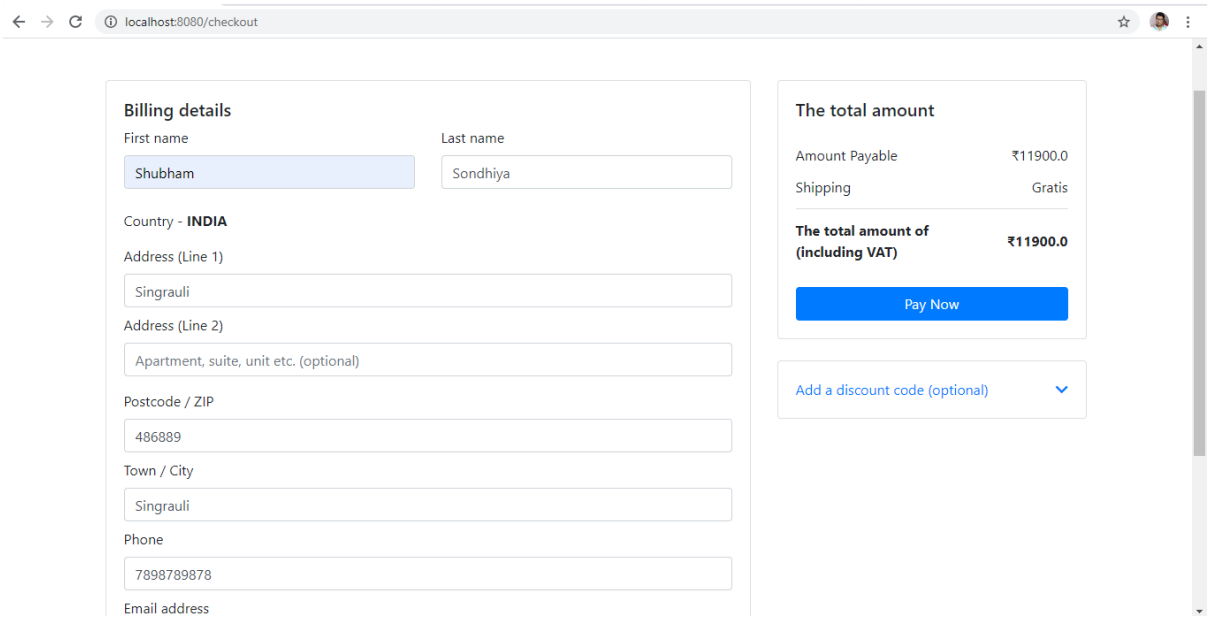


User can also remove products from cart by clicking on Remove Item link

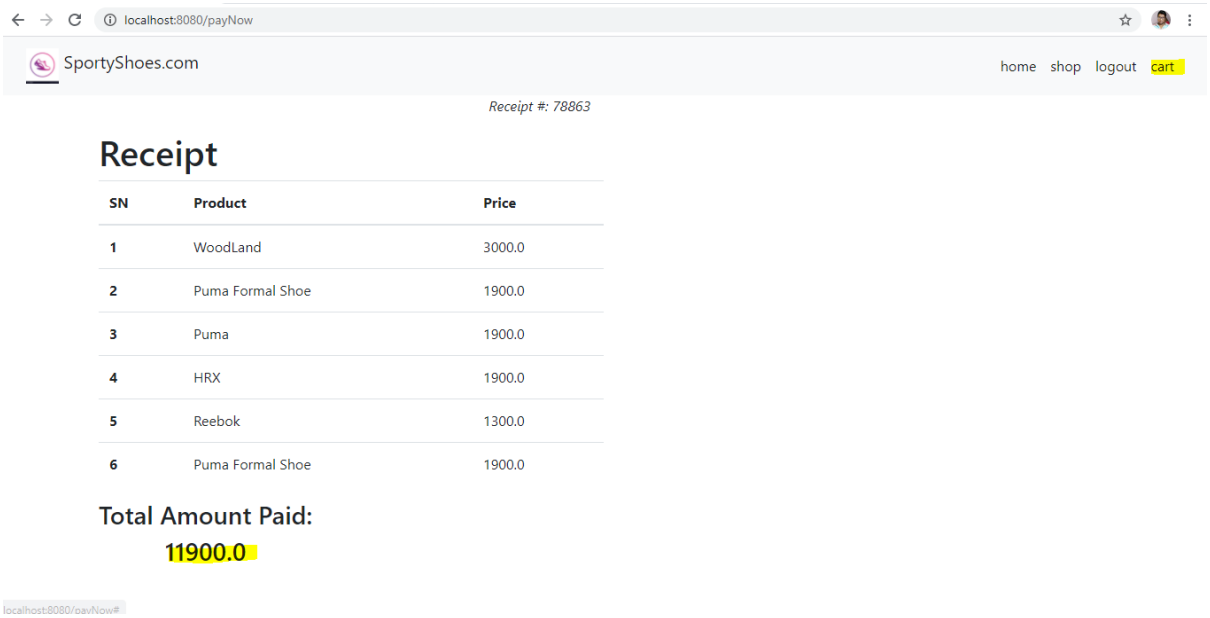


(First item has been removed and now the total cart items are 6)

When user clicks Got to checkout button they will be redirected to Checkout Page where they need to fill Billing details and pay the total amount

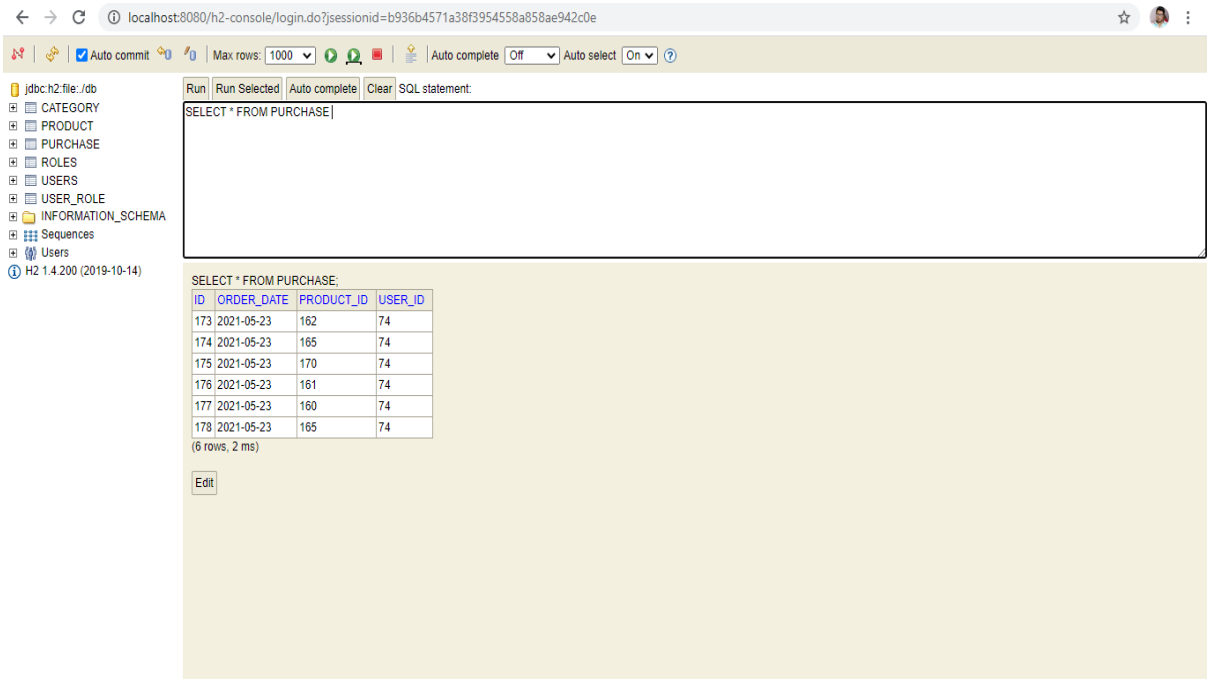


When user clicks on Pay now then they are redirected to Order Placed page where they can see the details of order placed and the products purchased by the user are persisted to the database



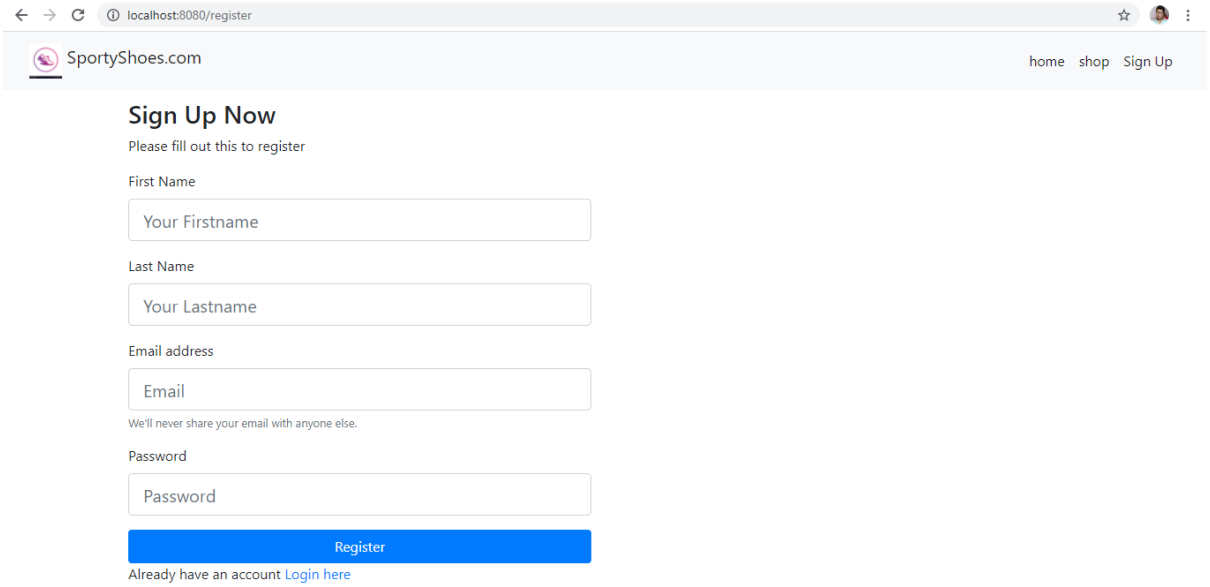
(We can see now the cart is showing 0 items)

All the bought products by user are persisted in the database

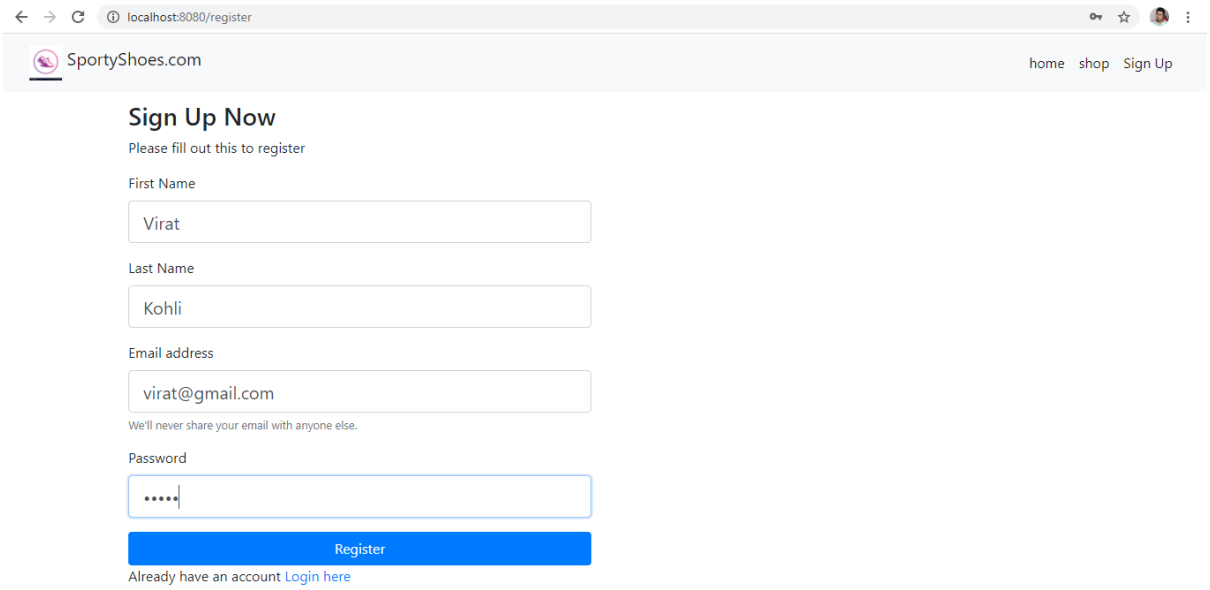


2. User Sign In and Register

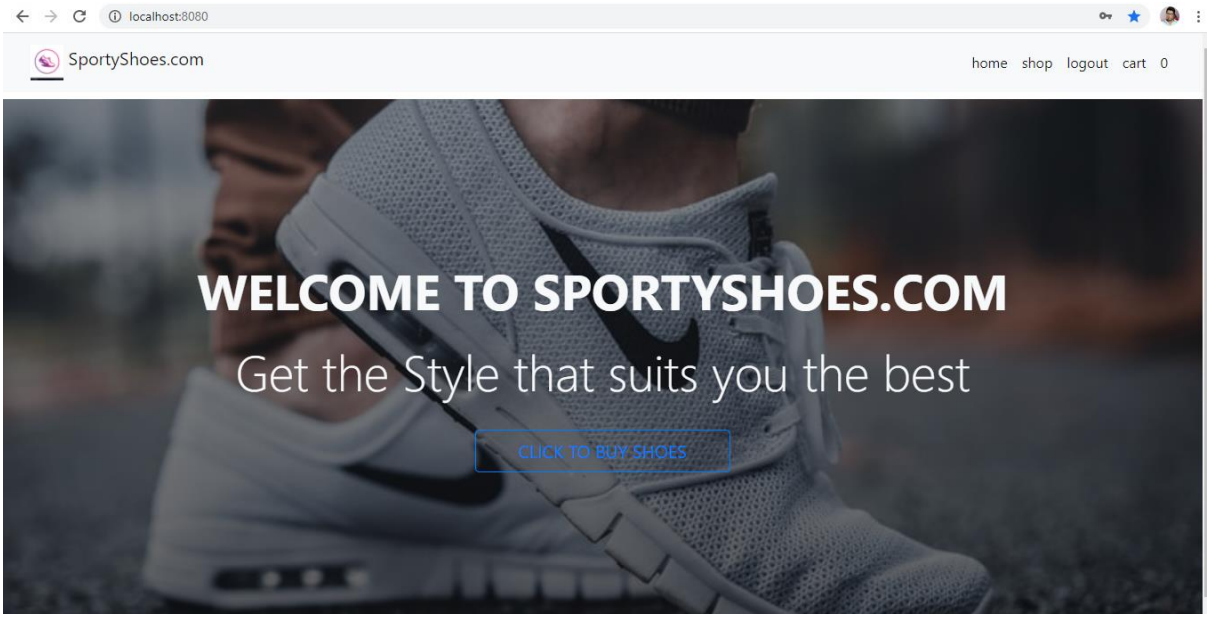
New users can register in the system by signing up



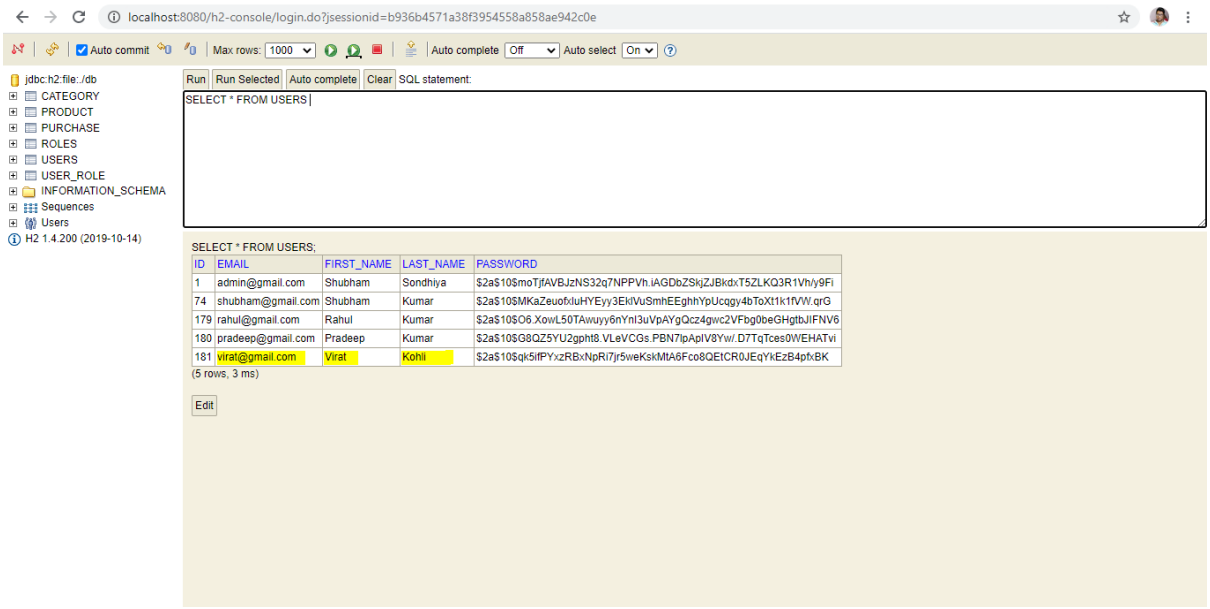
Filling up the details in the register form



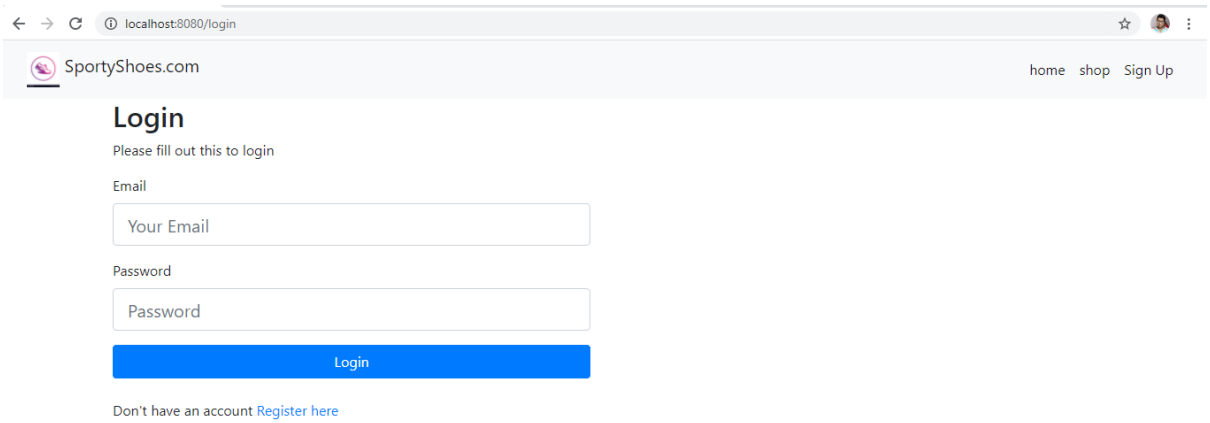
Once user gets registered they are automatically logged in and redirected to Home Page where they can buy products

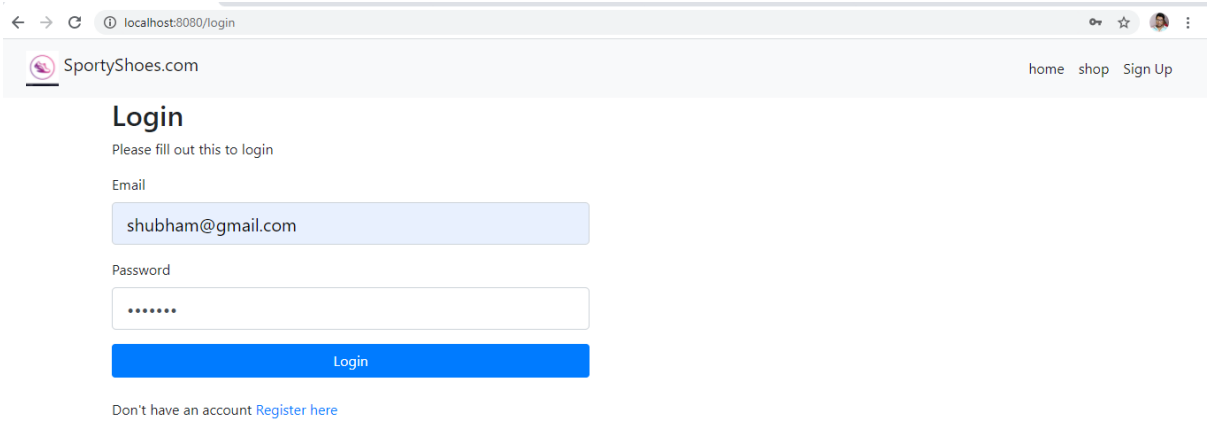


We can see user registered user is added in the database

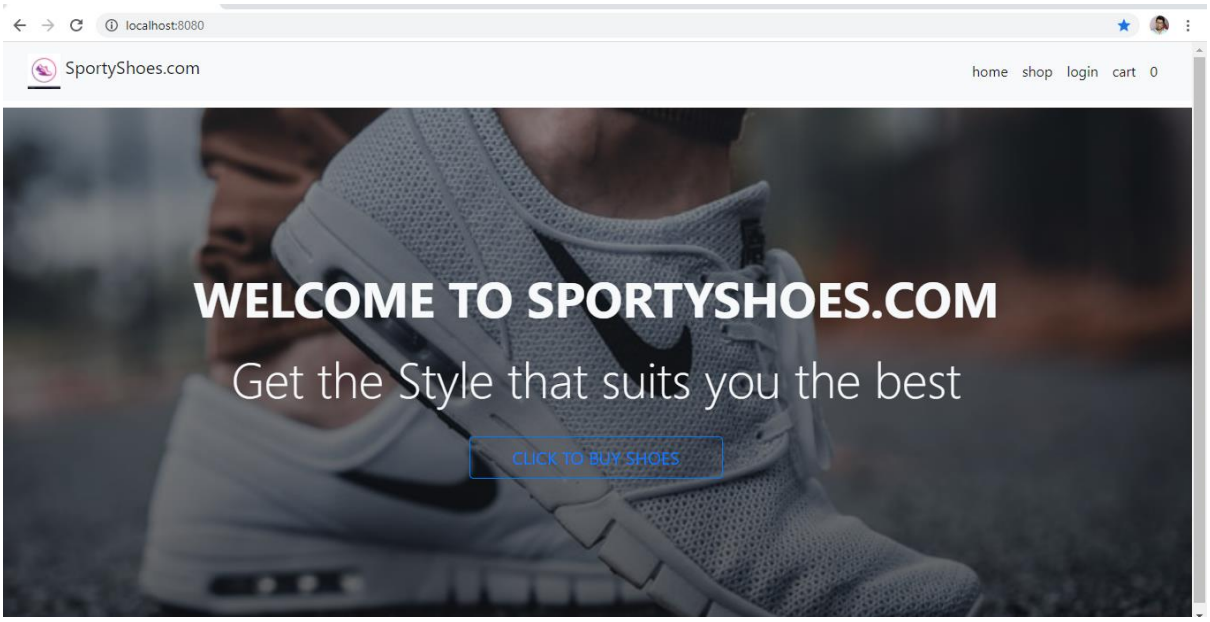


Already registered users can login directly in the system

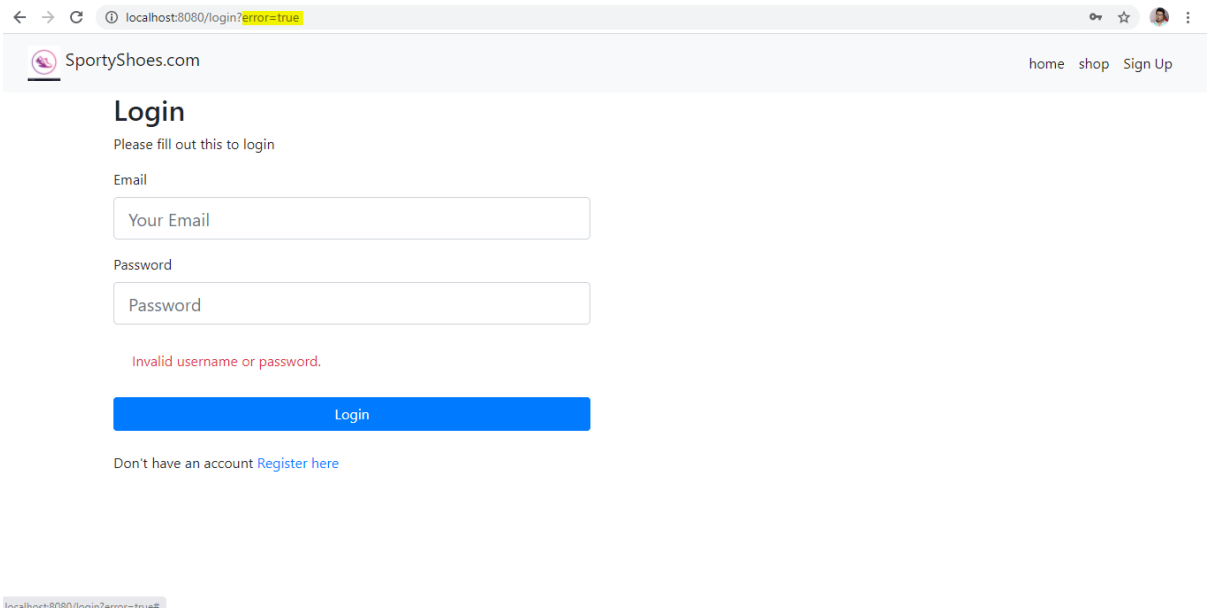




After successful login they are redirected to Home page where they can buy products

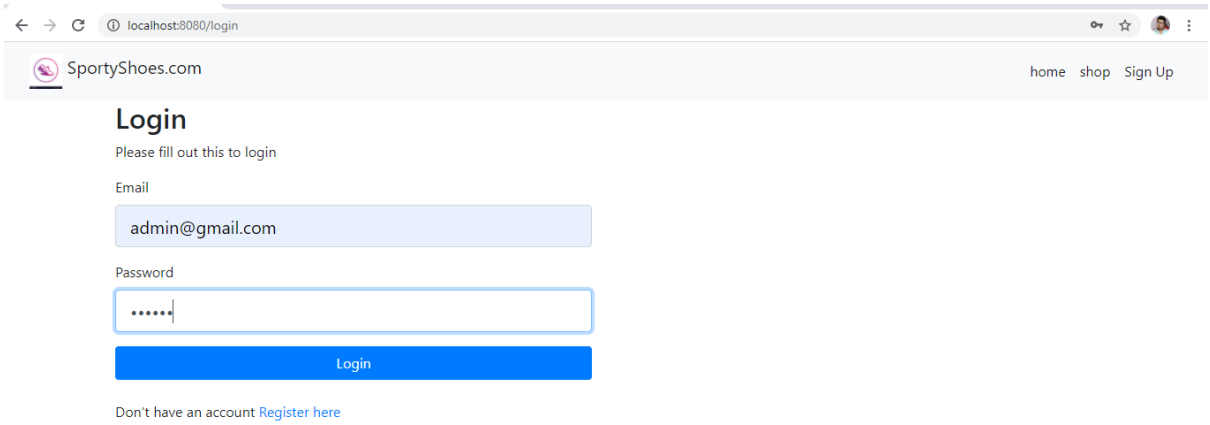


When user provided invalid details they are redirected to the same login page with error message as Invalid username or password

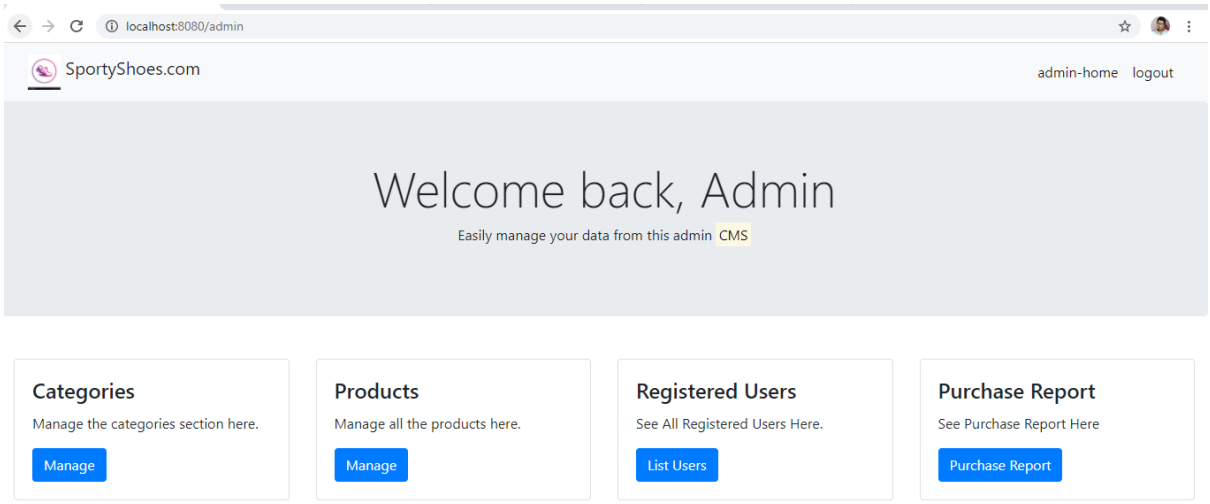


3. Admin Related Scenarios

Admin can login in the system by providing their credentials (Admin is added from the backend)

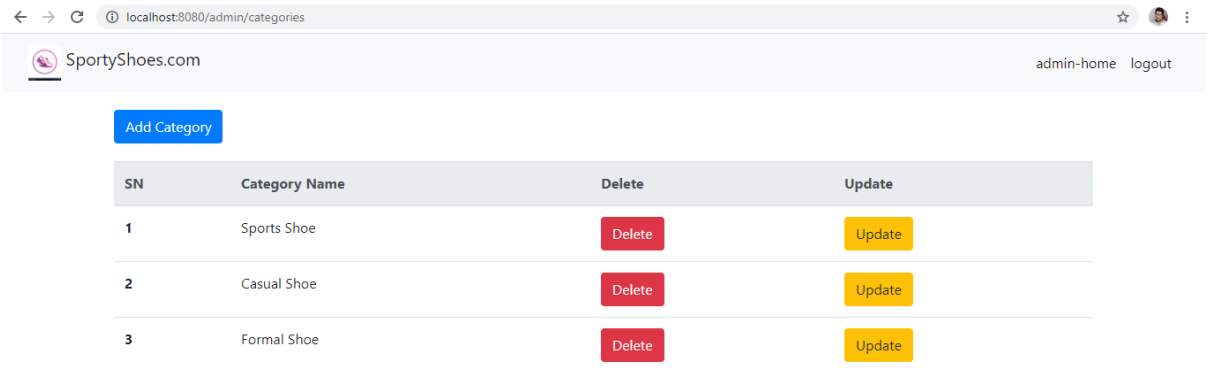


Once admin is logged in they can see All the operations they can perform like , Manage Categories, Manage Products, See all the registered Users list in the system, See the purchase report

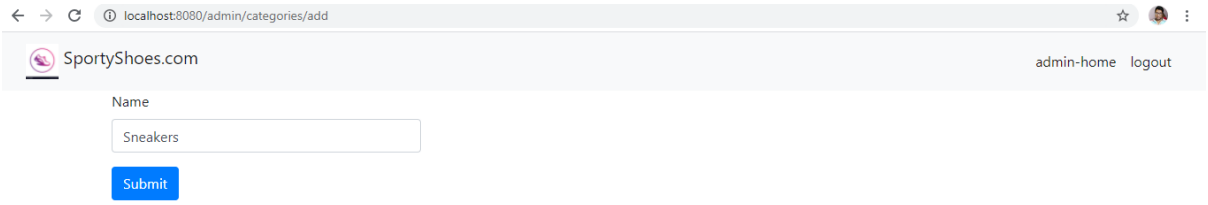


3.1. Admin can Manage Categories (All CRUD Operations)

Admin can see list of all categories added in the system

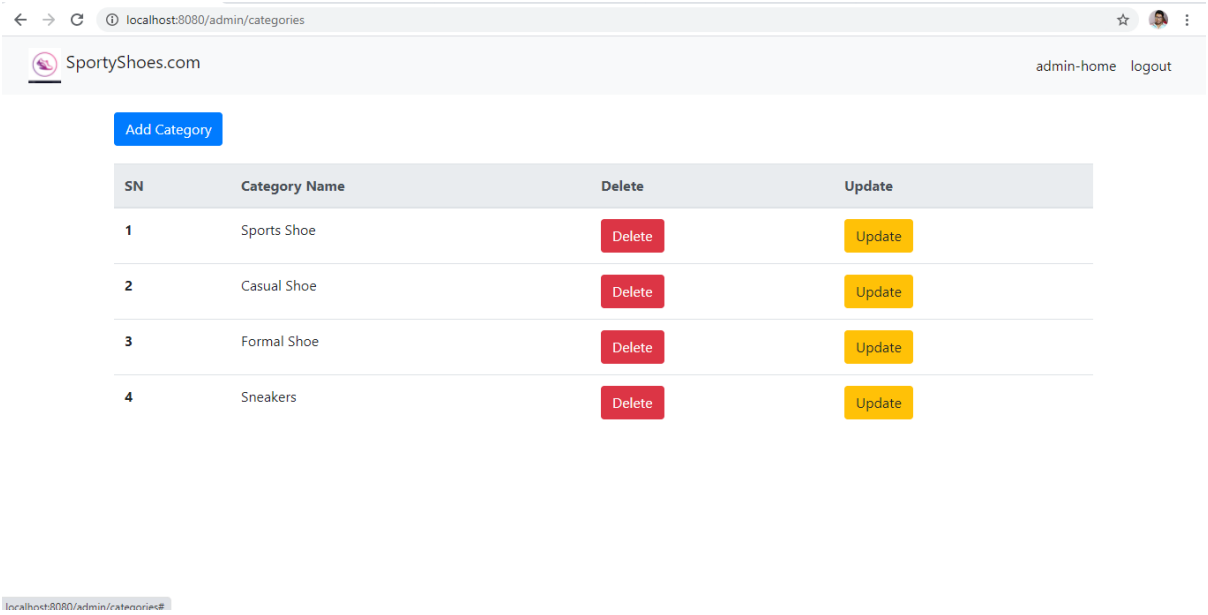


Admin can add a new category by clicking on Add Category Button

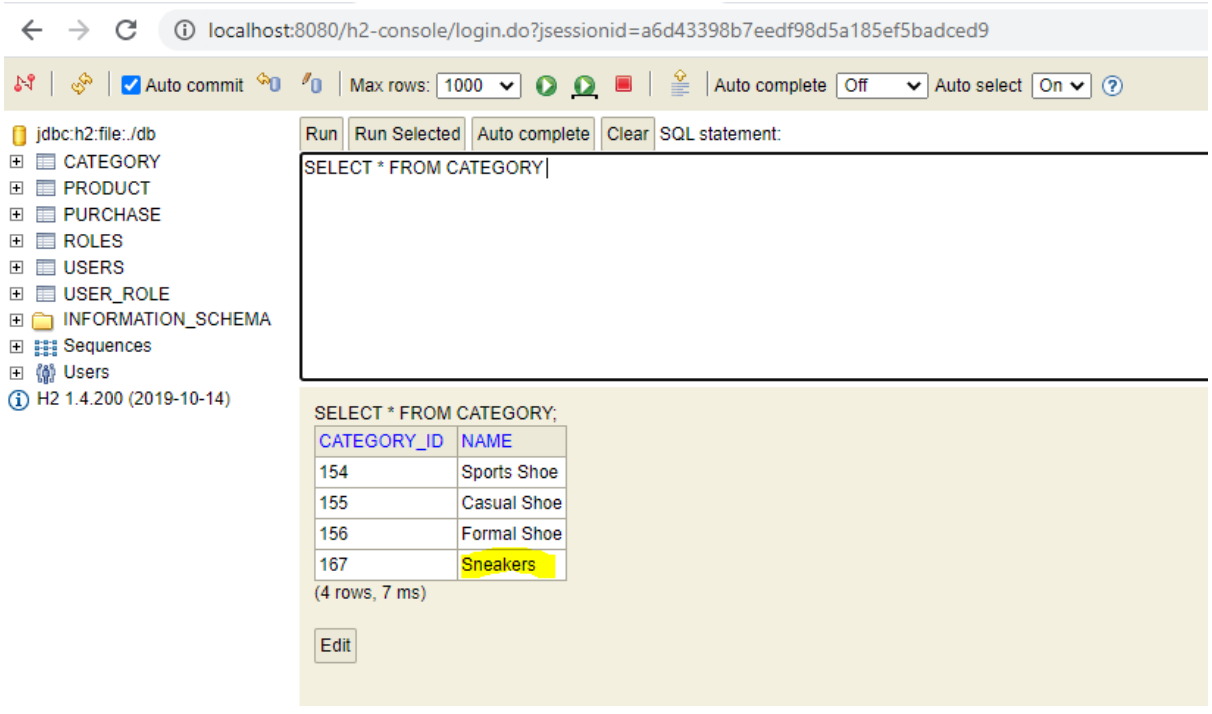


(Here admin provide category name and adds a category)

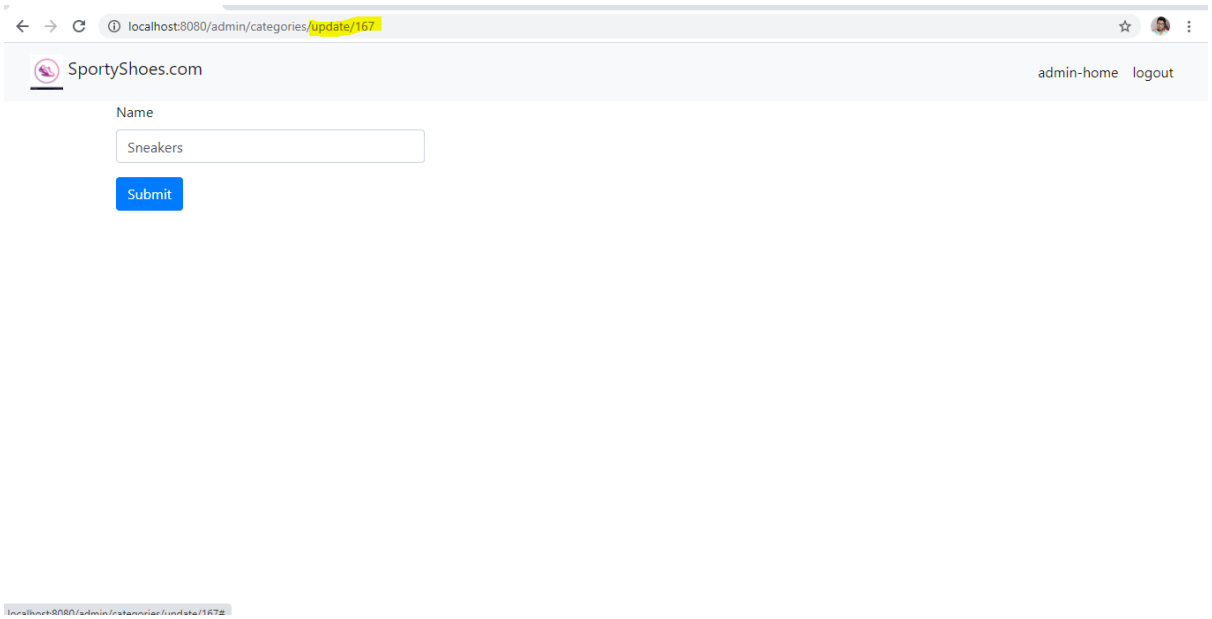
We can see new category Sneakers is added



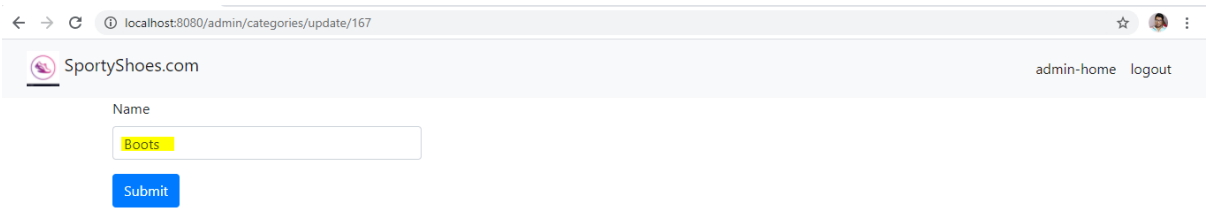
New added category is persisted in the database

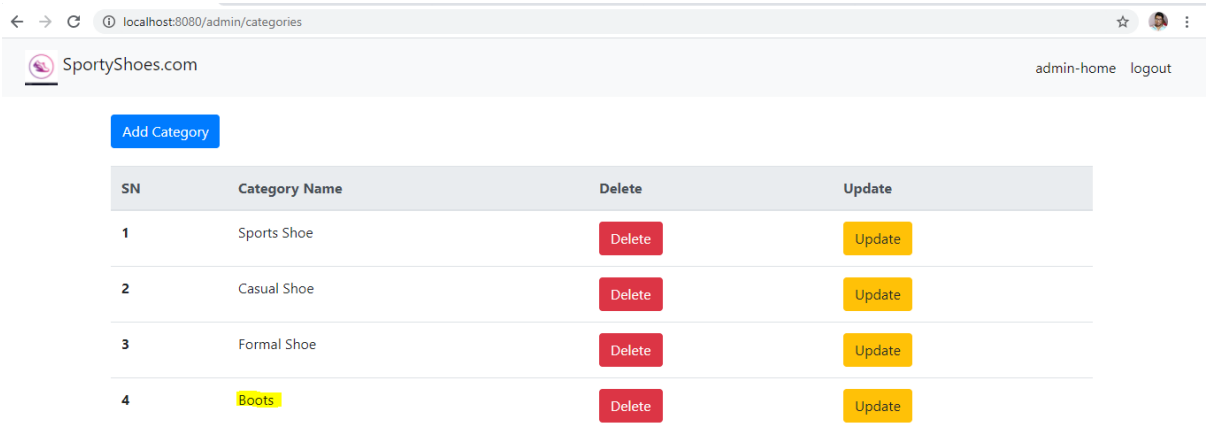


Admin can update a category by clicking on update button next to each category (Form comes prefilled with current values)

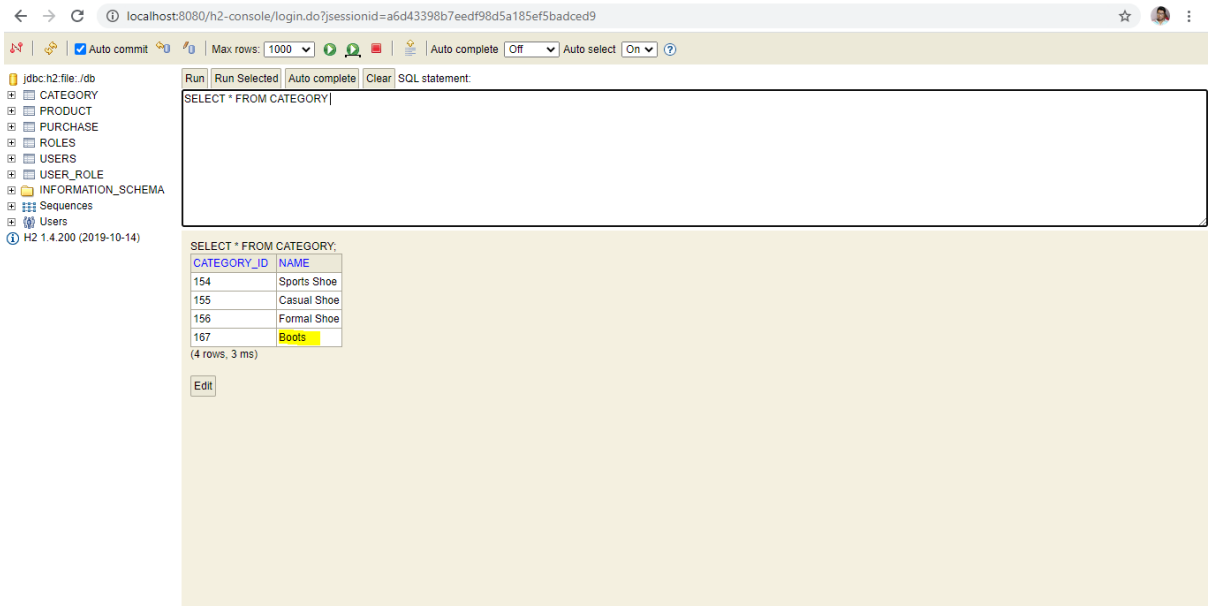


Sneakers category Updated to Boots

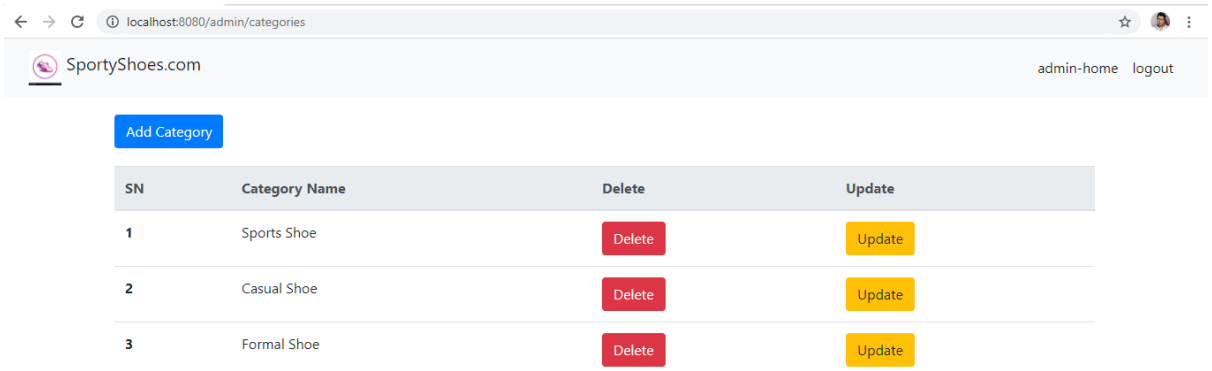




Category updated in the database also

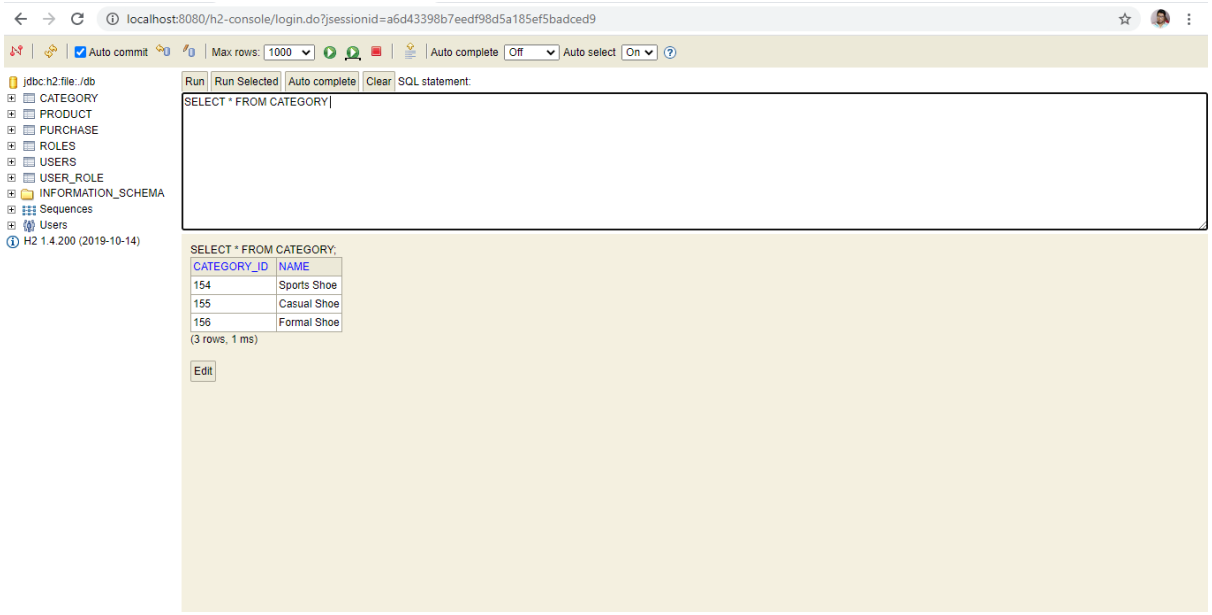


Admin can also delete a category by clicking on delete button next to each category



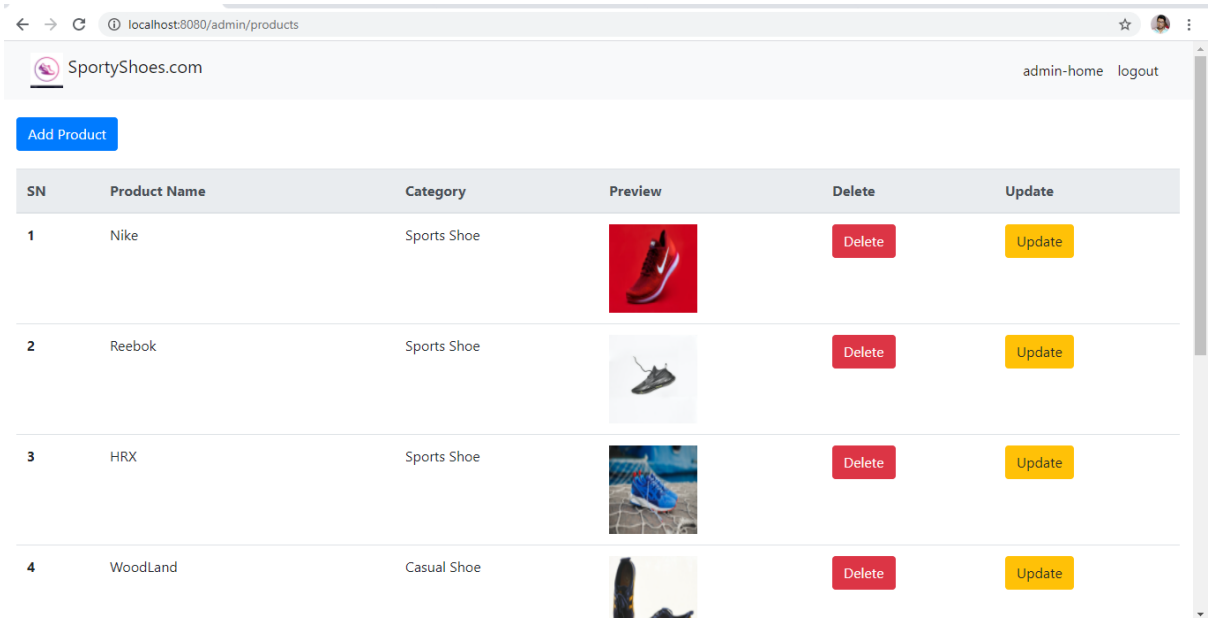
(Boots category Deleted)

Boots category deleted from database also

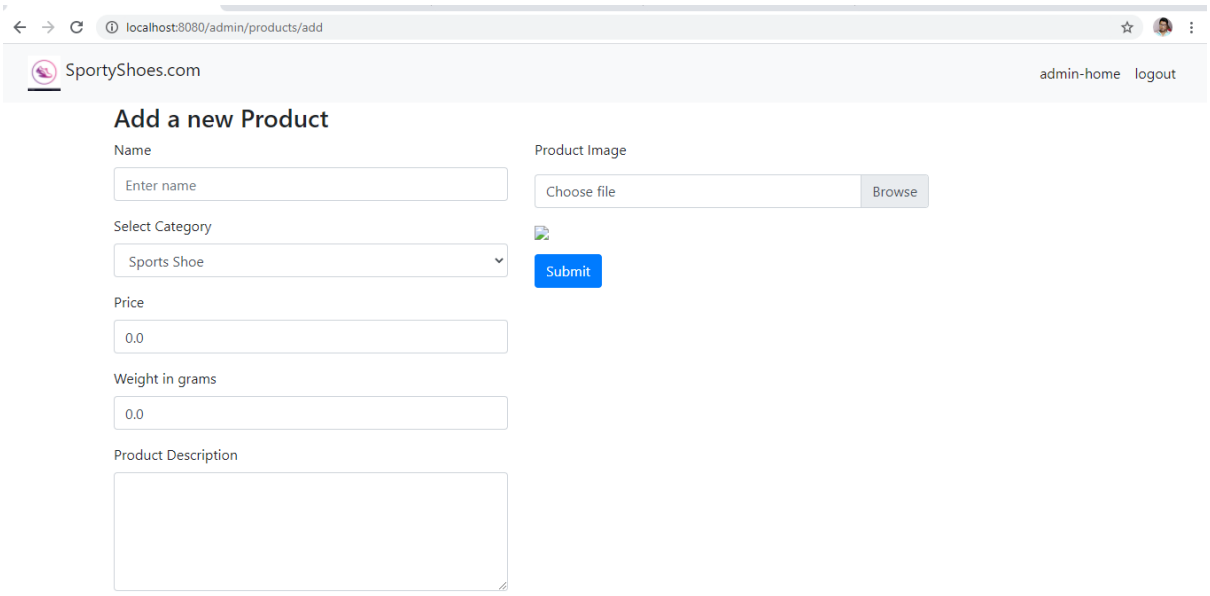


3.2. Admin can Manage Products (All CRUD Operations)

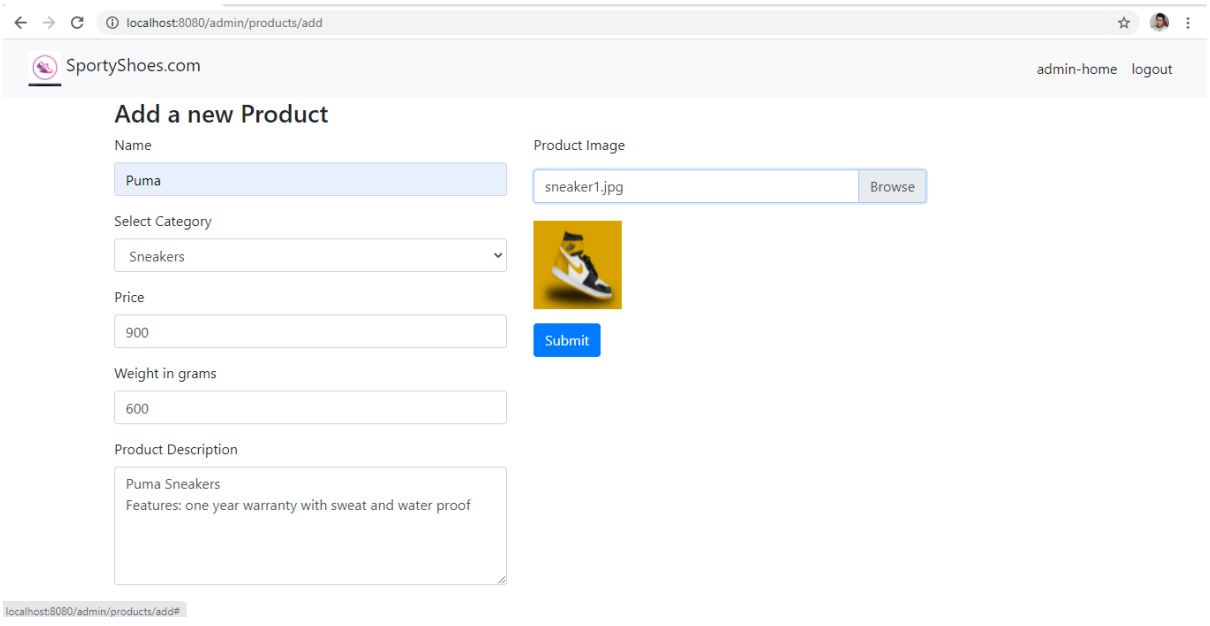
Admin can see list of all products added in the system



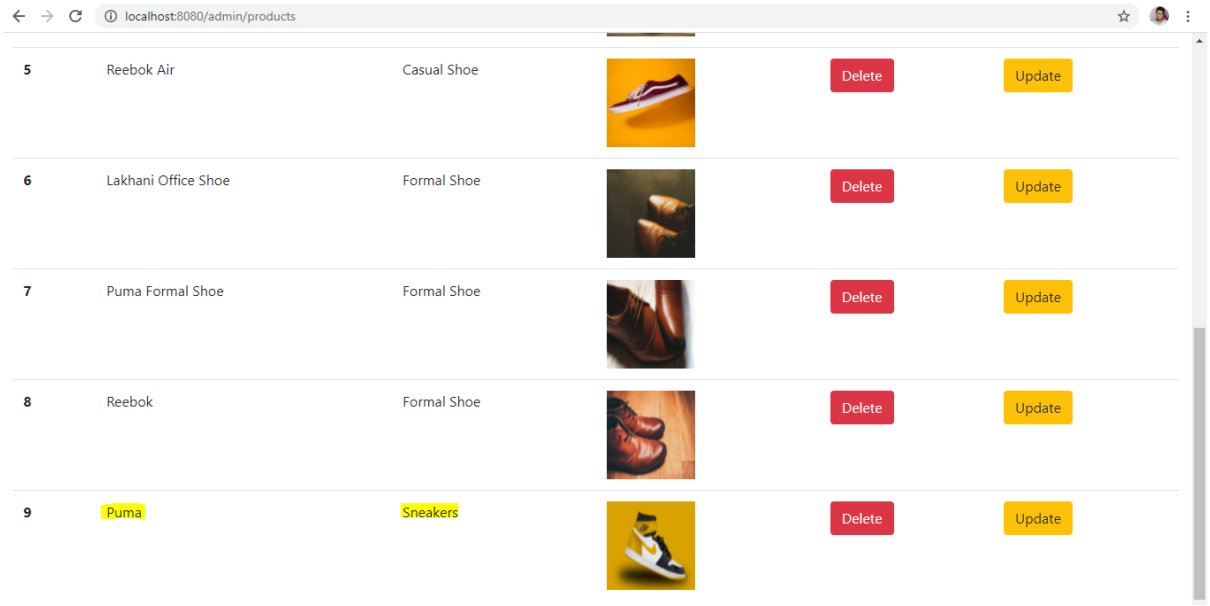
Admin can add a new product by clicking on Add Product Button



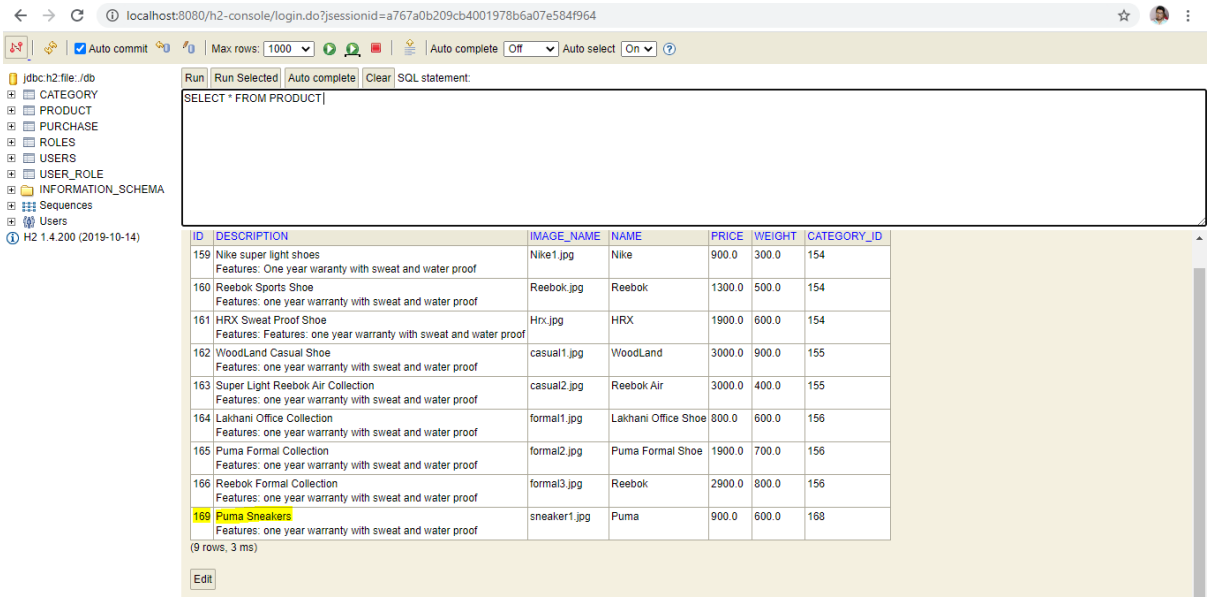
Admin has to fill all the details of the product and also provide an image from the system for the product (Adding PUMA Shoes of category Sneakers)



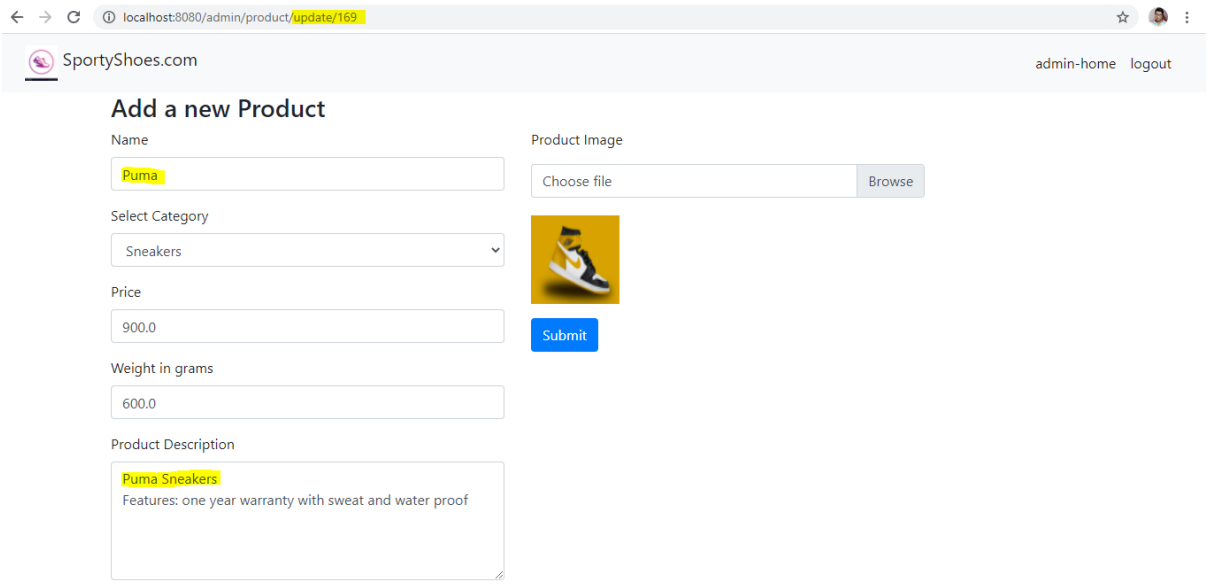
New product added in the product list after submission of details



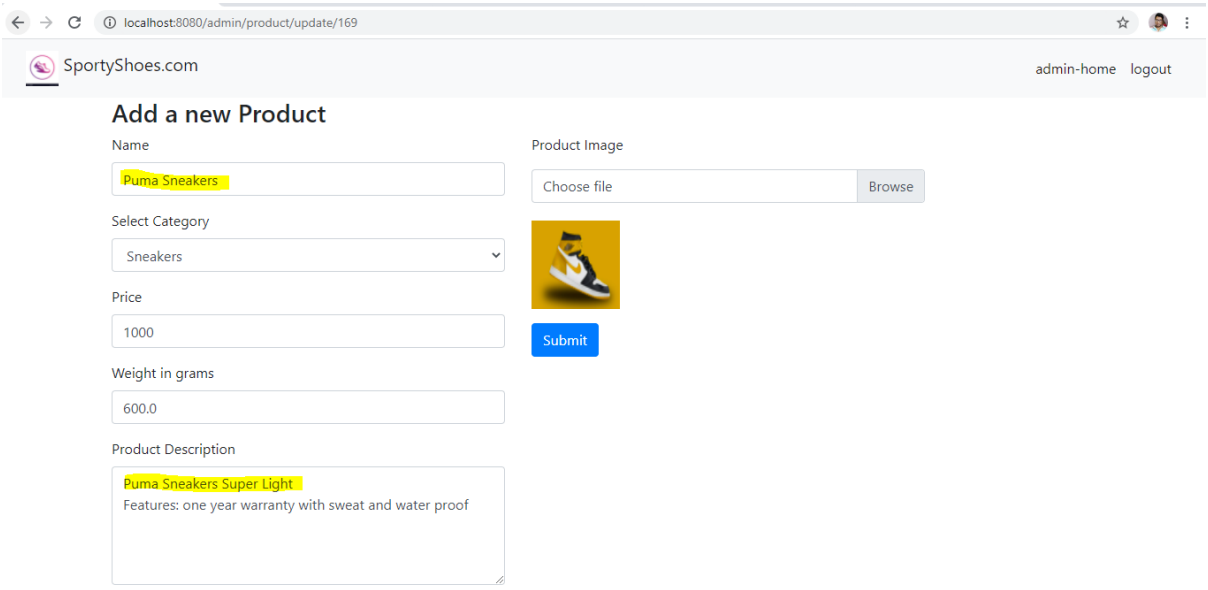
New Added product is also persisted in the database



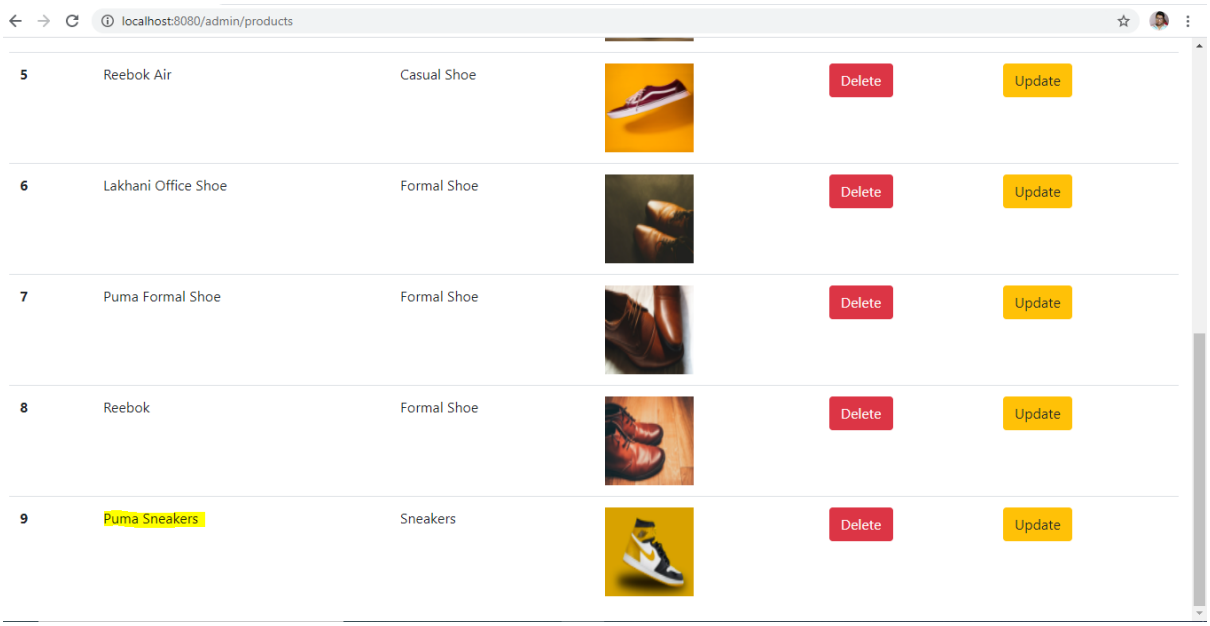
Admin can update a product by clicking on update button next to each product (Form comes prefilled with current values)



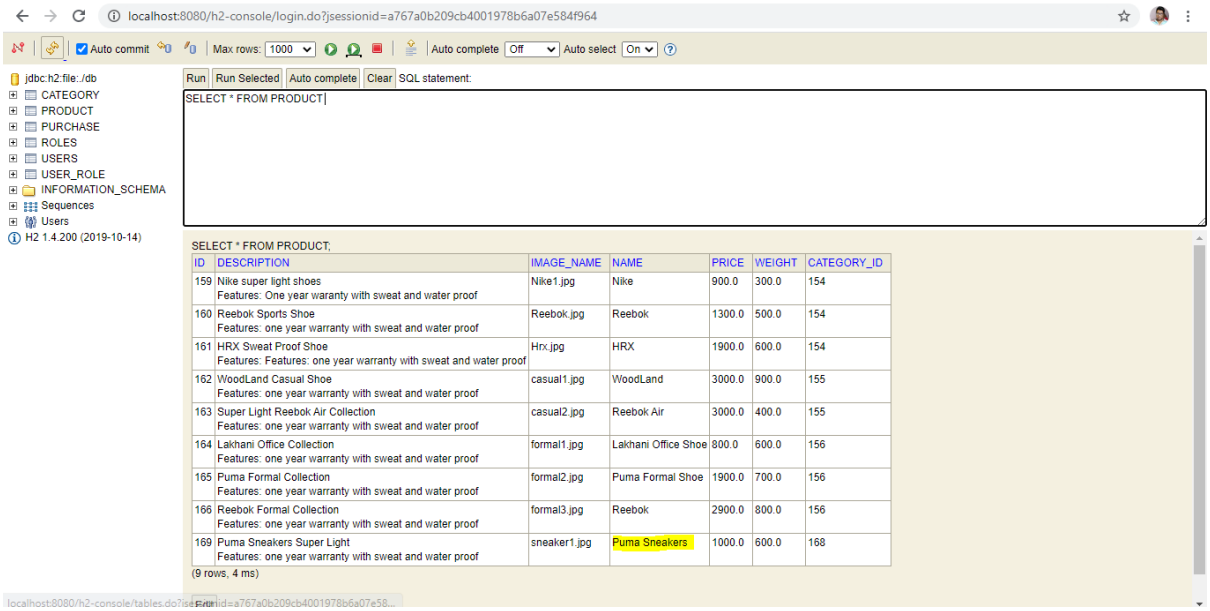
Updating to New Values



Product details gets updated in the product list

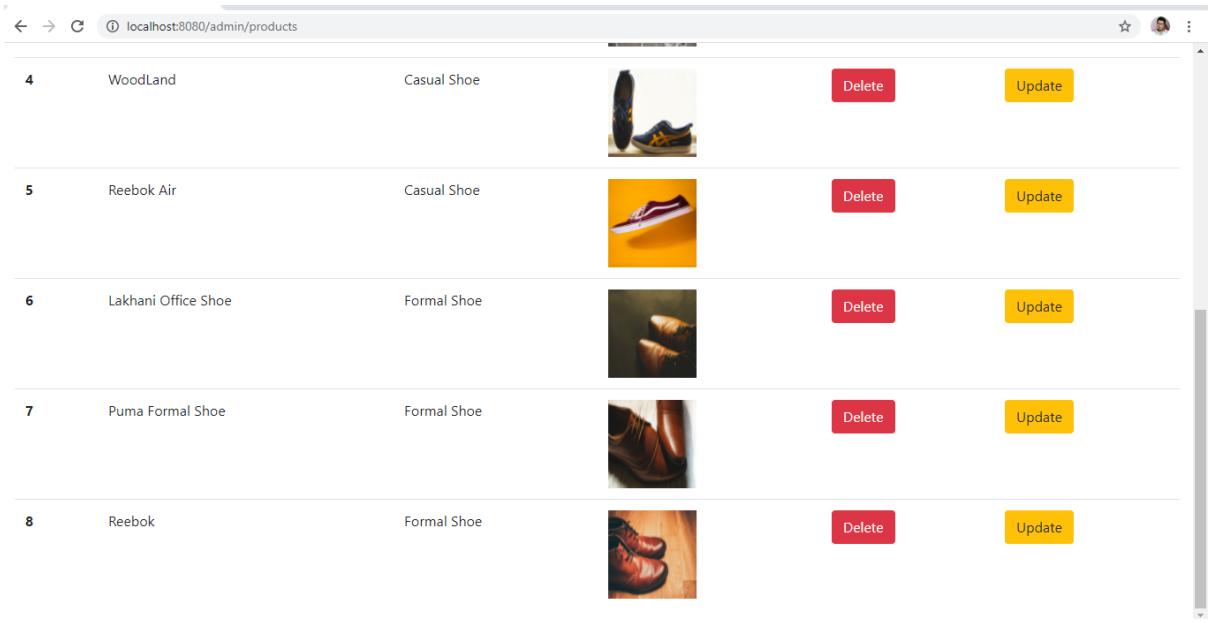


Product also gets updated in the database

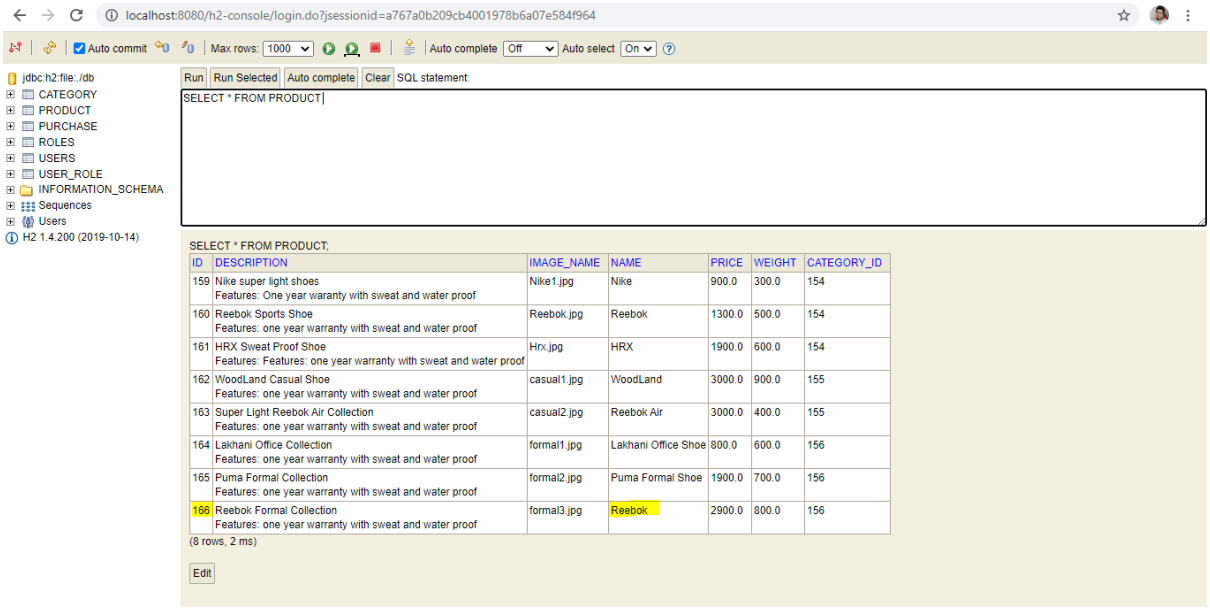


Admin can also delete a product by clicking on delete button next to each product

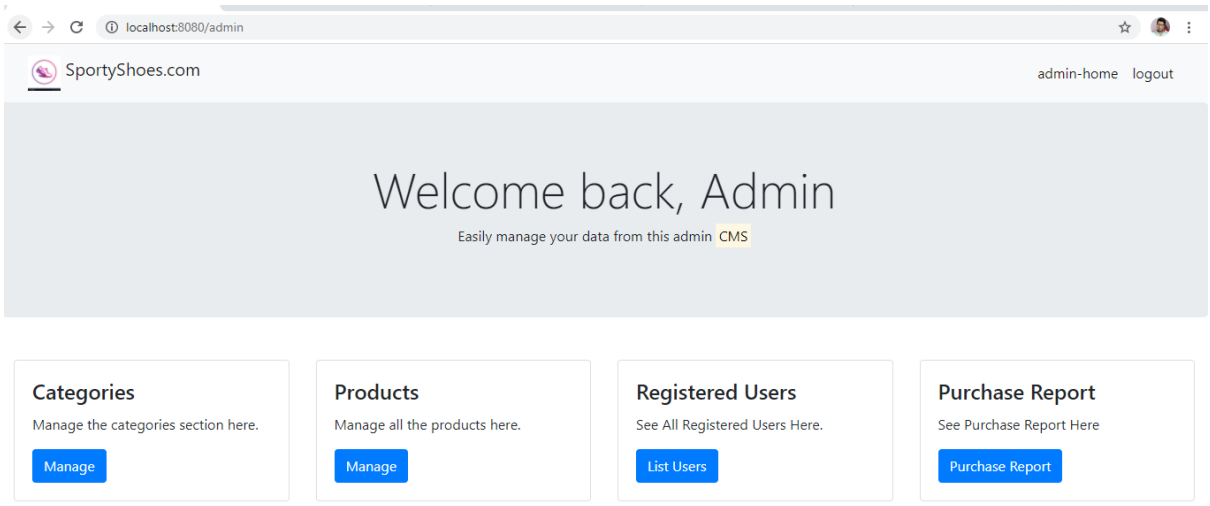
(Deleted Puma Sneakers product)



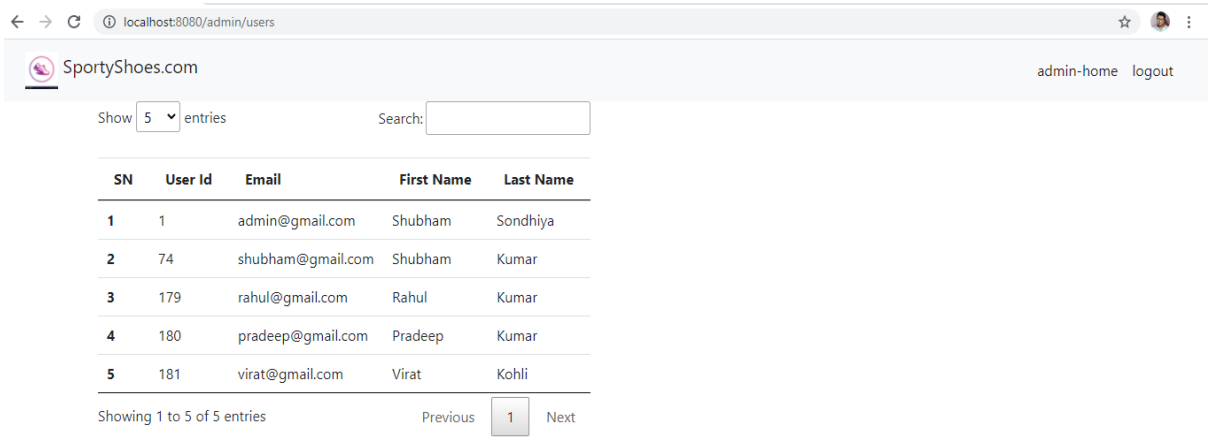
Product is also deleted from the database



3.3. Admin can see a list of all the registered users in the system (By going in Registered Users Tab)

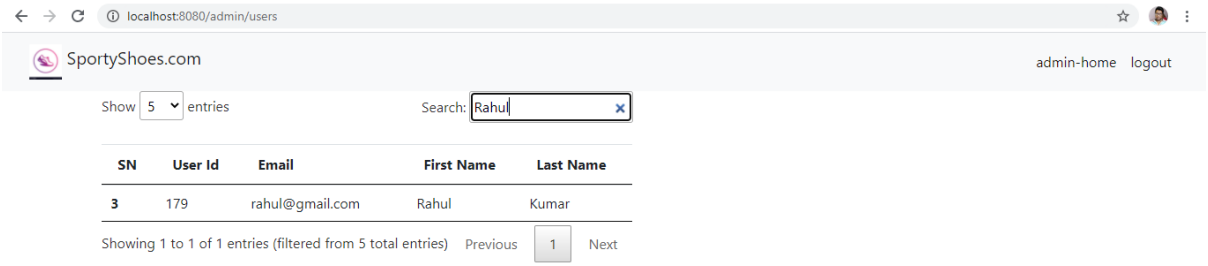


Admin can see list all registered users in the system

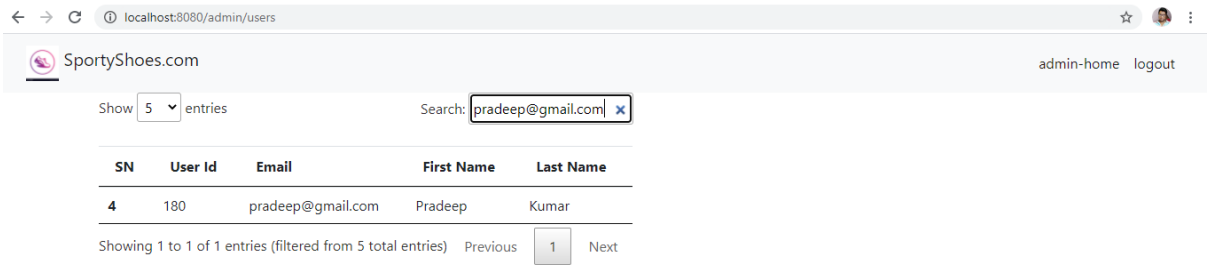


Admin can also filter the results from the search bar

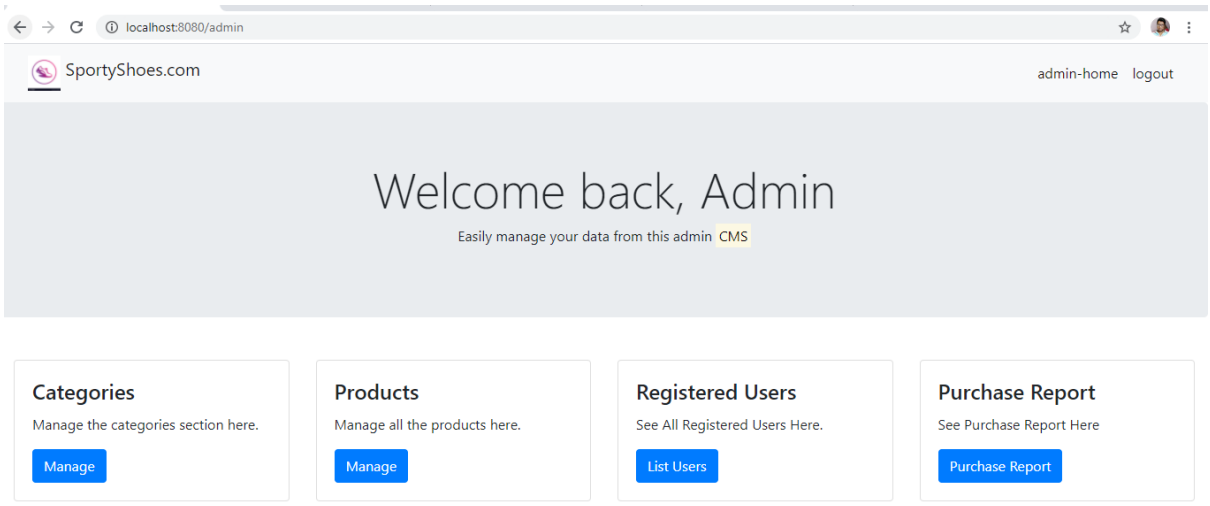
Filtering by First Name



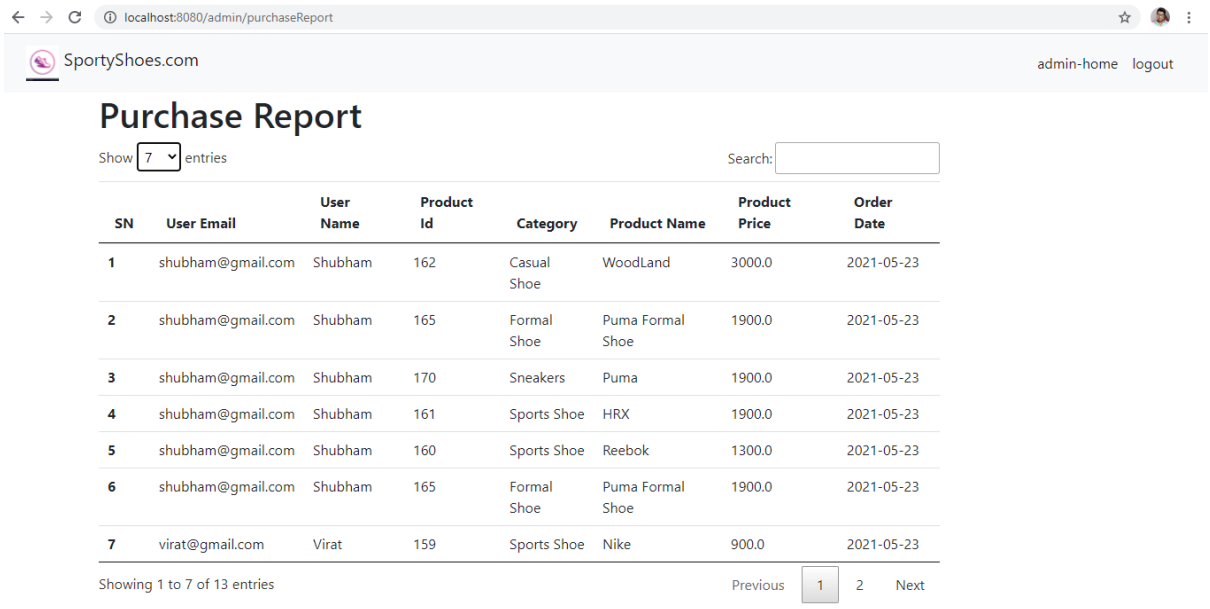
Results filtered by email

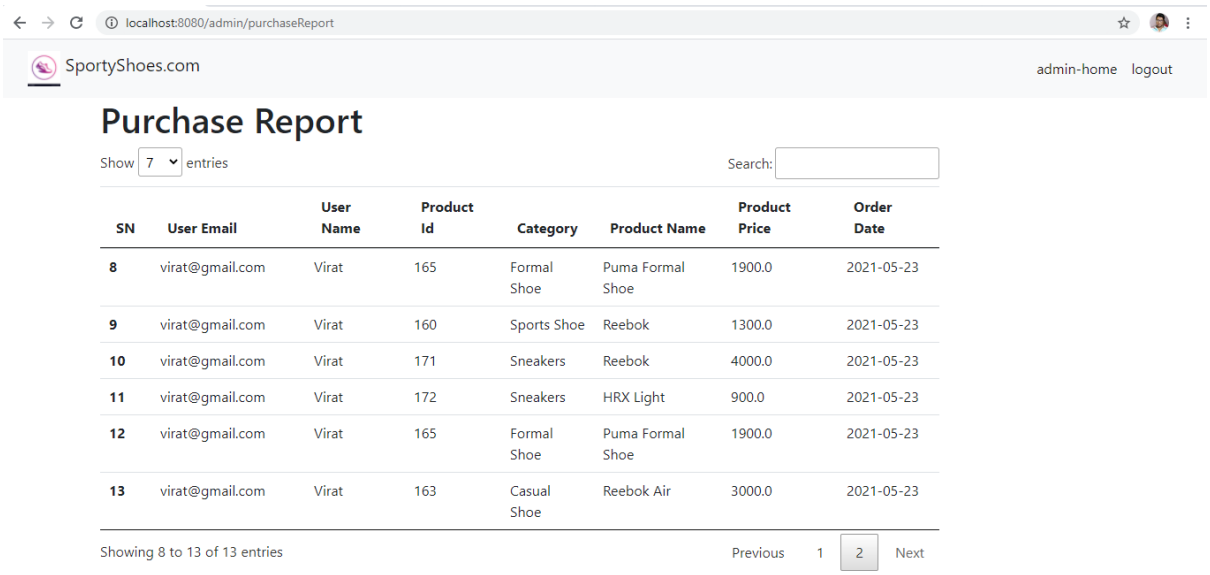


3.4. Admin can see Purchase report in the system (By going in Purchase report Tab)

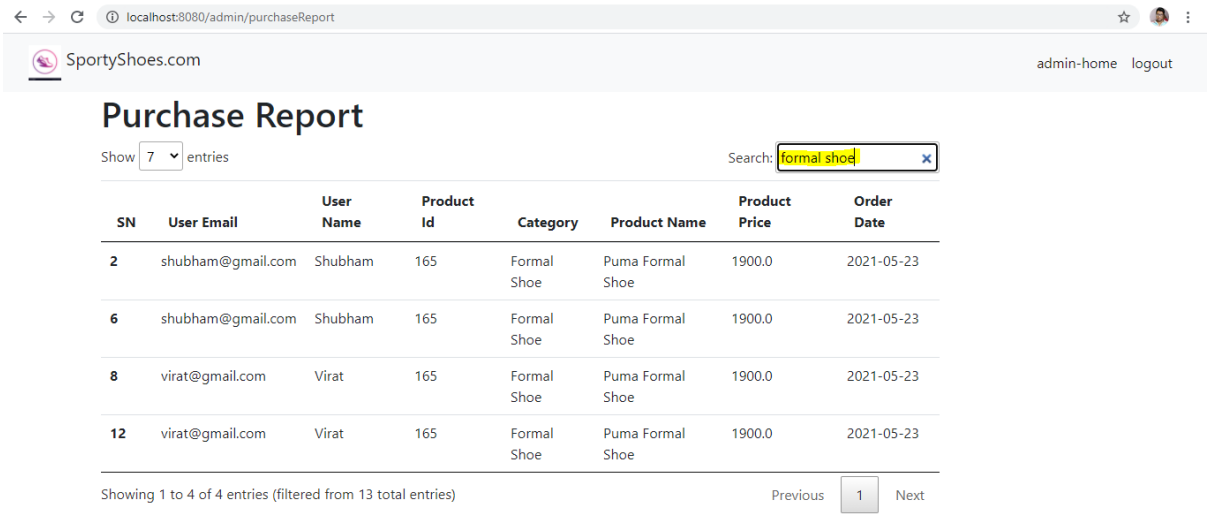


If nothing is present in the search bar then All purchases are returned

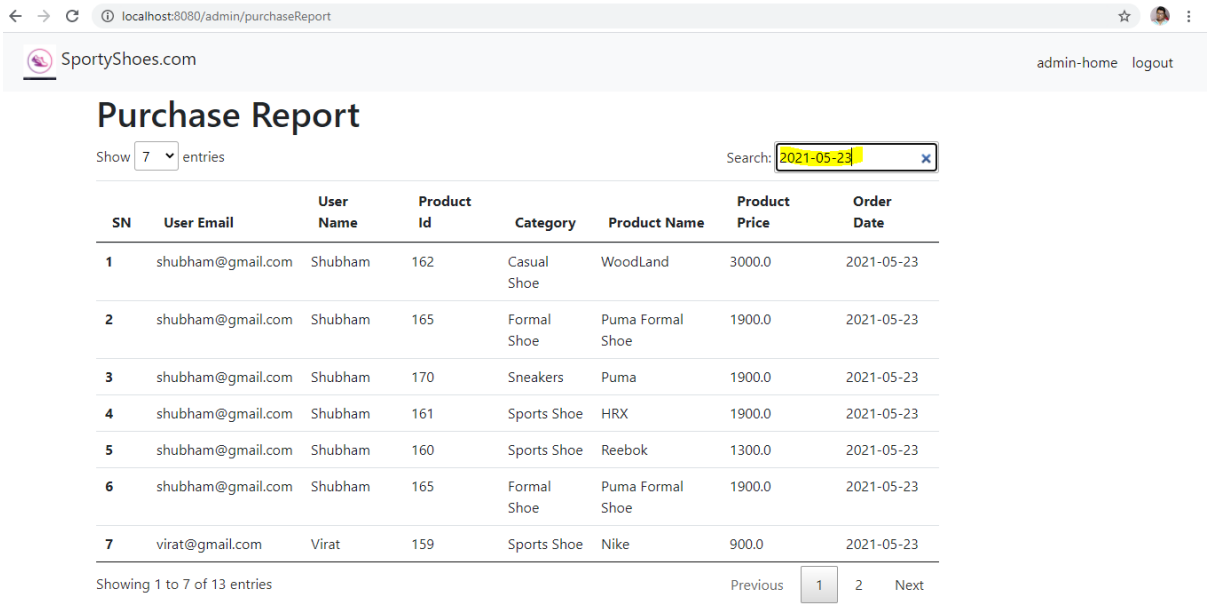




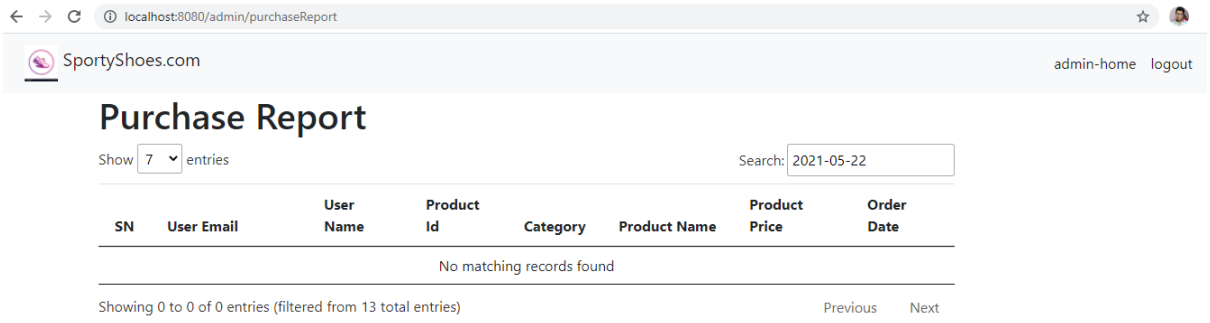
Purchase report filtered by category (Formal Shoe)



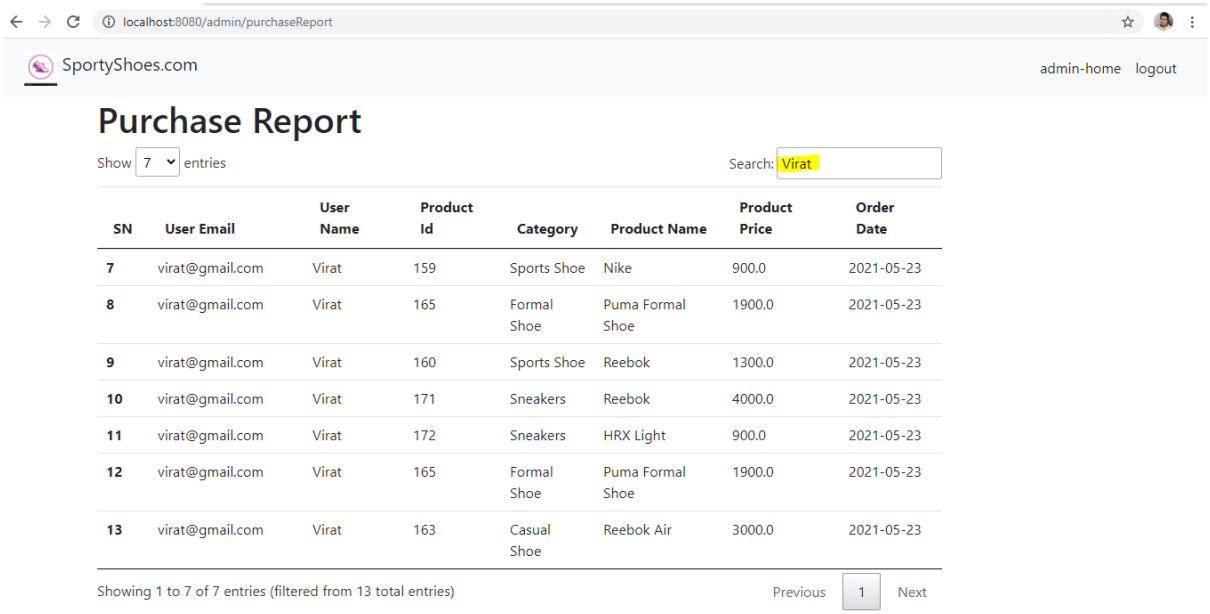
Purchase report filtered by Order date



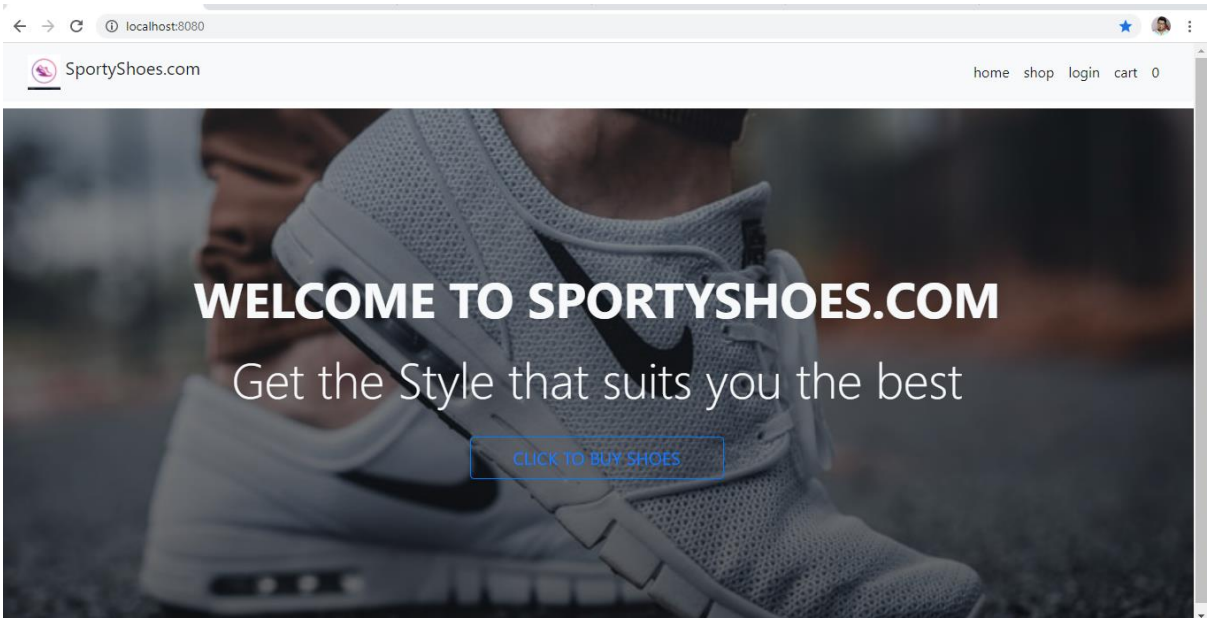
If there is no match present with respect to filter criteria then no records are returned



Purchase report filtered by User Name



When Admin Logouts of the system they are redirected to Home Page



AdminController

```
package com.project.sportyshoes.controller;


import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;


import com.project.sportyshoes.dto.ProductDTO;
```

```
import com.project.sportyshoes.global.PurchaseReport;
import com.project.sportyshoes.model.Category;
import com.project.sportyshoes.model.Product;
import com.project.sportyshoes.model.Purchase;
import com.project.sportyshoes.model.User;
import com.project.sportyshoes.repository.UserRepository;
import com.project.sportyshoes.service.CategoryService;
import com.project.sportyshoes.service.ProductService;
import com.project.sportyshoes.service.PurchaseService;

@Controller
public class AdminController {

	public static String uploadDir = System.getProperty("user.dir") +
"/src/main/resources/static/productImages";

	@Autowired
	CategoryService categoryService;

	@Autowired
	ProductService productService;
```

```
	@Autowired
	PurchaseService purchaseService;

	@Autowired
	UserRepository userRepository;

	@GetMapping("/admin")
	public String adminHome() {

		return "adminHome";
	}

	@GetMapping("/admin/categories")
	public String getCat(Model model) {
		model.addAttribute("categories" ,
categoryService.getAllCategories());
		return "categories";
	}

	@GetMapping("/admin/categories/add")
	public String getCatAdd(Model model) {
		model.addAttribute("category" , new Category());
```

```
        return "categoriesAdd";
    }

    @PostMapping("/admin/categories/add")
    public String postCatAdd(@ModelAttribute("category") Category category) {
        categoryService.addCategory(category);
        return "redirect:/admin/categories";
    }

    @GetMapping("/admin/categories/delete/{id}")
    public String deleteCat(@PathVariable int id) {

        categoryService.removeCategoryById(id);

        return "redirect:/admin/categories";
    }


    @GetMapping("/admin/categories/update/{id}")
    public String updateCat(@PathVariable int id , Model model) {
```

```
        Optional<Category> category =
categoryService.getCatById(id);

        if(category.isPresent()) {
            model.addAttribute("category" , category.get());
            return "categoriesAdd";
        }else {
            return "404";
        }

    }

// Product Section


    @GetMapping("/admin/products")
    public String showProducts(Model model) {

        model.addAttribute("products" ,
productService.getAllProducts());

        return "products";
    }
```

```
	@GetMapping("/admin/products/add")
	public String addProduct(Model model) {

		model.addAttribute("productDTO" , new ProductDTO());
		model.addAttribute("categories" , 
categoryService.getAllCategories());
		return "productsAdd";
	}

	@PostMapping("/admin/products/add")
	public String productAddPost(@ModelAttribute ("productDTO") 
ProductDTO productDTO ,

	@RequestParam("productImage") MultipartFile file,

	@RequestParam("imgName") String imgName) throws 
IOException {

		Product product = new Product();

		product.setId(productDTO.getId());
```

```
            product.setName(productDTO.getName());

        product.setCategory(categoryService.getCatById(productDTO.get
CategoryId()).get());
            product.setPrice(productDTO.getPrice());
            product.setWeight(productDTO.getWeight());
            product.setDescription(productDTO.getDescription());
            String imageUUID;
            if (!file.isEmpty()) {
                imageUUID = file.getOriginalFilename();
                Path fileNameAndPath = Paths.get(uploadDir,
imageUUID);
                Files.write(fileNameAndPath, file.getBytes());
            }else {

                imageUUID = imgName;
            }

            product.setImageName(imageUUID);
            productService.addProduct(product);



            return "redirect:/admin/products";
```

```
}


@GetMapping("/admin/product/delete/{id}")
public String deleteProd(@PathVariable long id) {

        productService.removeProductById(id);

        return "redirect:/admin/products";
}

@GetMapping("/admin/product/update/{id}")
public String updateProd(@PathVariable long id , Model model) {
        Product product = productService.getProductById(id).get();

        ProductDTO productDTO = new ProductDTO();
        productDTO.setId(product.getId());
        productDTO.setName(product.getName());
        productDTO.setCategoryId(product.getCategory().getId());
        productDTO.setPrice(product.getPrice());
        productDTO.setWeight(product.getWeight());
```

```
            productDTO.setDescription(product.getDescription());
            productDTO.setImageName(product.getImageName());

            model.addAttribute("productDTO" , productDTO);
            model.addAttribute("categories" ,
categoryService.getAllCategories());

            return "productsAdd";
        }


        //Purchase Report
        @GetMapping("/admin/purchaseReport")
        public String purchaseReport(Model model) {

            List<Purchase>purchaseList =
purchaseService.getAllPurchases();


            List<PurchaseReport>purchaseReportList = new
ArrayList<PurchaseReport>();

            for(Purchase pur : purchaseList) {
```

```
                Product product = new Product();
                User user = new User();
                Category category = new Category();
                PurchaseReport purchaseReport =  new
PurchaseReport();
                Long productId = pur.getProductId();
                int userId = pur.getUserId();

                product =
productService.getProductById(productId).get();
                user = userRepository.findById(userId).get();
                category = product.getCategory();

                purchaseReport.setEmail(user.getEmail());
                purchaseReport.setName(user.getFirstName());
                purchaseReport.setProductId(product.getId());
                purchaseReport.setProductName(product.getName());
                purchaseReport.setPrice(product.getPrice());

        purchaseReport.setDate(pur.getOrderDate().toString());
                purchaseReport.setCategory(category.getName());
                //System.out.println(purchaseReport.getEmail() + " " +
purchaseReport.getProductId() + "  " + purchaseReport.getDate());
```

```
                purchaseReportList.add(purchaseReport);

        }

        model.addAttribute("puchaseList" , purchaseReportList);



        return "purchaseReport";
}

@GetMapping("/admin/users")
public String listUsers(Model model) {



        List<User>userList = userRepository.findAll();
        model.addAttribute("userlist" , userList);

        return "userList";
}
```

}

CartController

```
package com.project.sportyshoes.controller;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

import com.project.sportyshoes.global.GlobalData;
import com.project.sportyshoes.model.Product;
```

```
import com.project.sportyshoes.model.Purchase;
import com.project.sportyshoes.model.User;
import com.project.sportyshoes.repository.PurchaseRepository;
import com.project.sportyshoes.repository.UserRepository;
import com.project.sportyshoes.service.ProductService;

@Controller
public class CartController {

	@Autowired
	ProductService productService;

	@Autowired
	UserRepository userRepository;

	@Autowired
	PurchaseRepository purchaseRepository;

	@GetMapping("/addToCart/{id}")
	public String addToCart(@PathVariable int id) {
		GlobalData.cart.add(productService.getProductById(id).get());
		return "redirect:/shop";
	}
```

```
	@GetMapping("/cart")
	public String cartGet(Model model) {
		model.addAttribute("cartCount" , GlobalData.cart.size());
		model.addAttribute("total" , GlobalData.cart.stream().mapToDouble(Product::getPrice).sum());
		model.addAttribute("cart" , GlobalData.cart);
		return "cart";



	}

	@GetMapping("/cart/removeItem/{index}")
	public String cartItemRemove(@PathVariable int index) {
		GlobalData.cart.remove(index);
		return "redirect:/cart";
	}

	@GetMapping("/checkout")
	public String checkout(Model model) {
		model.addAttribute("total" , GlobalData.cart.stream().mapToDouble(Product::getPrice).sum());
		return "checkout";
	}

	@PostMapping("/payNow")
```

```
public String orderConfirmation(Model model) {
		model.addAttribute("total" , GlobalData.cart.stream().mapToDouble(Product::getPrice).sum());
		Authentication auth = SecurityContextHolder.getContext().getAuthentication();
		String currentPrincipalName = auth.getName();

		List<Purchase>purchaseList = new ArrayList<Purchase>();
		//System.out.println(currentPrincipalName);
		User user = userRepository.findUserByEmail(currentPrincipalName).get();
		for(Product product: GlobalData.cart) {
			Purchase purchase = new Purchase();
			//System.out.println(product.getId() + " " + product.getName());
			purchase.setProductId(product.getId());
			purchase.setUserId(user.getId());
			purchase.setOrderDate(LocalDate.now());
			purchaseList.add(purchase);
		}

		int n = 10000 + new Random().nextInt(90000);
		model.addAttribute("Reciept" , n);
		//System.out.println(purchaseList.toString());
		model.addAttribute("products" , GlobalData.cart);
		purchaseRepository.saveAll(purchaseList);
```

```
            return "orderPlaced";
        }


}
```

HomeController

```
package com.project.sportyshoes.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import com.project.sportyshoes.global.GlobalData;
import com.project.sportyshoes.service.CategoryService;
import com.project.sportyshoes.service.ProductService;

@Controller
public class HomeController {

        @Autowired
```

```
	CategoryService categoryService;

	@Autowired
	ProductService productService;

	@GetMapping({"/" , "/home"})
	public String home(Model model) {
		model.addAttribute("cartCount" , GlobalData.cart.size());
		return "index";
	}

	@GetMapping("/shop")
	public String shop(Model model) {
		model.addAttribute("cartCount" , GlobalData.cart.size());
		model.addAttribute("categories" , categoryService.getAllCategories());
		model.addAttribute("products" , productService.getAllProducts());

		return "shop";
	}

	@GetMapping("/shop/category/{id}")
	public String shopByCategory(Model model , @PathVariable int id) {
		model.addAttribute("cartCount" , GlobalData.cart.size());
```

```
		model.addAttribute("categories" , categoryService.getAllCategories());
		model.addAttribute("products" , productService.getAllProductsByCategoryId(id));

		return "shop";
	}

	@GetMapping("/shop/viewproduct/{id}")
	public String viewProduct(Model model , @PathVariable long id) {
		model.addAttribute("product" , productService.getProductById(id).get());
		model.addAttribute("cartCount" , GlobalData.cart.size());
		return "viewProduct";
	}


}
```

LoginController

```
package com.project.sportyshoes.controller;

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

import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.project.sportyshoes.global.GlobalData;
import com.project.sportyshoes.model.Role;
import com.project.sportyshoes.model.User;
import com.project.sportyshoes.repository.RoleRepository;
import com.project.sportyshoes.repository.UserRepository;

@Controller
public class LoginController {

	@Autowired
	private BCryptPasswordEncoder bCryptPasswordEncoder;

	@Autowired
	UserRepository userRepository;
```

```
	@Autowired
	RoleRepository roleRepository;

	@GetMapping("/login")
	public String login() {
		GlobalData.cart.clear();
		return "login";
	}

	@GetMapping("/register")
	public String registerGet() {
		return "register";
	}

	@PostMapping("/register")
	public String registerPost(@ModelAttribute("user") User user ,
HttpServletRequest request) throws ServletException {

		String passoword = user.getPassword();

		user.setPassword(bCryptPasswordEncoder.encode(passoword));
		List<Role>roles = new ArrayList<Role>();
		roles.add(roleRepository.findById(2).get());
		user.setRoles(roles);
		userRepository.save(user);
```

```
            request.login(user.getEmail(), passoword);
            return "redirect:/";

        }

}
```

SecurityConfig

```
package com.project.sportyshoes.configuration;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authenti
cationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.builders.WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSe
curity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityC
onfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import com.project.sportyshoes.service.CustomUserDetailService;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {



	@Autowired
	CustomUserDetailService customUserDetailService;

	@Override
	protected void configure(HttpSecurity http) throws Exception {

		http
				.authorizeRequests()
				.antMatchers("/" , "/shop/**" , "/forgotpassword" , "/register" , "/h2-console/**" ).permitAll()
				.antMatchers("/admin/**").hasRole("ADMIN")
				.anyRequest()
				.authenticated()
				.and()
				.formLogin()
				.loginPage("/login")
```

```
                .permitAll()
                .failureUrl("/login?error=true")
                .defaultSuccessUrl("/")
                .usernameParameter("email")
                .passwordParameter("password")
                .and()
                .logout()
                .logoutRequestMatcher(new
AntPathRequestMatcher("/logout"))
                .logoutSuccessUrl("/login")
                .invalidateHttpSession(true)
                .deleteCookies("JSESSIONID")
                .and()
                .exceptionHandling()
                .and()
                .csrf()
                .disable();

        http.headers().frameOptions().disable();
        }

        @Bean
        public BCryptPasswordEncoder bCryptPasswordEncoder() {
                return new BCryptPasswordEncoder();
        }
```

```
	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws
Exception {

		auth.userDetailsService(customUserDetailService);

	}

	@Override
	public void configure(WebSecurity web) throws Exception {

		web.ignoring().antMatchers("/resources/**" , "/static/**" ,
"/images/**" , "/productImages/**" , "/css/**" , "/js/**")
		;
	}









}
```

ProductDTO

```
package com.project.sportyshoes.dto;


public class ProductDTO {


	private Long id;
	private String name;
	private int categoryId;
	private double price;
	private double weight;
	private String description;
	private String imageName;


	public Long getId() {
		return id;
	}


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


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

```
public void setName(String name) {
        this.name = name;
}

public int getCategoryId() {
        return categoryId;
}

public void setCategoryId(int categoryId) {
        this.categoryId = categoryId;
}

public double getPrice() {
        return price;
}

public void setPrice(double price) {
        this.price = price;
}

public double getWeight() {
        return weight;
}
```

```
	public void setWeight(double weight) {
		this.weight = weight;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getImageName() {
		return imageName;
	}

	public void setImageName(String imageName) {
		this.imageName = imageName;
	}

}
```

PurchaseReport

```
package com.project.sportyshoes.global;
```

```
import org.springframework.stereotype.Component;

@Component
public class PurchaseReport {

	private String email;

	private String name;

	private Long productId;

	private String productName;

	private double price;

	private String date;

	private String category;

	public String getCategory() {
		return category;
	}
```

```
public void setCategory(String category) {
	this.category = category;
}

public String getEmail() {
	return email;
}

public void setEmail(String email) {
	this.email = email;
}

public String getName() {
	return name;
}

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

public Long getProductId() {
	return productId;
}
```

```
public void setProductId(Long productId) {
	this.productId = productId;
}

public String getProductName() {
	return productName;
}

public void setProductName(String productName) {
	this.productName = productName;
}

public double getPrice() {
	return price;
}

public void setPrice(double price) {
	this.price = price;
}

public String getDate() {
	return date;
}
```

```
	public void setDate(String date) {
		this.date = date;
	}
```

```
}
```

Model:

Category

```
package com.project.sportyshoes.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Category {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "category_id")
```

```
	private int id;

	private String name;

	public int getId() {
		return id;
	}

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

	public String getName() {
		return name;
	}

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

}
```

CustomUserDetail

```
package com.project.sportyshoes.model;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

public class CustomUserDetail extends User implements UserDetails {

	public CustomUserDetail(User user) {
		super(user);
	}

	@Override
	public Collection<? extends GrantedAuthority> getAuthorities() {

		List<GrantedAuthority>authorityList = new
ArrayList<GrantedAuthority>();

		super.getRoles().forEach(role -> {
```

```
                        authorityList.add(new
SimpleGrantedAuthority(role.getName()));
                });

                return authorityList;
        }

        @Override
        public String getPassword() {
                // TODO Auto-generated method stub
                return super.getPassword();
        }

        @Override
        public String getUsername() {
                // TODO Auto-generated method stub
                return super.getEmail();
        }

        @Override
        public boolean isAccountNonExpired() {
                // TODO Auto-generated method stub
                return true;
        }
```

```
	@Override
	public boolean isAccountNonLocked() {
		// TODO Auto-generated method stub
		return true;
	}

	@Override
	public boolean isCredentialsNonExpired() {
		// TODO Auto-generated method stub
		return true;
	}

	@Override
	public boolean isEnabled() {
		// TODO Auto-generated method stub
		return true;
	}



}
```

Product

```
package com.project.sportyshoes.model;
```

```
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;



@Entity
public class Product {

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

	private String name;



	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "category_id" , referencedColumnName =
"category_id")
	private Category category;

	private double price;
```

```
private double weight;

private String description;

private String imageName;

public Long getId() {
        return id;
}

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

public String getName() {
        return name;
}

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

public Category getCategory() {
```

```
        return category;
}

public void setCategory(Category category) {
        this.category = category;
}

public double getPrice() {
        return price;
}

public void setPrice(double price) {
        this.price = price;
}

public double getWeight() {
        return weight;
}

public void setWeight(double weight) {
        this.weight = weight;
}

public String getDescription() {
```

```
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getImageName() {
		return imageName;
	}

	public void setImageName(String imageName) {
		this.imageName = imageName;
	}



}
```

Purchase

```
package com.project.sportyshoes.model;

import java.time.LocalDate;

import javax.persistence.Column;
```

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Purchase {
	@Id
	@GeneratedValue(strategy = GenerationType.SEQUENCE)
	private long id;



	private int userId;

	private Long productId;

	@Column(name = "order_date")
	private LocalDate orderDate;

	public long getId() {
		return id;
	}

	public void setId(long id) {
```

```
        this.id = id;
}

public int getUserId() {
        return userId;
}

public void setUserId(int userId) {
        this.userId = userId;
}

public Long getProductId() {
        return productId;
}

public void setProductId(Long productId) {
        this.productId = productId;
}

public LocalDate getOrderDate() {
        return orderDate;
}

public void setOrderDate(LocalDate orderDate) {
```

```
            this.orderDate = orderDate;
        }



}
```

Role

```
package com.project.sportyshoes.model;

import java.util.List;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotEmpty;

@Entity
@Table(name = "roles")
public class Role {
```

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

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

@ManyToMany(mappedBy = "roles")
private List<User>users;

public Integer getId() {
	return id;
}

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

public String getName() {
	return name;
}

public void setName(String name) {
```

```
		this.name = name;
	}

	public List<User> getUsers() {
		return users;
	}

	public void setUsers(List<User> users) {
		this.users = users;
	}

}
```

User

```
package com.project.sportyshoes.model;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;

@Entity
@Table(name = "users")
public class User {

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

	@NotEmpty
	@Column(nullable = false)
	private String firstName;



	private String lastName;

	@NotEmpty
	@Column(nullable = false , unique = true)
```

```
	@Email(message = "{errors.invalid_email}")
	private String email;

	private String password;

	@ManyToMany(cascade = CascadeType.MERGE , fetch = FetchType.EAGER)
	@JoinTable(

			name = "user_role",
			joinColumns = {@JoinColumn( name = "USER_ID" , referencedColumnName = "ID" )},
			inverseJoinColumns = {@JoinColumn( name = "ROLE_ID" , referencedColumnName = "ID" )}

			)
	private List<Role>roles;

	public Integer getId() {
		return id;
	}

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

```
public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}

public String getLastName() {
	return lastName;
}

public void setLastName(String lastName) {
	this.lastName = lastName;
}

public String getEmail() {
	return email;
}

public void setEmail(String email) {
	this.email = email;
}
```

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

public void setPassword(String passsword) {
	this.password = passsword;
}

public List<Role> getRoles() {
	return roles;
}

public void setRoles(List<Role> roles) {
	this.roles = roles;
}

public User(User user) {
	super();
	this.firstName = user.getFirstName();
	this.lastName = user.getLastName();
	this.email = user.getEmail();
	this.password = user.getPassword();
	this.roles = user.getRoles();
}
```

```
    public User() {}

}
```

Repository :

CategoryRepository

```
package com.project.sportyshoes.repository;

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

import com.project.sportyshoes.model.Category;

public interface CategoryRepository extends JpaRepository<Category, Integer> {

}
```

ProductRepository

```
package com.project.sportyshoes.repository;
```

```
import java.util.List;

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

import com.project.sportyshoes.model.Product;

public interface ProductRepository extends JpaRepository<Product, Long> {

	List<Product> findAllByCategory_Id(int id);

}
```

PurchaseRepository

```
package com.project.sportyshoes.repository;

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

import com.project.sportyshoes.model.Purchase;

public interface PurchaseRepository extends JpaRepository<Purchase, Long> {

}
```

RoleRepository

```
package com.project.sportyshoes.repository;
```

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

import com.project.sportyshoes.model.Role;



public interface RoleRepository extends JpaRepository<Role, Integer> {



}
```

UserRepository

```
package com.project.sportyshoes.repository;

import java.util.Optional;

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

import com.project.sportyshoes.model.User;



public interface UserRepository extends JpaRepository<User, Integer> {

	Optional<User>findUserByEmail(String email);
}
```