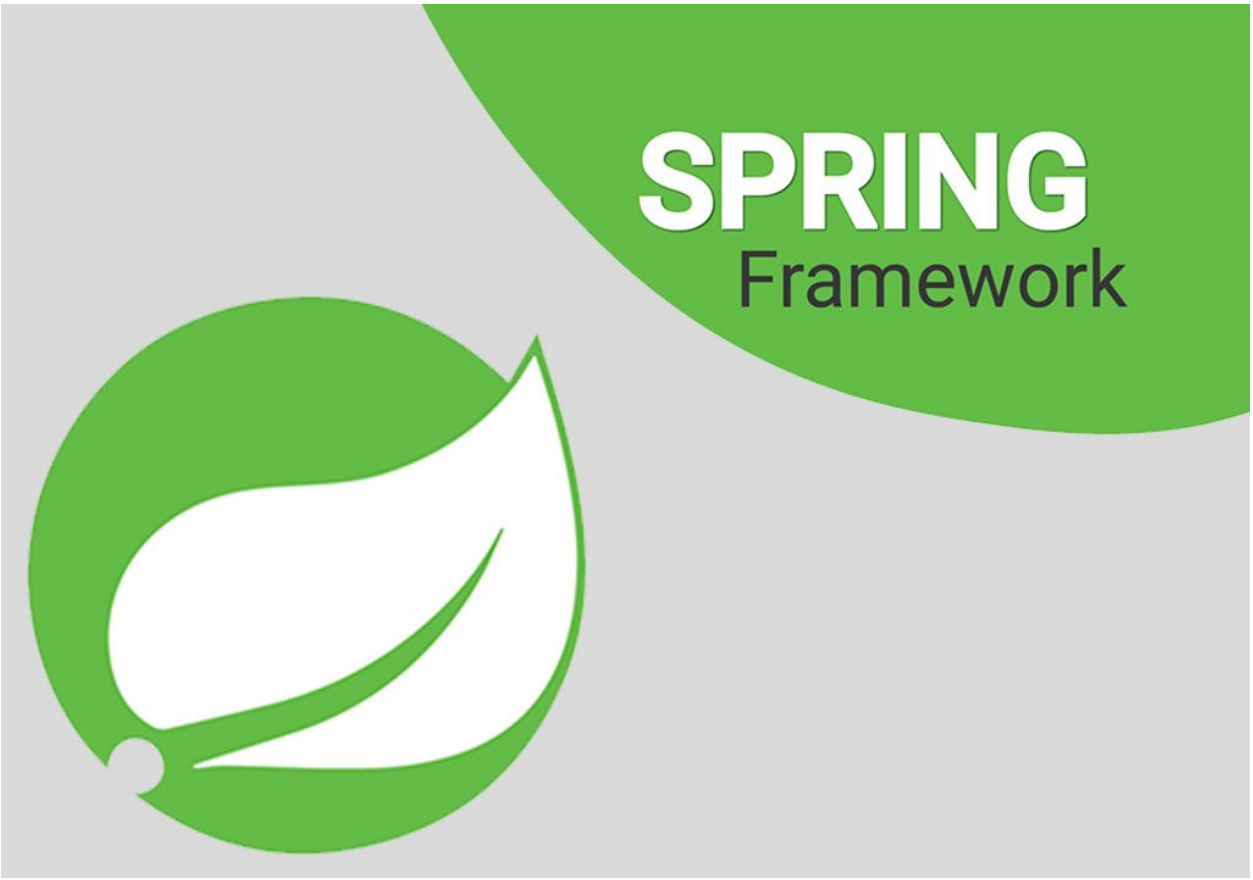


Spring Apparel Store

23.04.2022

Atanu Ghosh
001910501005
IT Assignment-4
JU BCSE UG-3 (A1)

Problem Statement

Design an online apparel store using servlets. The store keeps records for its items in a database where some items may be discounted and some other items should be displayed as “new arrivals”. A user may search for a specific item. By default, when a user signs in, based on his/her profile (male/female etc.), show him/her preferred set of clothing. Users will be divided into two groups: some users looking for seasonal clothing items mainly, some others looking for new arrivals. So, depending on their preference already set in the database, their shopping experience would be different. Show the user products of the price range based on his/her purchase history. You may apply the concept of session and cookies for tracking user behaviour. Build this application using the Spring framework.

Specifications

- JDK Version used for this project - Java SE 16.0.2
- Spring Boot Version - 2.6.6
- Maven Version - 4.0.0
- Apache Tomcat Version - 9.0.60

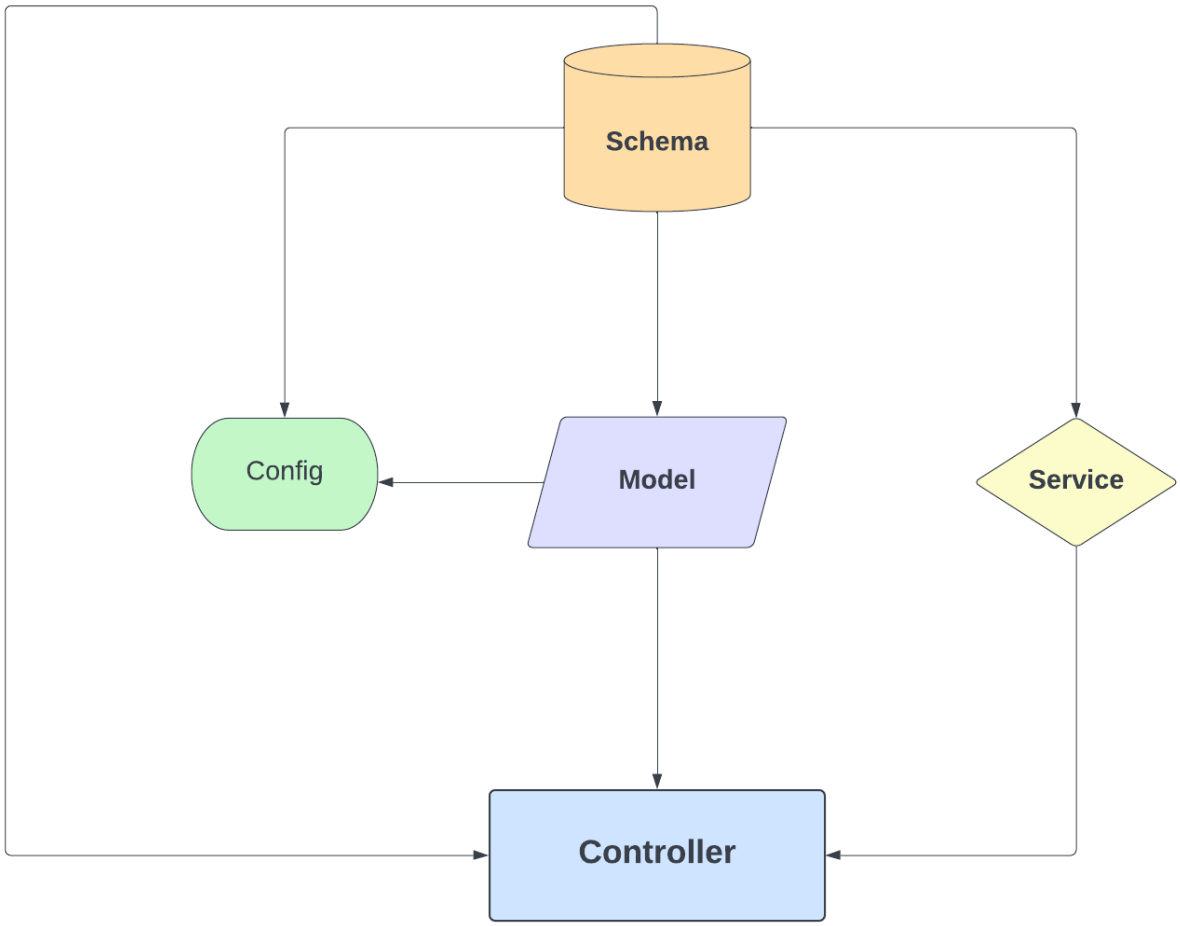
Introduction

This project is implemented using **Spring Boot Framework** which uses Java Programming Language as its core. **Hibernate** is used as a medium to connect to the database in an object-oriented way. Database tables get mapped as Java Objects automatically. (It can be thought of as a replacement for **JDBC**, though it uses **JDBC** internally but that fact is hidden from the developer.

Design

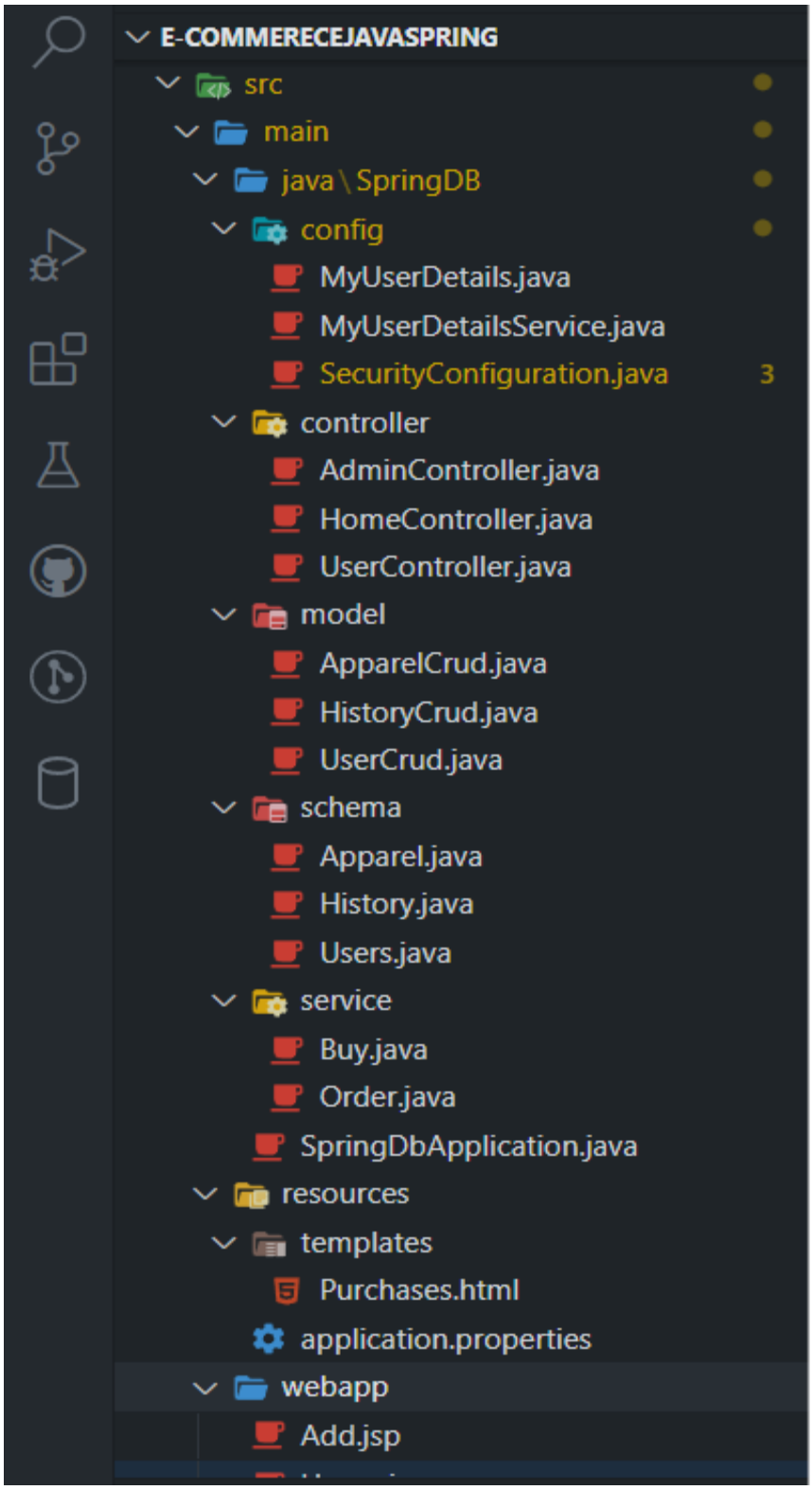
I. Flow Diagram

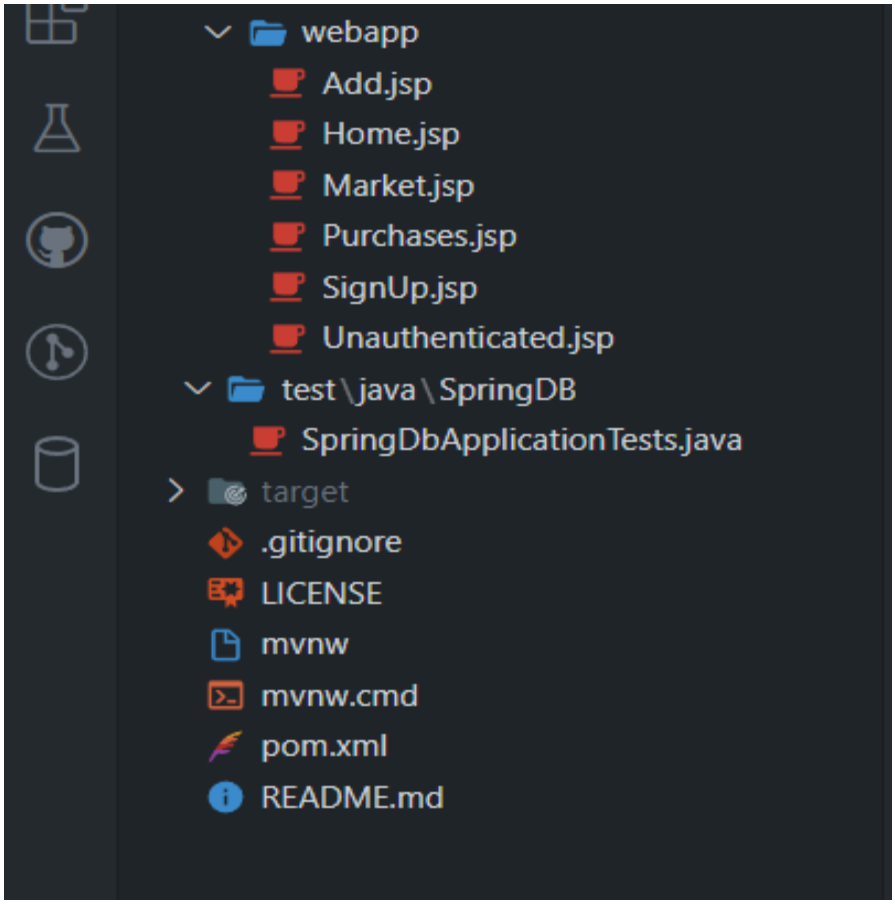
The basic flow of this project is represented in the schematic diagram below. **`Config`** files are written using the **`Schema`** and the **`Model`**. The **`Controller`** uses **`Schema`** and **`Service`** as its utility imports. On the other hand, **`Service`** uses **`Schema`** as its supporting folder. So, implicit or explicitly all the parts of this project use **`Schema`** as its supporting import.



II. Source Code Structure / Folder Structure

The basic overview of this project folder is pasted below.


E-COMMERECEJAVASPRING
src
main
java\SpringDB
config
MyUserDetails.java
MyUserDetailsService.java
SecurityConfiguration.java
3
controller
AdminController.java
HomeController.java
UserController.java
model
ApparelCrud.java
HistoryCrud.java
UserCrud.java
schema
Apparel.java
History.java
Users.java
service
Buy.java
Order.java
SpringDbApplication.java
resources
templates
Purchases.html
application.properties
webapp
Add.jsp



Implementation

1. Schema

a. Apprel.java

- Contains the Apparel Class with getter and setter methods of Price, Apparel ID, Brands, Category, Type and Gender.

b. History.java

- Contains the History Class with getter and setter methods of Newly arrived Items, Seasonal Items and Users.

c. Users.java

- Contains Users Class with getter and setter methods about their Purchase History, Gender, User ID, Name, Email, Password, Role of specific Users (Admin or Normal Customer).

2. Model

a. ApparelCrud.java

- Contains Template Class for searching and soring different apparel available in the store.

b. HistoryCrud.java

- Contains Template Class for JpaRepository (Adds the bought item to a specific user's buying history).

c. UserCrud.java

- Contains Template Class for JpaRepository (Checks whether the User exists in Database by searching with the entered email Id of that user).

3. Config

a. MyUserDetails.java

- Contains Method for getting User Credentials (UserID, Email, Password, Gender).
- Contains Method for Expired Sessions or Expired Credentials.

```
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

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

import SpringDB.schema.Users;

You, 22 hours ago | 1 author (You)
public class MyUserDetails implements UserDetails {

    int userId;
    String userName;
    String password;
    String gender;
    private List<GrantedAuthority> authorities; // List of roles of users that are authenticated

    public MyUserDetails() {
    }

    // Instantiates the username
    public MyUserDetails(String uname) {
        this.userName = uname;
    }

    // Overrides prev method to fetch user details
    // authorities => gets list of comma-separated roles of the authenticated users
    public MyUserDetails(Users u) {
        userId = u.getUserId();
        userName = u.getEmail();
        password = u.getPassword();
        gender = u.getGender();
        authorities = Arrays.stream(u.getRole().split(regex: ","))
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());
        System.out.println(authorities);
    }
```

b. MyUserDetailsService.java

- Contains a method that Searches the DB to get the User by Corresponding User Mail ID.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import SpringDB.model.UserCrud;
import SpringDB.schema.Users;

You, 6 hours ago | 1 author (You)
@Service
public class MyUserDetailsService implements UserDetailsService {

    @Autowired
    UserCrud uc;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // TODO Auto-generated method stub
        // Users user = uc.findByEmail(username).orElse(new Users());
        Users user = uc.findByEmail(username).orElse(other: null);
        if (user == null)
            throw new UsernameNotFoundException(username);
        return new MyUserDetails(user);
    }
}
```

c. SecurityConfiguration.java

- Handles **login and logout** facility of Admins.
- Handles **SignUp, SignIn/login and logout** facility of Users.
- **Encodes** the User's Passwords to ensure security.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    UserDetailsService userDetailsService;

    /**
     * Handles authentication service to login to the admin account
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser( username: "admin")
                .password( password: "12345")
                .roles( ... roles: "ADMIN");
        auth.userDetailsService(userDetailsService);
    }

    /**
     *  Handles actions of users/admins based on the roles assigned to them
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers( ... antPatterns: "/user/**").hasRole( role: "USER")
                .antMatchers( ... antPatterns: "/admin/**").hasRole( role: "ADMIN")
                .antMatchers( ... antPatterns: "/**").permitAll()
                .and()
                .formLogin()
                .and().logout()
                .and().exceptionHandling().accessDeniedPage( accessDeniedUrl: "/accessdenied");
        http.cors().and().csrf().disable();
    }

    /**
     *
     * @Bean Indicates that a method produces a bean to be managed by the Spring container.
     *
     */
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList( ... a: "*"));
        configuration.setAllowedMethods(Arrays.asList( ... a: "*"));
        configuration.setAllowedHeaders(Arrays.asList( ... a: "*"));
        configuration.setAllowCredentials( allowCredentials: true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( pattern: "/**", configuration);
        return source;
    }
```

4. Service

a. Buy.java

- If the user buys apparel, associate it with the user and set the order type according to the items bought.

```java
import java.util.Set;
import org.springframework.stereotype.Service;
import SpringDB.schema.Apparel;
import SpringDB.schema.Users;

You, 22 hours ago | 1 author (You)
@Service
public class Buy {

    // If user buys an apparel, associate it with the user and set the order type
    // according to the items bought
    public Users buy(Apparel a, Users u) {
        Order o = new Order();
        Set<Apparel> ap = u.getAp();
        ap.add(a);
        u.setAp(ap);
        if (a.getType().equals(anObject: "Seasonal"))
            o.seasonal(u);
        else
            o.newArrival(u);
        return u;
    }
}
```

b. Order.java

- If the user has no history (neither seasonal nor new-arrival), create a new history and set the user accordingly.

- For seasonal/new-arrived items, create the item, increment the count, and set the user type as seasonal/new-arrived.

```
You, 22 hours ago | 1 author (You)
@Service
public class Order {

    // If the user has no history (neither seasonal nor new-arrival), create a new
    // history and set the user accordingly
    public void set(Users u) {
        History h = null;
        if (u.getH() == null) {
            h = new History();
            u.setH(h);
            h.setU(u);
        }
    }

    // For seasonal items, create the item, increment the count, and set the user
    // type as seasonal
    public Users seasonal(Users u) {
        set(u);
        History h = u.getH();
        h.incS();
        u.setH(h);
        return u;
    }

    // For new-arrival items, create the item, increment the count, and set the user
    // type as new-arrival
    public Users newArrival(Users u) {
        set(u);
        History h = u.getH();
        h.incN();
        u.setH(h);
        return u;
    }
}
```

5. Controller

a. AdminController.java

- Contains support for adding and deleting and searching operations on apparel by admin,

```java
    @RequestMapping("/admin")
    public ModelAndView Add() {
        ModelAndView mv = new ModelAndView( viewName: "/Add.jsp");
        List<Apparel> l = ac.findAll();
        mv.addObject( attributeName: "apparels", l);
        return mv;
    }

    // Testing
    @RequestMapping("/admin/hello")
    public void display() {
        System.out.println( x: "whhy");
    }

    // Adds a new item for display on clicking AddApparel
    @RequestMapping(value = "/admin/addApparel", method = RequestMethod.POST, consumes = MediaType.APPLICATION_FORM_URLENCODED_VALUE)
    public ModelAndView Submit(Apparel ap) {
        ac.save(ap);
        ModelAndView mv = new ModelAndView( viewName: "/Add.jsp");
        List<Apparel> l = ac.findAll();
        mv.addObject( attributeName: "apparels", l);
        return mv;
    }

    // Deletes the item on clicking DeleteApparel
    @RequestMapping("/admin/deleteApparel")
    public ModelAndView delete(@RequestParam("id") int id) {
        ac.deleteById(id);
        ModelAndView mv = new ModelAndView( viewName: "/Add.jsp");
        List<Apparel> l = ac.findAll();
        mv.addObject( attributeName: "apparels", l);
        return mv;
    }

    // Gets the admin-searched item(s) if the searched string is a substring of any
    // of the listed item-details set by the admin itself
    @RequestMapping("/admin/Search")
    public ModelAndView search(@RequestParam("search") String search) {
        ModelAndView mv = new ModelAndView( viewName: "/Add.jsp");
        List<Apparel> l = ac.findBySearch(search);
        mv.addObject( attributeName: "apparels", l);
        return mv;
    }
```

b. HomeController.java

- Redirects to the Unauthentication Page if anyone of User/Admin tries to access each other's roles.

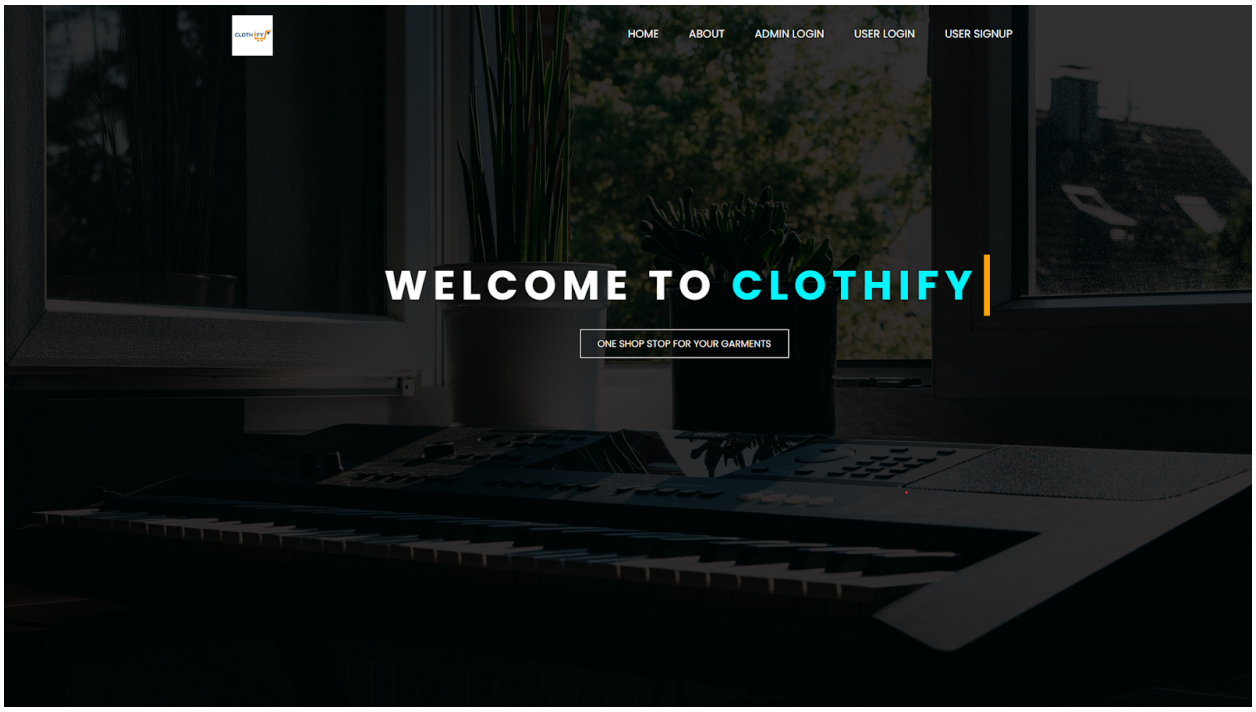
```java
@Controller
public class HomeController {

    @Autowired
    UserCrud u;

    @RequestMapping({ "/home", "/" })
    public String display() {
        return "Home.jsp";
    }

    // If not logged-in, deny the user to access any information and redirect to the
    // error 403 page
    @RequestMapping("accessdenied")
    public String unauthenticated() {
        return "Unauthenticated.jsp";
    }
}
```

c. UserController.java

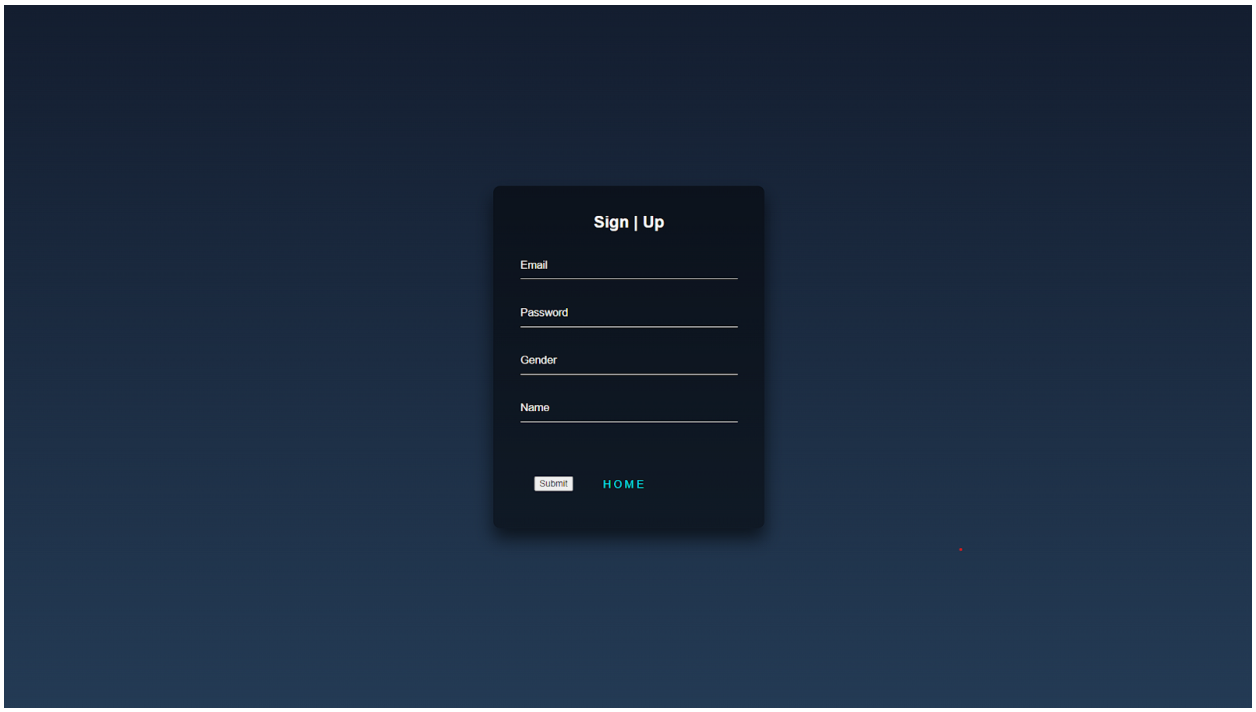
- Contains support for Sorting and searching and purchasing operations on apparel by users,

ScreenShots Of Functionalities and UI

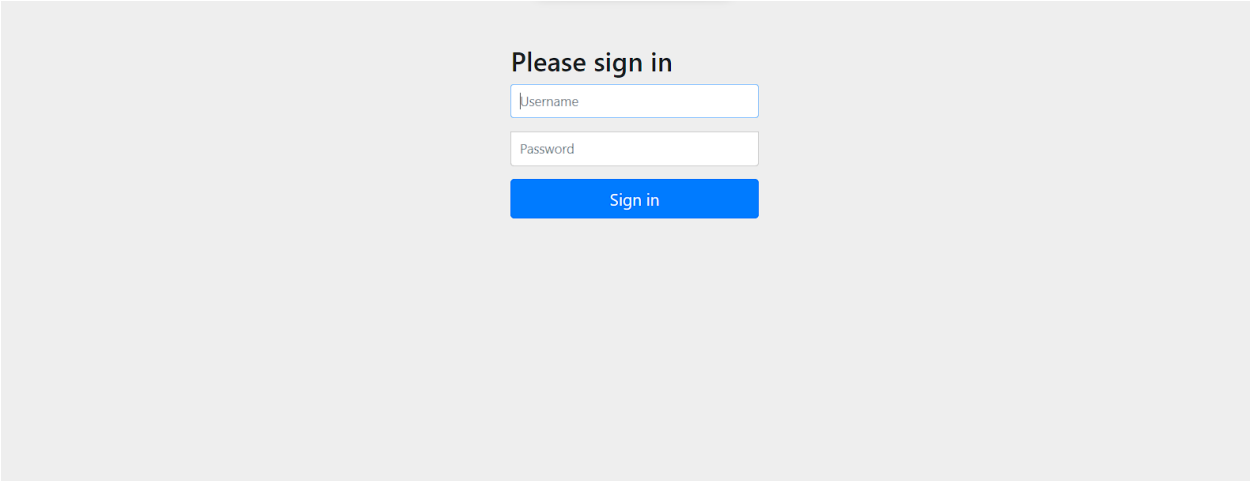
Landing Page



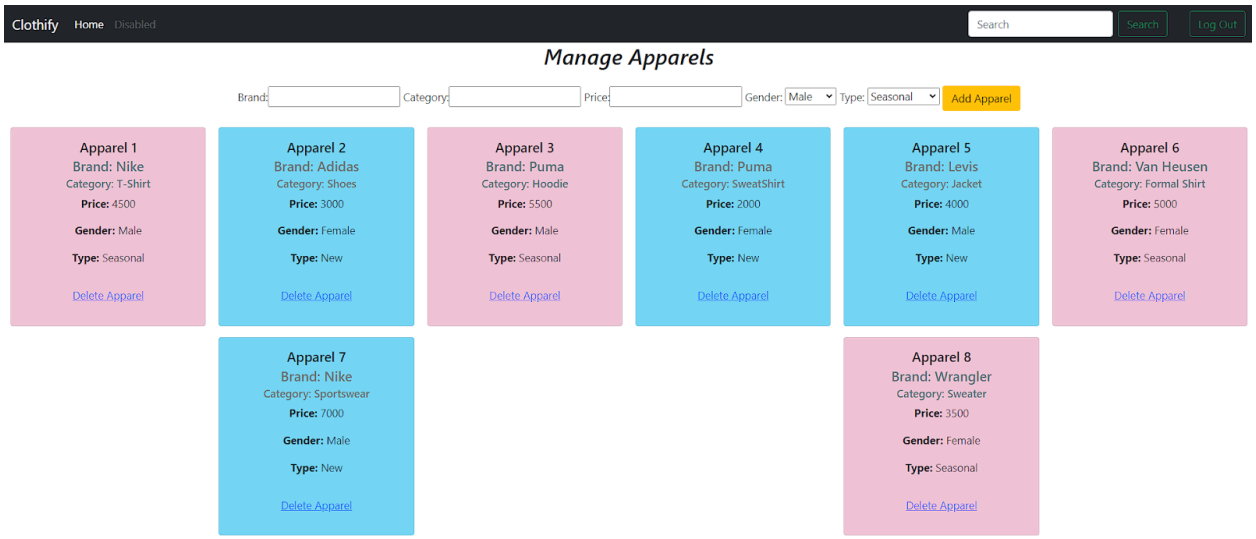
User SignUp Page



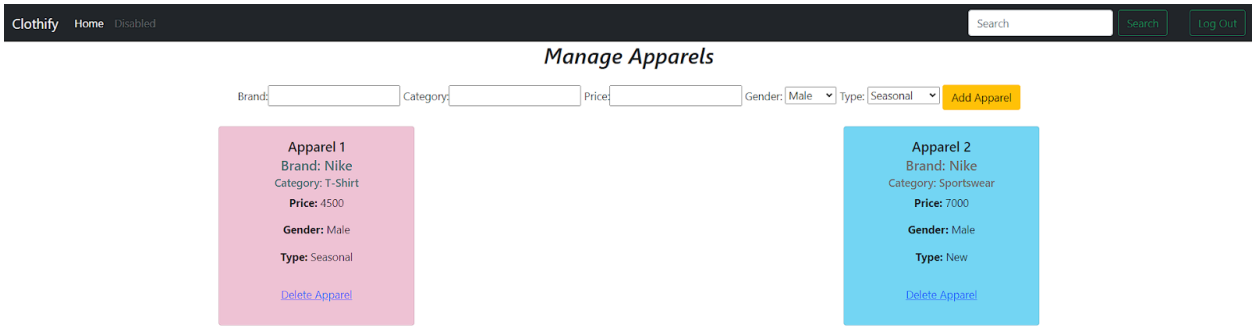
Admin/User login Page



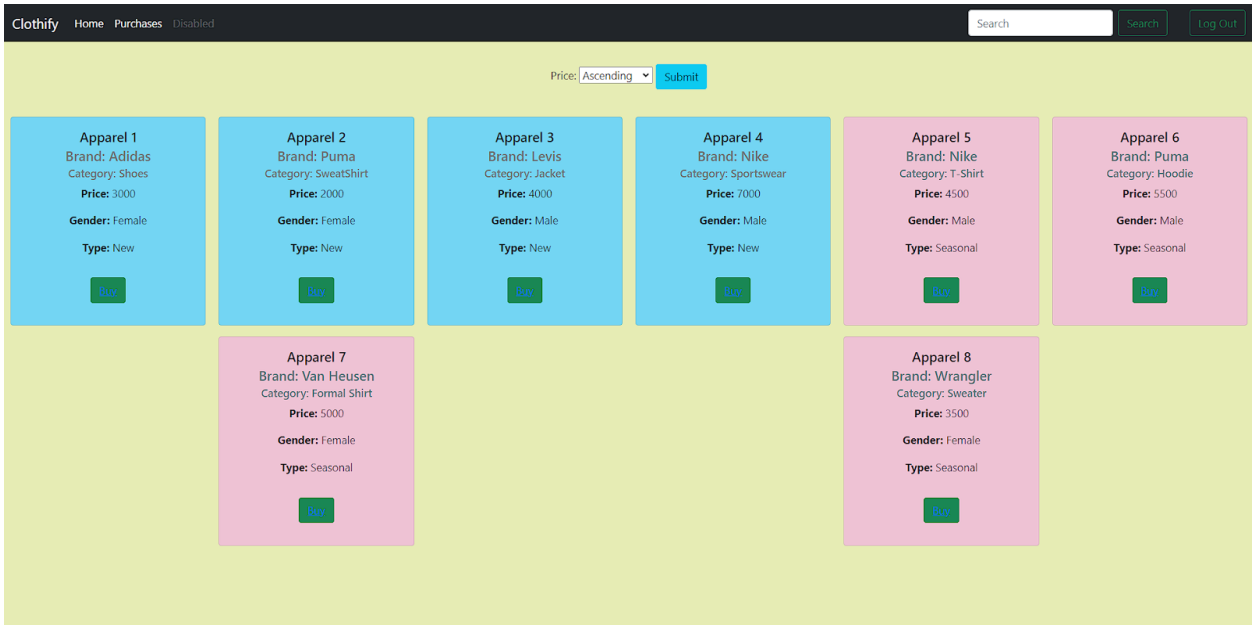
Admin's Apparel Add/Delete Page



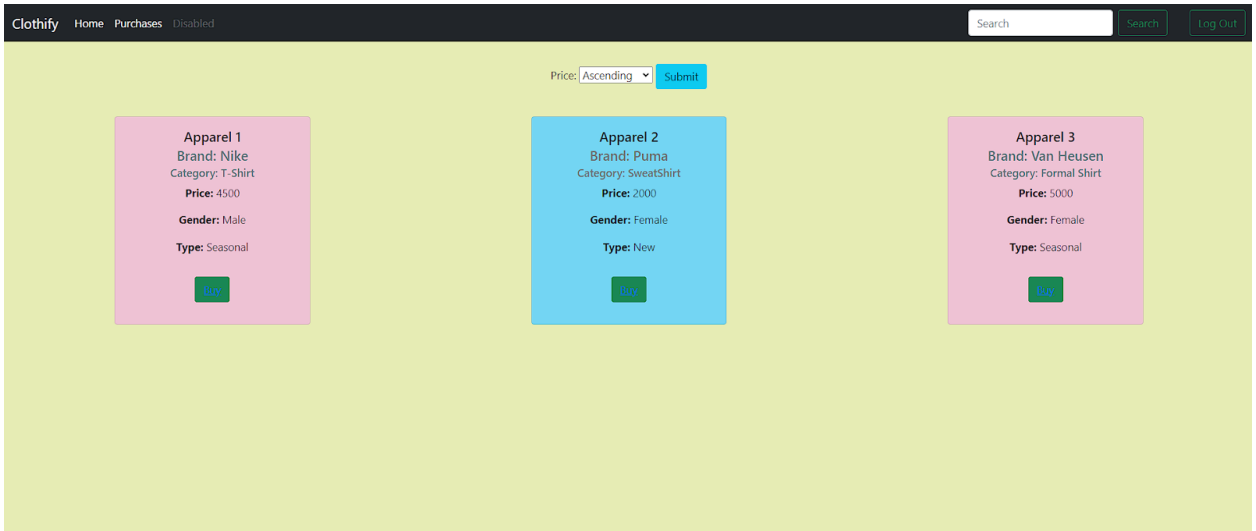
Admin's Apparel Search Page



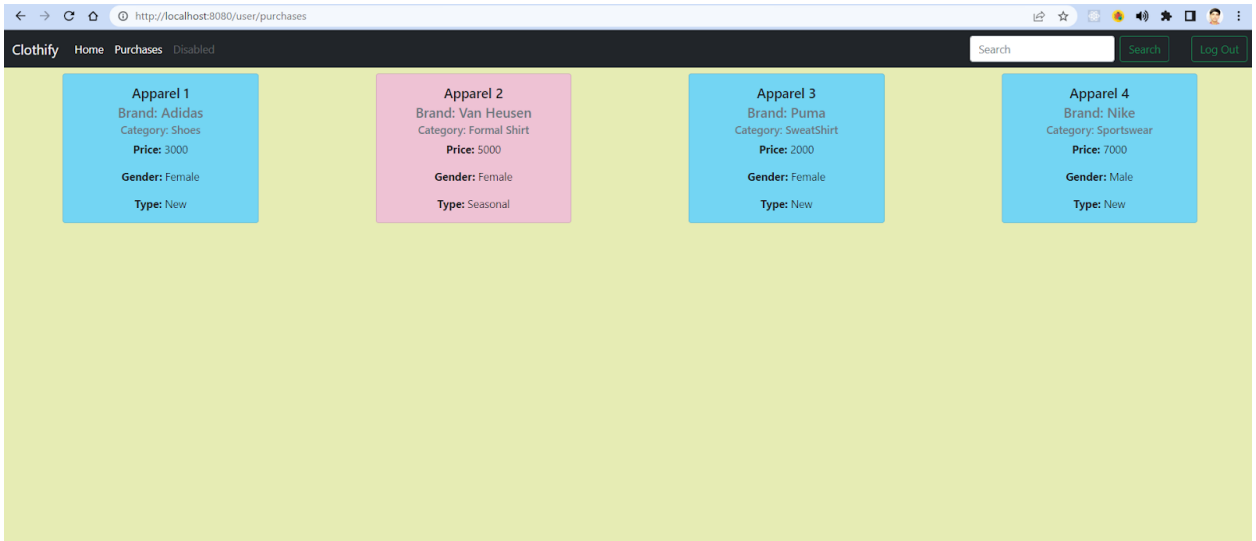
Users' Apparel List Page



Users' Apparel Search Page



Users' Purchase History Page



Scopes Of Improvement

- **Image Upload** functionality could be added against each apprel by the store admin, it'd help an user to get a visual overview of the product he/she is planning to buy.

- Support for setting the **price range** about any category of product could be added in order to give the user the exact affordable priced products that he/she is planning to buy.

- **Checkout** Page could be added.

Acknowledgement

I would like to give our profound thanks to **Prof. Chandreye Chowdhury** Ma'am and **Prof. Sarbani Roy** ma'am for providing us the opportunity to work on something which is very practical and essential for any computer science student. This was a great opportunity for all of us and I am highly grateful to and would welcome any further suggestions for the scope of improvement.