

MyMoviePlan

This document contains the following.

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

Description

Create a dynamic and responsive web application for booking movie tickets online for different genres and languages.

Background of the problem statement:

NMS Cinemas is a chain of single screen theatres that screen movie shows of different genres and languages at very genuine prices. It was established in 2004 in Pune, India. Recently, the business analysts noticed a decline in sales since 2010. They found out that the online booking of movie tickets from apps, such as BookMyShow and Paytm were gaining more profit by eliminating middlemen from the equation. As a result, the team decided to hire a Full Stack developer to develop an online movie ticket booking web application with a rich and user-friendly interface.

You are hired as the Full Stack Java developer and are asked to develop the web application. The management team has provided you with the requirements and their business model so that you can easily arrange different components of the application.

Features of the application:

1. Registration
2. Login
3. Payment gateway
4. Searching
5. Filtering
6. Sorting
7. Dynamic data
8. Responsive and compatible with different devices

Recommended technologies:

1. Database management: MySQL and Oracle
2. Backend logic: Java programming, NodeJS
3. Frontend development: JSP, Angular, Bootstrap, HTML/CSS, and Javascript
4. Automation and testing technologies: Selenium, Jasmine, and TestNG
5. DevOps and production technologies: Git, GitHub, Jenkins, Docker, Kubernetes, and AWS

Project development guidelines:

- The project will be delivered within four sprints with every sprint delivering a minimal viable product.
- It is mandatory to perform proper sprint planning with user stories to develop all the components of the project.
- The learner can use any technology from the above-mentioned technologies for different layers of the project.
- The web application should be responsive and should fetch or send data dynamically without hardcoded values.
- The learner must maintain the version of the application over GitHub and every new change should be sent to the repository.
- The learner must implement a CI/CD pipeline using Jenkins.
- The learner should also deploy and host the application on an AWS EC2 instance.
- The learner should also implement automation testing before the application enters the CI/CD pipeline.
- The learner should use Git branching to do basic automation testing of the application in it separately.
- The learner should make a rich frontend of the application, which is user- friendly and easy for the user to navigate through the application.
- There will be two portals in the application, namely admin and user portal.

Admin Portal:

It deals with all the backend data generation and product information. The admin user should be able to:

- Add or remove different genres to or from the application to build a rich product line
- Edit movie details like name, ticket price, language, description, and show timings to keep it aligned to the current prices
- Enable or disable the movie shows from the application

User Portal:

It deals with the user activities. The end-user should be able to:

- Sign-in to the application to maintain a record of activities
- Search for movie tickets based on the search keyword
- Apply filters and sort results based on different genres
- Add all the selected movie tickets to a cart and customize the purchase at the end
- Experience a seamless payment process
- Receive a booking summary page once the payment is complete

Pushing the code to GitHub Repository:

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

cd MyMoviePlan MyMoviePlan

- 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/ MyMoviePlan .git](https://github.com/Mahendra1272/MyMoviePlan.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/MyMoviePlan.git>

Developer Details:

Mahendra Kumar Singh

mahendrakumarsingh9893@gmail.com

I. Sprint 1: Admin Login/Registration

1. Admin will only have login page, registration of admin will be hardcoded at the backend with the permission of production head. JWT authentication will be used for login verification.
2. Admin successful login will route to admin home page where all the desired admin functionalities will be added.
3. Admin home will have options to:
 - i. Add or remove any movie ticket from the application to build a rich product line.
 - ii. Edit movie ticket details like title, main actors, director's name, movie summary and movie poster to keep the product information updated with the current prices.
 - iii. View all the orders placed by all the users and their details, admin can manually change the status of any order available from placed to watched.
4. Admin can view all the details by applying filter as desired.

5. If Admin tries to visit any URL specifically designed for user, forbidden page will get loaded.

II. Sprint 2: User Login/Registration

1. User will have login and registration options, already registered users can login through their username and password and new users can register and create their username and password for further use. JWT authentication will be used for login verification.
2. Details entered by all users will be saved in the database, and password will be stored in encrypted format to maintain security.
3. After successful user login users will be able to:
 - i. View all the movie tickets available updated by admin on their home page.
 - ii. Click on view details tab to view more details of the movie ticket.
 - iii. Users can either click buy now or add to cart as desired both will lead them to payment details page.
 - iv. User can also view all the cart details by clicking on cart tab on navbar.
 - v. User can also view all the order details made by their ID by clicking on My-Orders tab on their navbar.
4. A logout functionality to logout current user.

III. Sprint 3: Searching/sorting/filtering options and dynamic data in user and admin portals.

1. User will be able to Search for products based on the search keywords & apply filters and sort results based on different categories.
2. Users should view the dynamic data of updated pricing on their site.
3. Admin will also be provided with searching/sorting function to apply CRUD operations over users list and available admin list.

IV. Sprint 4: Payment Gateway

1. User either can directly buy the selected movie ticket by clicking on buy now tab or can click on checkout tab in the cart.
2. After entering all payment details, a razorpay payment gateway will open up asking for payment details
3. Once clicked on pay now option user will be shown an order confirmation message.

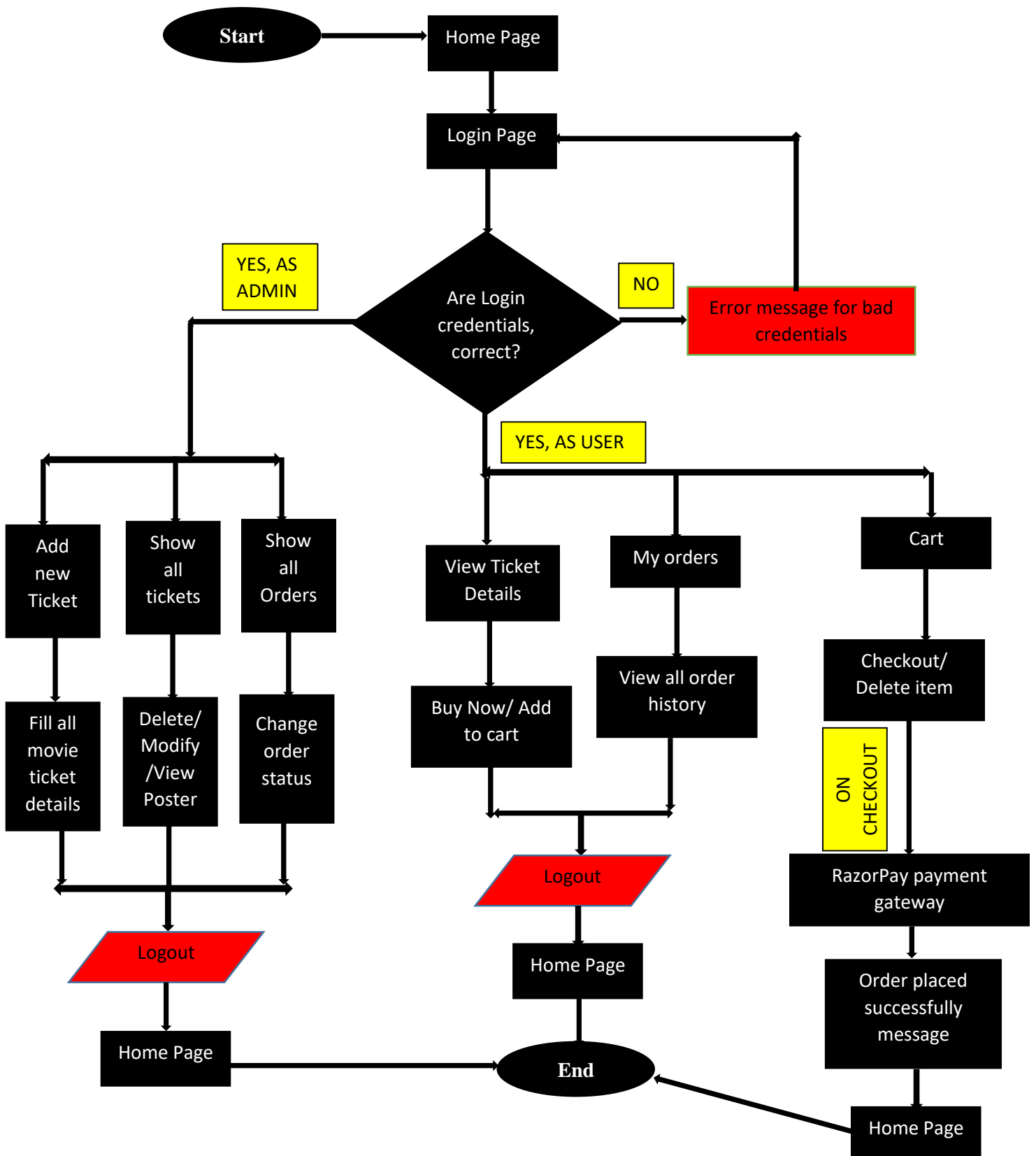
Functionalities

1. View trending Movies: The Home page of the application displays the popular movies.(Home Component)
2. Search : The user can search for movies by giving name as input.(search component)
3. View Details of a Particular movie : User can view the details of a particular movie(Movie Component)
4. Favourites: From the "Movie Component" Users can add and delete movies from their favourite list and view their favourite list in the "favoutrites Component"
5. Users can also add,edit and delete comments which they post for their favourite movies

Technologies used:

Database management: MySQL Backend logic: Java programming, SpringBoot Frontend development: Angular, Bootstrap, HTML/CSS, and Typescript

FlowChart



Source Code

Configuration

CorsConfiguration

```
package com.simplilearn.capstone2.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@Component
public class CorsConfiguration {

    private static final String GET = "GET";
    private static final String POST = "POST";
    private static final String DELETE = "DELETE";
    private static final String PUT = "PUT";

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
```

```

        @Override
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**")
                .allowedMethods(GET,PUT,POST,DELETE)
                .allowedHeaders("*")
                .allowedOriginPatterns("*")
                .allowCredentials(true);
        }
    };
}
}

```

JwtAuthenticationEntryPoint

```

package com.simplilearn.capstone2.configuration;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

```



```

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,
                     AuthenticationException authException) throws IOException, ServletException {
    //sending msg that access is unauthorized
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED,"Unauthorized");
}
}

```

JwtRequestFilter

```

package com.simplilearn.capstone2.configuration;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

```

```
import com.simplilearn.capstone2.service.JwtService;
```

```
import com.simplilearn.capstone2.util.JwtUtil;
```

```
import io.jsonwebtoken.ExpiredJwtException;
```

```
@Component
```

```
public class JwtRequestFilter extends OncePerRequestFilter {
```

```
    public static String CURRENT_USER = "";
```

```
    @Autowired
```

```
    private JwtUtil jwtutil;
```

```
    @Lazy
```

```
    @Autowired
```

```
    private JwtService jwtservice;
```

```
    @Override
```

```
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
```

```
                                   FilterChain filterChain)
```

```
        throws ServletException, IOException {
```

```
        final String header = request.getHeader("Authorization");
```

```
        String jwtToken = null;
```

```
        String userName = null;
```

```
        if (header != null && header.startsWith("Bearer ")) {
```

```
            jwtToken = header.substring(7);
```

```

try {

    userName = jwtutil.getUserNameFromToken(jwtToken);
    CURRENT_USER = userName;

} catch (IllegalArgumentException e) {
    System.out.println("Unable to get JWT token");
} catch (ExpiredJwtException e) {
    System.out.println("Jwt token is expired");
}

} else {
    System.out.println("Jwt token does not start with bearer");
}

if (userName != null && SecurityContextHolder.getContext().getAuthentication() == null)
{

    UserDetails userDetails = jwtservice.loadUserByUsername(userName);

    if (jwtutil.ValidateToken(jwtToken, userDetails)) {
        UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken =
                                new
UsernamePasswordAuthenticationToken(userDetails,
                                null, userDetails.getAuthorities());

        usernamePasswordAuthenticationToken.setDetails
        (new WebAuthenticationDetailsSource().buildDetails(request));
    }
}

```

```

        SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken)
    ;

    }

}

filterChain.doFilter(request, response);

}

}

```

WebSecurityConfiguration

```

package com.simplilearn.capstone2.configuration;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Lazy;
import org.springframework.http.HttpHeaders;
import org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

```

```
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.stereotype.Component;
```

```
import com.simplilearn.capstone2.service.JwtService;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

```
@Component
```

```
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {
```

```
    @Autowired
```

```
    private JwtAuthenticationEntryPoint jwtauthenticationentrypoint;
```

```
    //@Lazy
```

```
    @Autowired
```

```
    private JwtRequestFilter jwtrequestfilter;
```

```
    //@Lazy
```

```
    @Autowired
```

```
    private UserDetailsService jwtservice;
```

```
@Bean
```

```
@Override
```

```
public AuthenticationManager authenticationManagerBean() throws Exception {
```

```

        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpsecurity) throws Exception {
        httpsecurity.cors();

        httpsecurity.csrf().disable().authorizeRequests()

            .antMatchers("/authenticate", "/registerNewUser", "/movieticket/showall", "getTicketById/{ticket
            Id}", "/getTicketDetails/{isSingleTicketCheckout},{ticketId}")

            .permitAll().antMatchers(HttpHeaders.ALLOW)

                .permitAll().anyRequest().authenticated()

                .and()

                .exceptionHandling()

            .authenticationEntryPoint(jwtauthenticationentrypoint).and().sessionManagement()

                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        httpsecurity.addFilterBefore(jwtrequestfilter,
        UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder authenticatnManagerBuilder)
    throws Exception {

```

```

        authenticatnManagerBuilder.userDetailsService(jwtService).passwordEncoder(passwordEncoder
    ());
    }

}

```

Controller

CartController

```

package com.simplilearn.capstone2.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import com.simplilearn.capstone2.entity.Cart;
import com.simplilearn.capstone2.service.CartService;

@RestController

public class CartController {

    @Autowired
    private CartService cartService;

    @PreAuthorize("hasRole('User')")

```

```

    @GetMapping("/{addToCart/{ticketId}}")
    public Cart addToCart(@PathVariable(name="ticketId")Integer ticketId) {
        return cartService.addToCart(ticketId);
    }

    @PreAuthorize("hasRole('User')")
    @DeleteMapping("/{deleteCartItem/{cartId}}")
    public void deleteCartItem(@PathVariable (name="cartId")Integer cartId) {
        cartService.deleteCartItem(cartId);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping("/{getCartDetails}")
    public List<Cart> getCartDetails() {
        return cartService.getCartDetails();
    }
}

```

JwtController

```

package com.simplilearn.capstone2.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.simplilearn.capstone2.entity.JwtRequest;
import com.simplilearn.capstone2.entity.JwtResponse;

```



```

import com.simplilearn.capstone2.service.JwtService;

@RestController
@CrossOrigin

public class JwtController {

    @Autowired

    private JwtService jwtService;


    @PostMapping("/{authenticate}")

    public JwtResponse CreateJwtToken(@RequestBody JwtRequest jwtrequest)throws Exception {

        return jwtService.createJwtToken(jwtrequest);

    }

}

```

MovieTicketController

```

package com.simplilearn.capstone2.controller;

import java.io.IOException;

import java.util.HashSet;
import java.util.List;
import java.util.Set;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;

```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
```

```
import com.simplilearn.capstone2.dao.MovieTicketDao;
import com.simplilearn.capstone2.entity.MovieTickets;
import com.simplilearn.capstone2.entity.Posters;
import com.simplilearn.capstone2.service.MovieTicketService;
```

```
@RestController
```

```
@CrossOrigin
```

```
public class MovieTicketController {
```

```
    @Autowired
```

```
    private MovieTicketService mtService;
```

```
//    @Autowired
```

```
//    private MovieTicketDao mtDao;
```

```
    @PreAuthorize("hasRole('Admin')")
```

```
    @PostMapping(value = { "/movieticket/add" }, consumes = {
        MediaType.MULTIPART_FORM_DATA_VALUE })
```

```
    public MovieTickets addNewTicket(@RequestPart("movieticket") MovieTickets mt,
```

```
        @RequestPart("posters") MultipartFile[] file) {
```

```
        // return mtService.addNewTicket(mt);
```

```

        try {
            Set<Posters> posters = uploadPoster(file);
            mt.setMoviePoster(posters);
            return mtService.addNewTicket(mt);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return null;
        }
    }

    // for processing these uploaded images
    public Set<Posters> uploadPoster(MultipartFile[] multipartFiles) throws IOException {
        Set<Posters> posters = new HashSet<>();

        for (MultipartFile file : multipartFiles) {
            Posters p = new Posters(file.getOriginalFilename(), file.getContentType(),
file.getBytes());
            posters.add(p);
        }
        return posters;
    }

    // @PreAuthorize("hasRole('Admin')")
    @GetMapping({ "/movieticket/showall" })
    public List<MovieTickets> getAllTickets() {
        return mtService.getAllTickets();
    }

```

```

    @PreAuthorize("hasRole('Admin')")
    @DeleteMapping({ "/deleteticket/{ticketId}" })
    public void deleteTicket(@PathVariable("ticketId") Integer ticketId) {
        mtService.deleteTicket(ticketId);
    }

    @GetMapping("getTicketById/{ticketId}")
    public MovieTickets getTicketById(@PathVariable Integer ticketId) {
        return mtService.getTicketsById(ticketId);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping({ "/getTicketDetails/{isSingleTicketCheckout}/{ticketId}" })
    public List<MovieTickets> getTicketDetails(@PathVariable(name = "isSingleTicketCheckout")
boolean isSingleTicketCheckout,
        @PathVariable(name = "ticketId")Integer ticketId) {
        return mtService.getTicketDetails(isSingleTicketCheckout, ticketId);
    }
}

```

OrderDetailController

```

package com.simplilearn.capstone2.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;

```

```

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import com.simplilearn.capstone2.entity.OrderDetail;
import com.simplilearn.capstone2.entity.OrderInput;
import com.simplilearn.capstone2.entity.TransactionDetails;
import com.simplilearn.capstone2.service.OrderDetailService;

@RestController

public class OrderDetailController {

    @Autowired
    private OrderDetailService orderDetailService;

    @PreAuthorize("hasRole('User')")
    @PostMapping("/{place/order/{isSingleTicketCheckout}}")
    public void placeOrder(@PathVariable (name="isSingleTicketCheckout") boolean
isSingleTicketCheckout,
                           @RequestBody OrderInput orderInput) {
        orderDetailService.placeOrder(orderInput,isSingleTicketCheckout);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping("/{getOrderDetails}")
    public List<OrderDetail> getOrderDetails() {

```

```

        return orderDetailService.getOrderDetails();
    }

    @PreAuthorize("hasRole('Admin')")
    @GetMapping("/{getAllOrderDetails/{status}}")
    public List<OrderDetail> getAllOrderDetails(@PathVariable(name = "status")String status){
        return orderDetailService.getAllOrderDetails(status);
    }

    @PreAuthorize("hasRole('Admin')")
    @GetMapping("/{markOrderWatched/{orderId}}")
    public void markOrderWatched(@PathVariable(name = "orderId")Integer orderId ) {
        //this can be used for marked order as delivered in other cases.
        orderDetailService.markOrderAsWatched(orderId);
    }

    @PreAuthorize("hasRole('User')")
    @GetMapping("/{createTransaction/{amount}}")
    public TransactionDetails CreateTransaction(@PathVariable(name= "amount")Double amount) {
        return orderDetailService.createTransaction(amount);
    }
}

```

RoleController

```

package com.simplilearn.capstone2.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import com.simplilearn.capstone2.entity.Role;
```

```
import com.simplilearn.capstone2.service.RoleService;
```

```
@RestController
```

```
public class RoleController {
```

```
    @Autowired
```

```
    private RoleService roleservice;
```

```
    @PostMapping("/{createNewRole"})
```

```
    public Role createNewRole(@RequestBody Role role) {
```

```
        return roleservice.createNewRole(role);
```

```
    }
```

```
}
```

UserController

```
package com.simplilearn.capstone2.controller;
```

```
import javax.annotation.PostConstruct;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.security.access.prepost.PreAuthorize;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import com.simplilearn.capstone2.entity.User;
```

```
import com.simplilearn.capstone2.service.UserService;
```

```
@RestController
```

```
public class UserController {
```

```
    @Autowired
```

```
    private UserService userservice;
```

```
    @PostConstruct//will run each the program starts
```

```
    public void initRolesAndUsers() {
```

```
        userservice.initRolesAndUser();
```

```
    }
```

```
    @PostMapping("/{registerNewUser}")
```

```
    public User RegisterNewUser(@RequestBody User user) {
```

```
        return userservice.RegisterNewUser(user);
```

```
    }
```

```
    @GetMapping("/{forAdmin}")
```

```
    @PreAuthorize("hasRole('Admin')") //this will give access only to users having role of Admin
```

```
    public String forAdmin() {
```

```
        return "This URL is only accessible to admin";
```

```
    }
```

```
    @GetMapping("/{forUser}")
```

```
    @PreAuthorize("hasRole('User')")
```

```
    public String forUser() {
```

```
        return "This URL is only accessible to User";
```

```
    }
```



```
}
```

Dao

CartDao

```
package com.simplilearn.capstone2.dao;
```

```
import java.util.List;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.simplilearn.capstone2.entity.Cart;
```

```
import com.simplilearn.capstone2.entity.User;
```

```
@Repository
```

```
public interface CartDao extends CrudRepository<Cart, Integer>{
```

```
    public List<Cart> findByUser(User user);
```

```
}
```

MovieTicketDao

```
package com.simplilearn.capstone2.dao;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.simplilearn.capstone2.entity.MovieTickets;
```

```
@Repository
```

```
public interface MovieTicketDao extends CrudRepository<MovieTickets, Integer> {  
  
}
```

OrderDetailDao

```
package com.simplilearn.capstone2.dao;
```

```
import java.util.List;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.simplilearn.capstone2.entity.OrderDetail;
```

```
import com.simplilearn.capstone2.entity.User;
```

```
@Repository
```

```
public interface OrderDetailDao extends CrudRepository<OrderDetail, Integer> {
```

```
    public List<OrderDetail> findByUser(User user);
```

```
    //this will take object of user and will return the list of orderdetail
```

```
    //in this method findBy is mandatory followed by variable name in OrderDetail class i.e is user in  
this case.
```

```
    public List<OrderDetail> findByOrderStatus(String status);
```

```
}
```

RoleDao

```
package com.simplilearn.capstone2.dao;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.simplilearn.capstone2.entity.Role;
```

```
@Repository
```

```
public interface RoleDao extends CrudRepository<Role, String>{
```

```
}
```

UserDao

```
package com.simplilearn.capstone2.dao;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.simplilearn.capstone2.entity.User;
```

```
@Repository
```

```
public interface UserDao extends CrudRepository<User, String> {
```

```
}
```

Entity

Cart

```
package com.simplilearn.capstone2.entity;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.OneToOne;
```

```
import javax.persistence.Table;
```

@Entity

@Table

public class Cart {

 @Id

 @GeneratedValue(strategy = GenerationType.AUTO)

 private Integer cartId;

 @OneToOne

 private MovieTickets movieTicket;

 @OneToOne

 private User user;

 public Cart() {

 }

 public Cart(MovieTickets movieTicket, User user) {

 super();

 this.movieTicket = movieTicket;

 this.user = user;

 }

 public Integer getCartId() {

 return cartId;

 }

 public void setCartId(Integer cartId) {

 this.cartId = cartId;

```

    }

    public MovieTickets getMovieTicket() {

        return movieTicket;

    }

    public void setMovieTicket(MovieTickets movieTicket) {

        this.movieTicket = movieTicket;

    }

    public User getUser() {

        return user;

    }

    public void setUser(User user) {

        this.user = user;

    }

}

```

JwtRequest

```

package com.simplilearn.capstone2.entity;

public class JwtRequest {

    private String userName;

    private String userPassword;

    public String getUserName() {

        return userName;

    }

    public void setUserName(String userName) {

        this.userName = userName;

    }

    public String getUserPassword() {

```

```

        return userPassword;
    }

    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
}

```

JwtResponse

```

package com.simplilearn.capstone2.entity;

public class JwtResponse {

    private User user;
    private String jwtToken;

    public JwtResponse(User user, String jwtToken) {
        super();
        this.user = user;
        this.jwtToken = jwtToken;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getJwtToken() {
        return jwtToken;
    }
}

```

```

    }

    public void setJwtToken(String jwtToken) {
        this.jwtToken = jwtToken;
    }

}

```

MovieTickets

```

package com.simplilearn.capstone2.entity;

import java.util.Set;

import javax.persistence.CascadeType;
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;

@Entity
public class MovieTickets {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ticketId;
    private String movieName;

```

```
private String genre;

private String actors;

private String director;

private String language;

private String description;

private double actualPrice;

private double discountedPrice;

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

@JoinTable(name = "ticket_posters", joinColumns = { @JoinColumn(name = "ticket_id") },
inverseJoinColumns = {

    @JoinColumn(name = "poster_id") })

private Set<Posters> moviePoster;
```

```
public Set<Posters> getMoviePoster() {

    return moviePoster;

}
```

```
public void setMoviePoster(Set<Posters> moviePoster) {

    this.moviePoster = moviePoster;

}
```

```
public String getLanguage() {

    return language;

}
```

```
public void setLanguage(String language) {

    this.language = language;
```



```
}
```

```
public String getDirector() {  
    return director;  
}
```

```
public void setDirector(String director) {  
    this.director = director;  
}
```

```
public int getTicketId() {  
    return ticketId;  
}
```

```
public void setTicketId(int ticketId) {  
    this.ticketId = ticketId;  
}
```

```
public String getMovieName() {  
    return movieName;  
}
```

```
public void setMovieName(String movieName) {  
    this.movieName = movieName;  
}
```

```
public String getGenre() {  
    return genre;  
}
```

```
public void setGenre(String genre) {  
    this.genre = genre;  
}
```

```
public String getActors() {  
    return actors;  
}
```

```
public void setActors(String actors) {  
    this.actors = actors;  
}
```

```
public String getDescription() {  
    return description;  
}
```

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

```
public double getActualPrice() {  
    return actualPrice;  
}
```

```
public void setActualPrice(double actualPrice) {  
    this.actualPrice = actualPrice;  
}
```

```

    public double getDiscountedPrice() {
        return discountedPrice;
    }

    public void setDiscountedPrice(double discountedPrice) {
        this.discountedPrice = discountedPrice;
    }

    public MovieTickets(int ticketId, String movieName, String genre, String actors, String director,
        String description, String language, double actualPrice, double discountedPrice)
    {
        super();
        this.ticketId = ticketId;
        this.movieName = movieName;
        this.genre = genre;
        this.actors = actors;
        this.director = director;
        this.language = language;
        this.description = description;
        this.actualPrice = actualPrice;
        this.discountedPrice = discountedPrice;
    }

    public MovieTickets() {
        super();
        // TODO Auto-generated constructor stub
    }
}

```

MovieTickets

```
package com.simplilearn.capstone2.entity;
```

```
import java.util.Set;
```

```
import javax.persistence.CascadeType;
```

```
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;
```

```
@Entity
```

```
public class MovieTickets {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private int ticketId;
```

```
    private String movieName;
```

```
    private String genre;
```

```
    private String actors;
```

```
    private String director;
```

```
    private String language;
```

```
    private String description;
```

```
    private double actualPrice;
```

```
    private double discountedPrice;
```

```
@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)

@JoinTable(name = "ticket_posters", joinColumns = { @JoinColumn(name = "ticket_id") },
inverseJoinColumns = {

    @JoinColumn(name = "poster_id") })

private Set<Posters> moviePoster;


public Set<Posters> getMoviePoster() {

    return moviePoster;

}


public void setMoviePoster(Set<Posters> moviePoster) {

    this.moviePoster = moviePoster;

}


public String getLanguage() {

    return language;

}


public void setLanguage(String language) {

    this.language = language;

}


public String getDirector() {

    return director;

}


public void setDirector(String director) {
```

```
        this.director = director;
    }
}
```

```
public int getTicketId() {
    return ticketId;
}
```

```
public void setTicketId(int ticketId) {
    this.ticketId = ticketId;
}
```

```
public String getMovieName() {
    return movieName;
}
```

```
public void setMovieName(String movieName) {
    this.movieName = movieName;
}
```

```
public String getGenre() {
    return genre;
}
```

```
public void setGenre(String genre) {
    this.genre = genre;
}
```

```
public String getActors() {
    return actors;
}
```

```
}
```

```
public void setActors(String actors) {
```

```
    this.actors = actors;
```

```
}
```

```
public String getDescription() {
```

```
    return description;
```

```
}
```

```
public void setDescription(String description) {
```

```
    this.description = description;
```

```
}
```

```
public double getActualPrice() {
```

```
    return actualPrice;
```

```
}
```

```
public void setActualPrice(double actualPrice) {
```

```
    this.actualPrice = actualPrice;
```

```
}
```

```
public double getDiscountedPrice() {
```

```
    return discountedPrice;
```

```
}
```

```
public void setDiscountedPrice(double discountedPrice) {
```

```
    this.discountedPrice = discountedPrice;
```

```
}
```

```

    public MovieTickets(int ticketId, String movieName, String genre, String actors, String director,
                        String description, String language, double actualPrice, double discountedPrice)
    {
        super();
        this.ticketId = ticketId;
        this.movieName = movieName;
        this.genre = genre;
        this.actors = actors;
        this.director = director;
        this.language = language;
        this.description = description;
        this.actualPrice = actualPrice;
        this.discountedPrice = discountedPrice;
    }

    public MovieTickets() {
        super();
        // TODO Auto-generated constructor stub
    }
}

```

OrderDetail

```

package com.simplilearn.capstone2.entity;

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

```



```
import javax.persistence.OneToOne;
```

```
@Entity
```

```
public class OrderDetail {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Integer orderId;
```

```
    private String orderFullname;
```

```
    private String orderFullAddress;
```

```
    private String orderContactNumber;
```

```
    private String orderAlternateNumber;
```

```
    private String orderStatus;
```

```
    private double orderAmount;
```

```
    @OneToOne
```

```
    private MovieTickets movieTicket;
```

```
    @OneToOne
```

```
    private User user;
```

```
    private String transactionId;
```

```
    public String getTransactionId() {
```

```
        return transactionId;
```

```
    }
```

```
    public void setTransactionId(String transactionId) {
```

```
        this.transactionId = transactionId;
```

```
    }
```

```
    public OrderDetail() {
```

```
}

public MovieTickets getMovieTicket() {
    return movieTicket;
}

public void setMovieTicket(MovieTickets movieTicket) {
    this.movieTicket = movieTicket;
}

public User getUser() {
    return user;
}

public void setUser(User user) {
    this.user = user;
}

public Integer getOrderId() {
    return orderId;
}

public void setOrderId(Integer orderId) {
    this.orderId = orderId;
}

public String getOrderFullname() {
    return orderFullname;
}

public void setOrderFullname(String orderFullname) {
    this.orderFullname = orderFullname;
}

public String getOrderFullAddress() {
    return orderFullAddress;
}
}
```

```
public void setOrderFullAddress(String orderFullAddress) {  
    this.orderFullAddress = orderFullAddress;  
}  
public String getOrderContactNumber() {  
    return orderContactNumber;  
}  
public void setOrderContactNumber(String orderContactNumber) {  
    this.orderContactNumber = orderContactNumber;  
}  
public String getOrderAlternateNumber() {  
    return orderAlternateNumber;  
}  
public void setOrderAlternateNumber(String orderAlternateNumber) {  
    this.orderAlternateNumber = orderAlternateNumber;  
}  
public String getOrderStatus() {  
    return orderStatus;  
}  
public void setOrderStatus(String orderStatus) {  
    this.orderStatus = orderStatus;  
}  
public double getOrderAmount() {  
    return orderAmount;  
}  
public void setOrderAmount(double orderAmount) {  
    this.orderAmount = orderAmount;  
}  
public OrderDetail( String orderFullname, String orderFullAddress, String orderContactNumber,
```

```
String orderAlternateNumber, String orderStatus, double orderAmount,  
MovieTickets movieTicket, User user,
```

```
String transactionId) {  
    super();  
  
    this.orderFullName = orderFullName;  
    this.orderFullAddress = orderFullAddress;  
    this.orderContactNumber = orderContactNumber;  
    this.orderAlternateNumber = orderAlternateNumber;  
    this.orderStatus = orderStatus;  
    this.orderAmount = orderAmount;  
    this.movieTicket = movieTicket;  
    this.user = user;  
    this.transactionId = transactionId;  
}
```

```
}
```

OrderInput

```
package com.simplilearn.capstone2.entity;
```

```
import java.util.List;
```

```
public class OrderInput {
```

```
    private String fullName;
```

```
private String fullAddress;  
private String contactNumber;  
private String alternateContactNumber;  
private List<OrderTicketQuantity> orderTicketQuantity;  
private String transactionId;
```

```
public String getTransactionId() {  
    return transactionId;  
}  
public void setTransactionId(String transactionId) {  
    this.transactionId = transactionId;  
}  
public String getFullName() {  
    return fullName;  
}  
public void setFullName(String fullName) {  
    this.fullName = fullName;  
}  
public String getFullAddress() {  
    return fullAddress;  
}  
public void setFullAddress(String fullAddress) {  
    this.fullAddress = fullAddress;  
}  
public String getContactNumber() {  
    return contactNumber;  
}  
public void setContactNumber(String contactNumber) {
```

```

        this.contactNumber = contactNumber;
    }

    public String getAlternateContactNumber() {
        return alternateContactNumber;
    }

    public void setAlternateContactNumber(String alternateContactNumber) {
        this.alternateContactNumber = alternateContactNumber;
    }

    public List<OrderTicketQuantity> getOrderTicketQuantity() {
        return orderTicketQuantity;
    }

    public void setOrderTicketQuantity(List<OrderTicketQuantity> orderTicketQuantity) {
        this.orderTicketQuantity = orderTicketQuantity;
    }
}

```

OrderTicketQuantity

```

package com.simplilearn.capstone2.entity;

```

```

public class OrderTicketQuantity {

    private Integer ticketId;
    private Integer quantity;

    public Integer getTicketId() {
        return ticketId;
    }

    public void setTicketId(Integer ticketId) {
        this.ticketId = ticketId;
    }
}

```

```

    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}

```

Posters

```

package com.simplilearn.capstone2.entity;

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

@Entity
@Table(name="movie_poster")
public class Posters {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long posterId;

    private String name;

    private String type;

    @Column(length = 5000000)

```

```
private byte[] pic;

public Posters(String name, String type, byte[] pic) {
    super();
    this.name = name;
    this.type = type;
    this.pic = pic;
}

public Posters() {
    super();
    // TODO Auto-generated constructor stub
}
```

```
public long getPosterId() {
    return posterId;
}

public void setPosterId(long posterId) {
    this.posterId = posterId;
}

public String getName() {
    return name;
}

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

public String getType() {
    return type;
}

public void setType(String type) {
```



```

        this.type = type;
    }

    public byte[] getPic() {
        return pic;
    }

    public void setPic(byte[] pic) {
        this.pic = pic;
    }
}

```

Posters

```

package com.simplilearn.capstone2.entity;

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

@Entity
@Table(name="movie_poster")
public class Posters {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long posterId;

    private String name;
}

```

```
private String type;
@Column(length = 5000000)
private byte[] pic;
public Posters(String name, String type, byte[] pic) {
    super();
    this.name = name;
    this.type = type;
    this.pic = pic;
}
public Posters() {
    super();
    // TODO Auto-generated constructor stub
}
```

```
public long getPosterId() {
    return posterId;
}
public void setPosterId(long posterId) {
    this.posterId = posterId;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getType() {
    return type;
}
```

```

    }

    public void setType(String type) {
        this.type = type;
    }

    public byte[] getPic() {
        return pic;
    }

    public void setPic(byte[] pic) {
        this.pic = pic;
    }
}

```

Role

```
package com.simplilearn.capstone2.entity;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Role {
```

```
    @Id
```

```
    private String roleName;
```

```
    private String roleDescription;
```

```
    public String getRoleName() {
```

```
        return roleName;
```

```
    }
```

```
public void setRoleName(String roleName) {  
    this.roleName = roleName;  
}  
public String getRoleDescription() {  
    return roleDescription;  
}  
public void setRoleDescription(String roleDescription) {  
    this.roleDescription = roleDescription;  
}  
  
}
```

TransactionDetails

```
package com.simplilearn.capstone2.entity;
```

```
public class TransactionDetails {  
  
    private String orderId;  
    private String currency;  
    private Integer amount;  
    private String key;  
  
    public TransactionDetails(String orderId, String currency, Integer amount, String key) {  
        super();  
        this.orderId = orderId;  
        this.currency = currency;  
        this.amount = amount;  
        this.key = key;  
    }  
}
```

```
}
```

```
public String getKey() {  
    return key;  
}
```

```
public void setKey(String key) {  
    this.key = key;  
}
```

```
public String getOrderId() {  
    return orderId;  
}
```

```
public void setOrderId(String orderId) {  
    this.orderId = orderId;  
}
```

```
public String getCurrency() {  
    return currency;  
}
```

```
public void setCurrency(String currency) {  
    this.currency = currency;  
}
```

```
public Integer getAmount() {  
    return amount;  
}
```

```
public void setAmount(Integer amount) {
```

```
        this.amount = amount;
    }

}
```

User

```
package com.simplilearn.capstone2.entity;
```

```
import java.util.Set;
```

```
import javax.persistence.CascadeType;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.FetchType;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
```

```
import javax.persistence.JoinTable;
```

```
import javax.persistence.ManyToMany;
```

```
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name="users")
```

```
public class User {
```

```
    @Id
```

```
    private String userName;
```

```
    private String fullName;
```

```

private String userpassword;

// many uses may have many different roles
/*
 * By itself create a third table named USER_ROLE that will have user and its
 * associated role details this third table will have user_id and associated
 * role_id
 */
@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(name = "USER_ROLE", joinColumns = { @JoinColumn(name = "USER_ID") },
inverseJoinColumns = {
    @JoinColumn(name = "ROLE_ID") })
private Set<Role> role;

public String getUsername() {
    return userName;
}

public void setUsername(String userName) {
    this.userName = userName;
}

public String getFullName() {
    return fullName;
}

public void setFullName(String fullName) {
    this.fullName = fullName;
}

```

```
public String getUserpassword() {  
    return userpassword;  
}
```

```
public void setUserpassword(String userpassword) {  
    this.userpassword = userpassword;  
}
```

```
public Set<Role> getRole() {  
    return role;  
}
```

```
public void setRole(Set<Role> role) {  
    this.role = role;  
}
```

```
}
```


Output

MyMoviePlan x avatar 2 imdb - Google Search x +

localhost:4200/addNewTicket

Gmail study Aptitude Test for So... Top 50+ Data Struc... RxJS in Angular: Cre... Asymptotic Notatio...

MyMoviePlan Home Admin Logout

Home Add new Ticket Show all Tickets Show all Order

Movie Title

Avatar: The Way of Water

Choose Files avatar-poster-1.jpg

Genre

Action-Adventure-Fantasy

Main Actors

Sam Worthington, Zoe Saldana, Sigourney Weaver

Director

James Cameron

Movie summary

Jake Sully lives with his newfound family formed on the extrasolar moon Panc

Language

English

Main Actors

Sam Worthington, Zoe Saldana, Sigourney Weaver

Director

James Cameron

Movie summary

Jake Sully lives with his newfound family formed on the extrasolar moon Panc

Language

English

Actual price

620

Discounted Price

540

Add Ticket Reset

MyMoviePlan

HomeAdmin

Logout

Home

Add new Ticket

Show all Tickets

Show all Order

Movie Title

Genre

Main Actors

Director

Movie summary

Language

Choose Files

No file chosen

MyMoviePlan

localhost:4200

Bautista, Jonathan Groff,
Ben Aldridge

₹480.00 ₹520.00

View Details

Shetty, Kishore Kumar,
G.Achyuth Kumar

₹400.00 ₹450.00

View Details

Sethupathi, Varalaxmi
Sarathkumar, Divyansha
Kaushik


₹400.00 ₹450.00

View Details

Keerthy Suresh, Shine
Tom Chacko

₹350.00 ₹480.00

View Details




Pathan

Main Actors: Shahrukh
Khan, Deepika Padukone,
John Abraham

₹395.00 ₹480.00

View Details




Tu Jhoothi Main
Makkaar

Main Actors: Ranbir
Kapoor, Shraddha Kapoor,
Hasleen Kaur

₹380.00 ₹420.00

View Details




Shehzada

Main Actors: Kartik
Aaryan, Kriti Sanon,
Paresh Rawal

₹410.00 ₹480.00

View Details



Avatar: The Way of
Water

Main Actors: Sam
Worthington, Zoe Saldana,
Sigourney Weaver

₹540.00 ₹620.00

View Details

MyMoviePlan

HomeAdmin

Logout

Home

Add new Ticket

Show all Tickets

Show all Order

Order Details:

All	Placed	Watched					
Order ID	Movie Title	User Full Name	Contact Number	Alternate Contact Number	Order Amount	Order Status	Mark as watched
20	Michael	Raj Malhotra	8987989700	8899007766	900	Watched	
21	Michael	Raj Malhotra	8877990000	9900887766	1350	Watched	
22	Kantara	Raj Malhotra	8800998877	8800994455	450	Watched	
23	Shehzada	Raj Malhotra	7766558899	8800994455	410	Watched	
24	Shehzada	Raj Malhotra	7766558800	8800994112	1640	Placed	<input checked="" type="checkbox"/>
25	Tu Jhoothi Main Makkaar	Raj Malhotra	7766558665	7766550000	760	Placed	<input checked="" type="checkbox"/>
26	Tu Jhoothi Main Makkaar	Raj Malhotra	6789760000	8800994455	380	Placed	<input checked="" type="checkbox"/>

MyMoviePlan

HomeAdmin

Logout

Home

Add new Ticket

Show all Tickets

Show all Order

Order Details:

All	Placed	Watched					
Order ID	Movie Title	User Full Name	Contact Number	Alternate Contact Number	Order Amount	Order Status	Mark as watched
24	Shehzada	Raj Malhotra	7766558800	8800994112	1640	Placed	
25	Tu Jhoothi Main Makkaar	Raj Malhotra	7766558665	7766550000	760	Placed	
26	Tu Jhoothi Main Makkaar	Raj Malhotra	6789760000	8800994455	380	Placed	
27	Pathan	Harshita Sahu	997777009	667777889	1185	Placed	
32	Knock at the Cabin	Harshita Sahu	7766889900	6655447788	480	Placed	
33	Tu Jhoothi Main Makkaar	Harshita Sahu	7766889900	6655447788	380	Placed	
34	Michael	Harshita Sahu	7766889900	6655447788	1200	Placed	
35	Knock at the Cabin	Harshita Sahu	7766558899	8800994455	480	Placed	

MyMoviePlan

HomeAdmin

Logout

Home

Add new Ticket

Show all Tickets

Show all Order

Order Details:

AllPlacedWatched

Order ID	Movie Title	User Full Name	Contact Number	Alternate Contact Number	Order Amount	Order Status	Mark as watched
20	Michael	Raj Malhotra	8987989700	8899007766	900	Watched	
21	Michael	Raj Malhotra	8877990000	9900887766	1350	Watched	
22	Kantara	Raj Malhotra	8800998877	8800994455	450	Watched	
23	Shehzada	Raj Malhotra	7766558899	8800994455	410	Watched	
49	Kantara	Sandeep Malhotra	7766558899	8800994455	1600	Watched	
50	Tu Jhoothi Main Makkaar	Sandeep Malhotra	6699880077	9988665500	1520	Watched	

Order Details:

AllPlacedWatched

Order ID	Movie Title	User Full Name	Contact Number	Alternate Contact Number	Order Amount	Order Status	Mark as watched
20	Michael	Raj Malhotra	8987989700	8899007766	900	Watched	
21	Michael	Raj Malhotra	8877990000	9900887766	1350	Watched	
22	Kantara	Raj Malhotra	8800998877	8800994455	450	Watched	
23	Shehzada	Raj Malhotra	7766558899	8800994455	410	Watched	
24	Shehzada	Raj Malhotra	7766558800	8800994112	1640	Watched	
25	Tu Jhoothi Main Makkaar	Raj Malhotra	7766558665	7766550000	760	Watched	
26	Tu Jhoothi Main Makkaar	Raj Malhotra	6789760000	8800994455	380	Placed	<div></div>
27	Pathan	Harshita Sahu	997777009	667777889	1185	Placed	<div></div>
32	Knock at the Cabin	Harshita Sahu	7766889900	6655447788	480	Placed	<div></div>
33	Tu Jhoothi Main Makkaar	Harshita Sahu	7766889900	6655447788	380	Placed	<div></div>
34	Michael	Harshita Sahu	7766889900	6655447788	1200	Placed	<div></div>

MyMoviePlan

localhost:4200/showAllTickets

Gmail

study

Aptitude Test for So...

Top 50+ Data Struc...



RxJS in Angular: Cre...

Asymptotic Notatio...

Paused

Tickets Details:

ID	Movie Title	Genre	Main Actors	Actual Price	Discounted Price	Poster	Delete	Edit
5	Knock at the Cabin							
6	Kantara							
7	Michael							
9	Dasara							
14	Pathan							
16	Tu Jhoothi Main Makkaar							
18	Shehzada							
62	Avatar: The Way of Water							

MyMoviePlan

Home Admin

Logout

Home

Add new Ticket

Show all Tickets

Show all Order

Tickets Details:

ID	Movie Title	Genre	Main Actors	Actual Price	Discounted Price	Poster	Delete	Edit
5	Knock at the Cabin	Horror-Mystery-Thriller	Dave Bautista, Jonathan Groff, Ben Aldridge	520	480			
6	Kantara	Action-Adventure-Thriller-Drama	Rishab Shetty, Kishore Kumar, G.Achyuth Kumar	450	400			
7	Michael	Action-Crime-Drama	Vijay Sethupathi, Varalaxmi Sarathkumar, Divyansha Kaushik	450	400			
9	Dasara	Action-Adventure-Drama	Nani, Keerthy Suresh, Shine Tom Chacko	480	350			
14	Pathan	Action_Adventure-Drama	Shahrukh Khan, Deepika Padukone, John Abraham	480	395			
16	Tu Jhoothi Main Makkaar	comedy_Romance	Ranbir Kapoor, Shraddha Kapoor, Hasleen Kaur	420	380			
18	Shehzada	Action-Comedy-Drama	Kartik Aaryan, Kriti Sanon, Paresh Rawal	480	410			

MyMoviePlan

localhost:4200/showAllTickets

Gmail

study

























Aptitude Test for So...

Top 50+ Data Struc...

RxJS in Angular: Cre...

Asymptotic Notatio...

Tickets Details:

ID	Movie Title	Genre	Main Actors	Actual Price	Discounted Price	Poster	Delete	Edit
5	Knock at the Cabin	Horror-Mystery-Thriller	Dave Bautista, Jonathan Groff, Ben Aldridge	520	480			
6	Kantara	Action-Adventure-Thriller-Drama	Rishab Shetty, Kishore Kumar, G.Achyuth Kumar	450	400			
7	Michael	Action-Crime-Drama	Vijay Sethupathi, Varalaxmi Sarathkumar, Divyansha Kaushik	450	400			
9	Dasara	Action-Adventure-Drama	Nani, Keerthy Suresh, Shine Tom Chacko	480	350			
14	Pathan	Action_Adventure-Drama	Shahrukh Khan, Deepika Padukone, John Abraham	480	395			
16	Tu Jhoothi Main Makkaar	comedy_Romance	Ranbir Kapoor, Shraddha Kapoor, Hasleen Kaur	420	380			
18	Shehzada	Action-Comedy-Drama	Kartik Aaryan, Kriti Sanon, Paresh Rawal	480	410			
62	Avatar: The Way of Water	Action-Adventure-Fantasy	Sam Worthington, Zoe Saldana, Sigourney Weaver	620	540			

MyMoviePlan

localhost:4200/addNewTicket;ticketId=18

Gmail

study

Aptitude Test for So...

Top 50+ Data Struc...

RxJS in Angular: Cre...

Asymptotic Notatio...

Movie Title

Shehzada

Genre

Action-Comedy-Drama

Main Actors

Kartik Aaryan, Kriti Sanon, Paresh Rawal

Director

Rohit Dhawan

Movie summary

Bantu is hated by his father Valmiki since he was a toddler. Samara, his boss,

Language

Hindi


Actual price

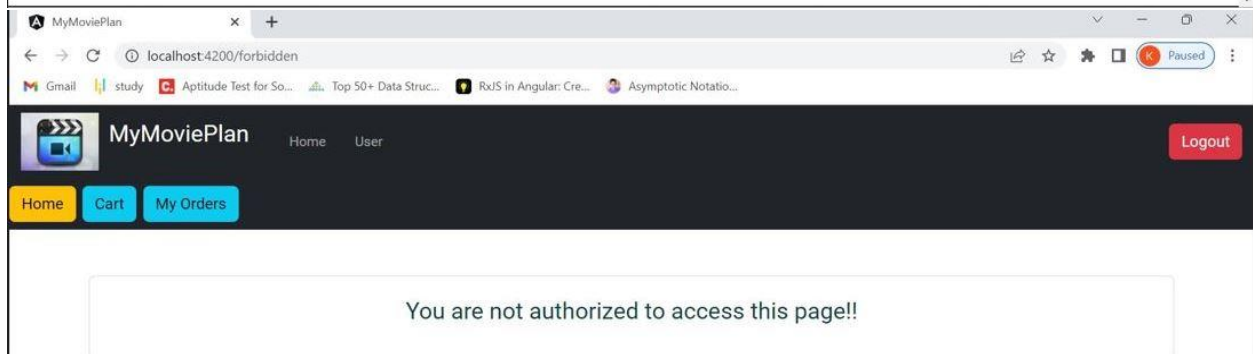
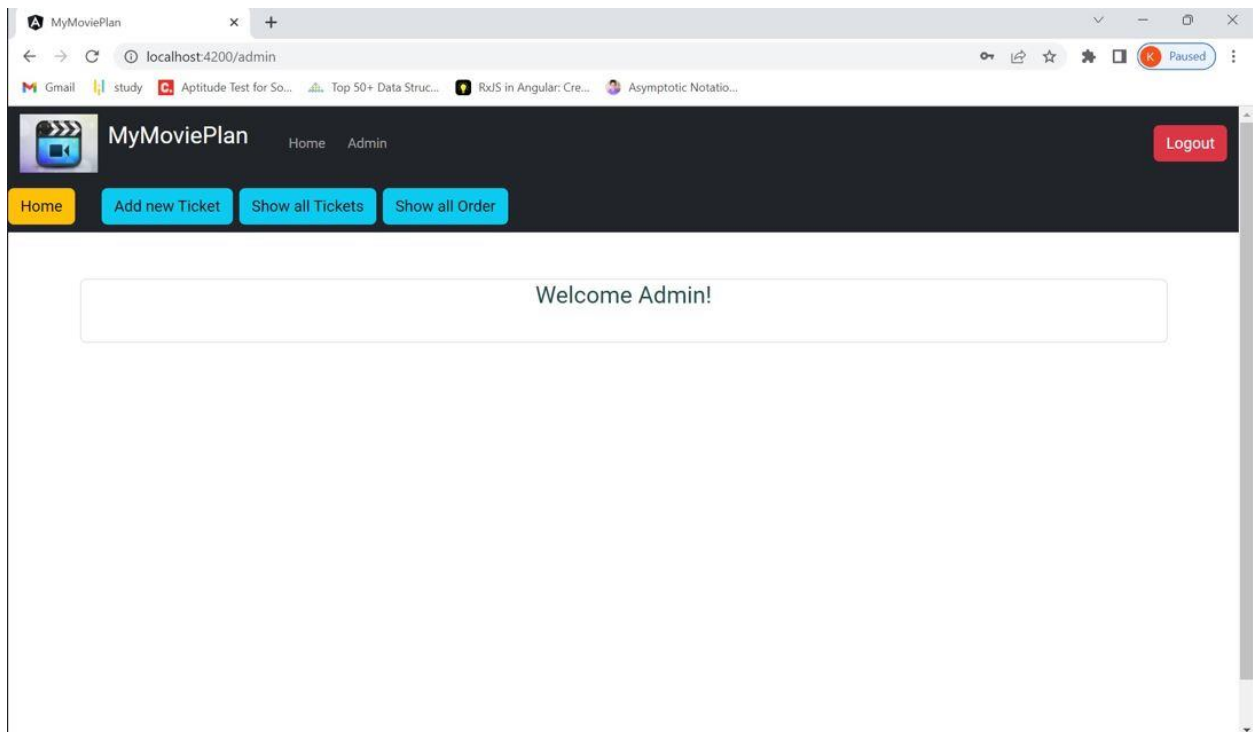
480

Discounted Price

Choose Files

No file chosen






MyMoviePlan

Home Admin Logout

Home Add new Ticket Show all Tickets Show all Order

Book Your Movie Tickets here!!




Knock at the Cabin

Main Actors: Dave Bautista, Jonathan Groff, Ben Aldridge

₹480.00 ₹520.00

View Details




Kantara

Main Actors: Rishab Shetty, Kishore Kumar, G.Achyuth Kumar

₹400.00 ₹450.00

View Details




Michael

Main Actors: Vijay Sethupathi, Varalaxmi Sarathkumar, Divyansha Kaushik

₹400.00 ₹450.00

View Details



Dasara

Main Actors: Nani, Keerthy Suresh, Shine Tom Chacko


₹350.00 ₹480.00

View Details

MyMoviePlan

Home Login

Book Your Movie Tickets here!!




Knock at the Cabin

Main Actors: Dave Bautista, Jonathan Groff, Ben Aldridge

₹480.00 ₹520.00

View Details




Kantara

Main Actors: Rishab Shetty, Kishore Kumar, G.Achyuth Kumar

₹400.00 ₹450.00

View Details




Michael

Main Actors: Vijay Sethupathi, Varalaxmi Sarathkumar, Divyansha Kaushik

₹400.00 ₹450.00

View Details

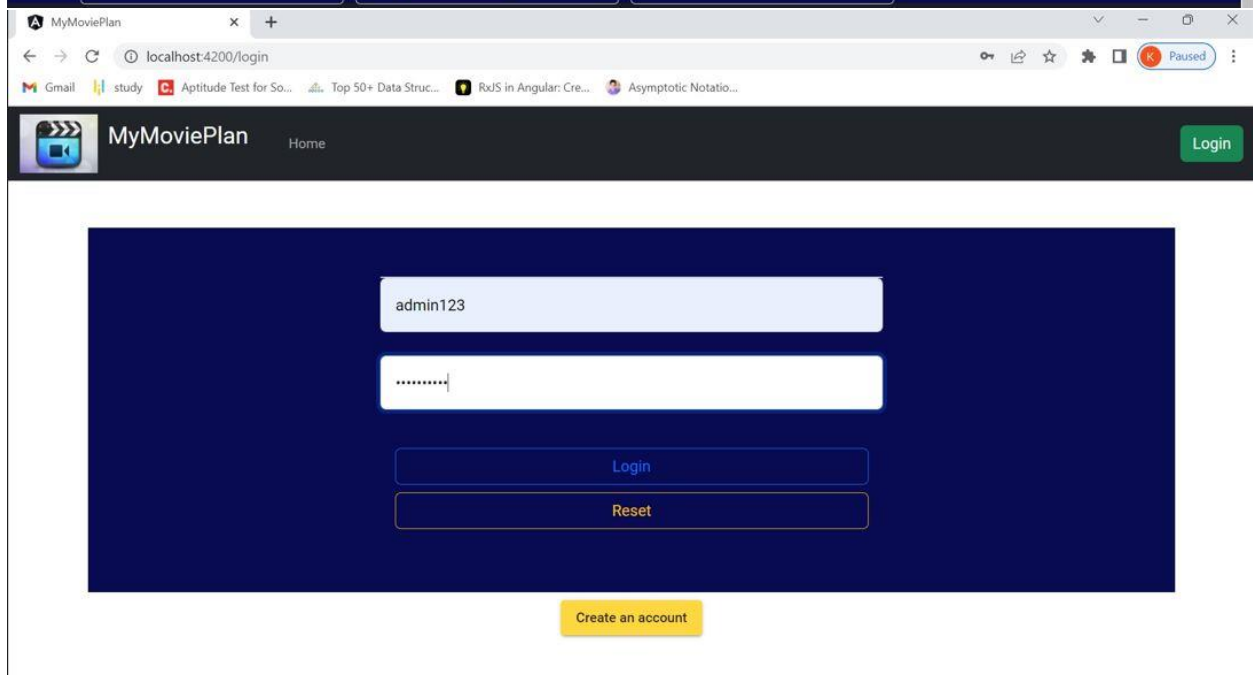
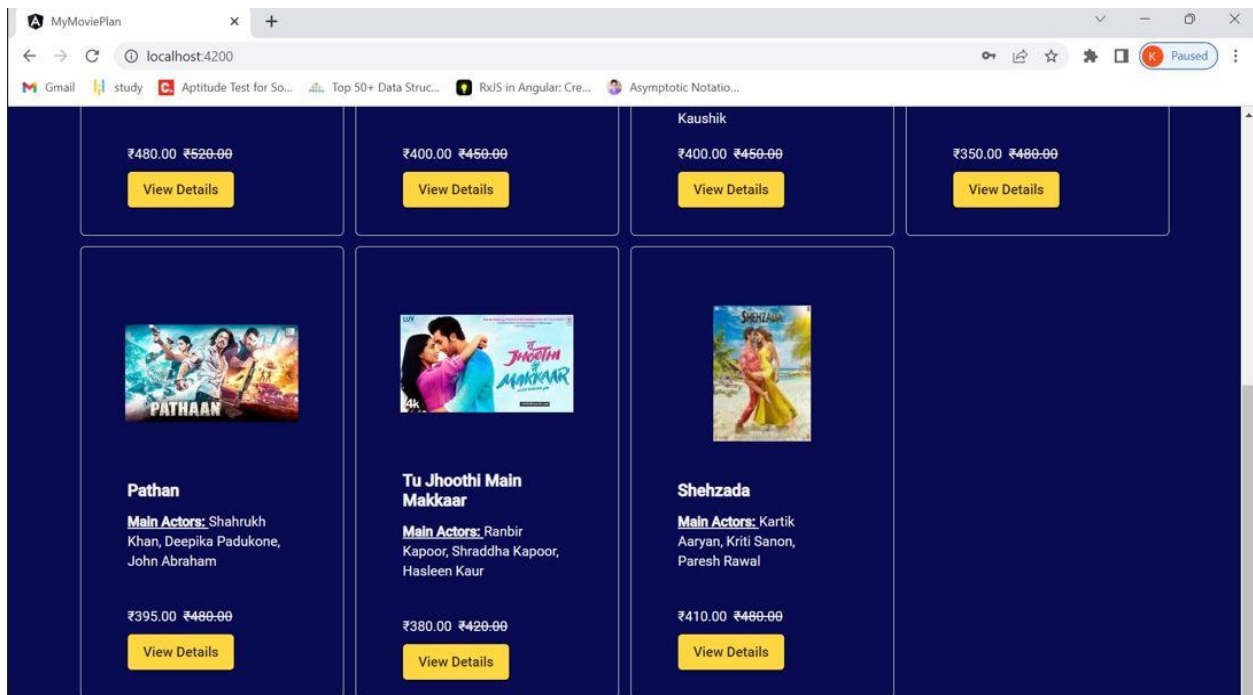


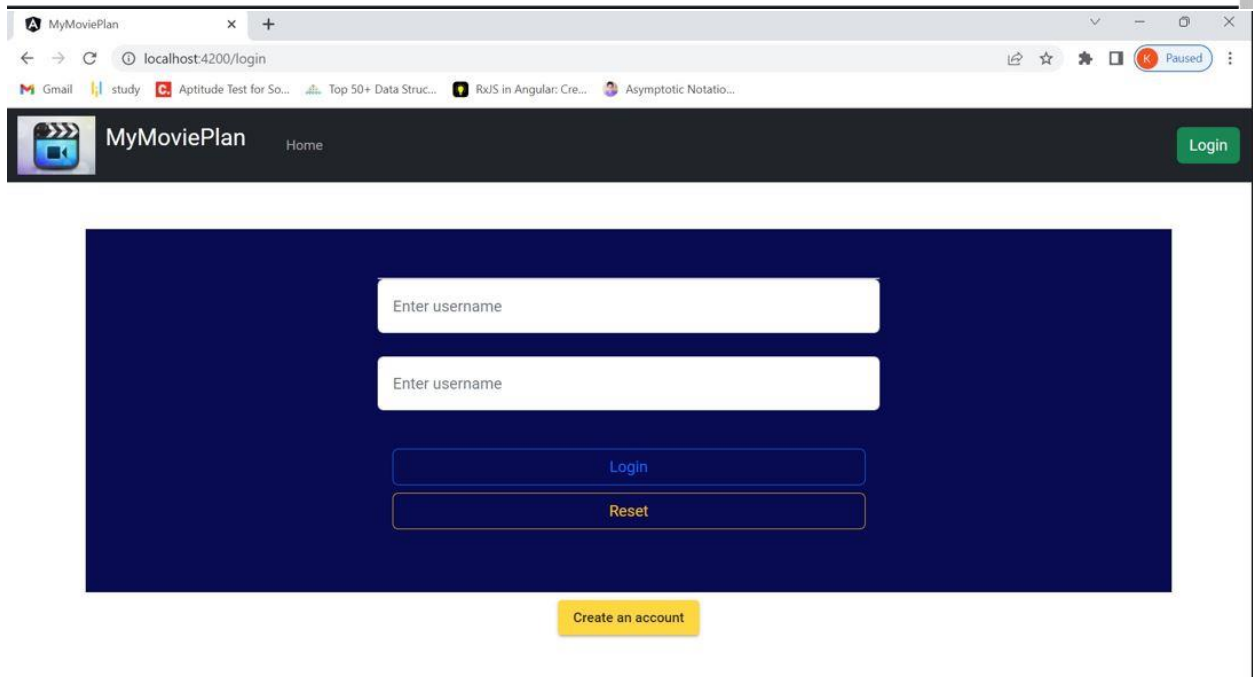
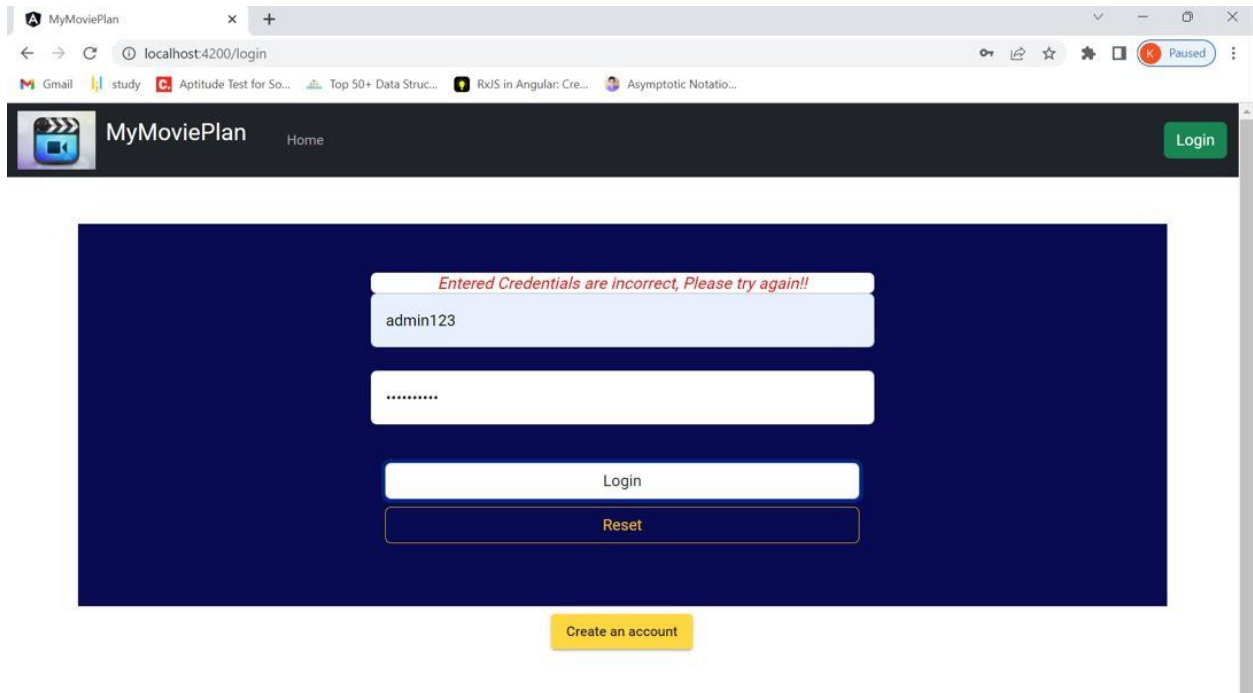
Dasara

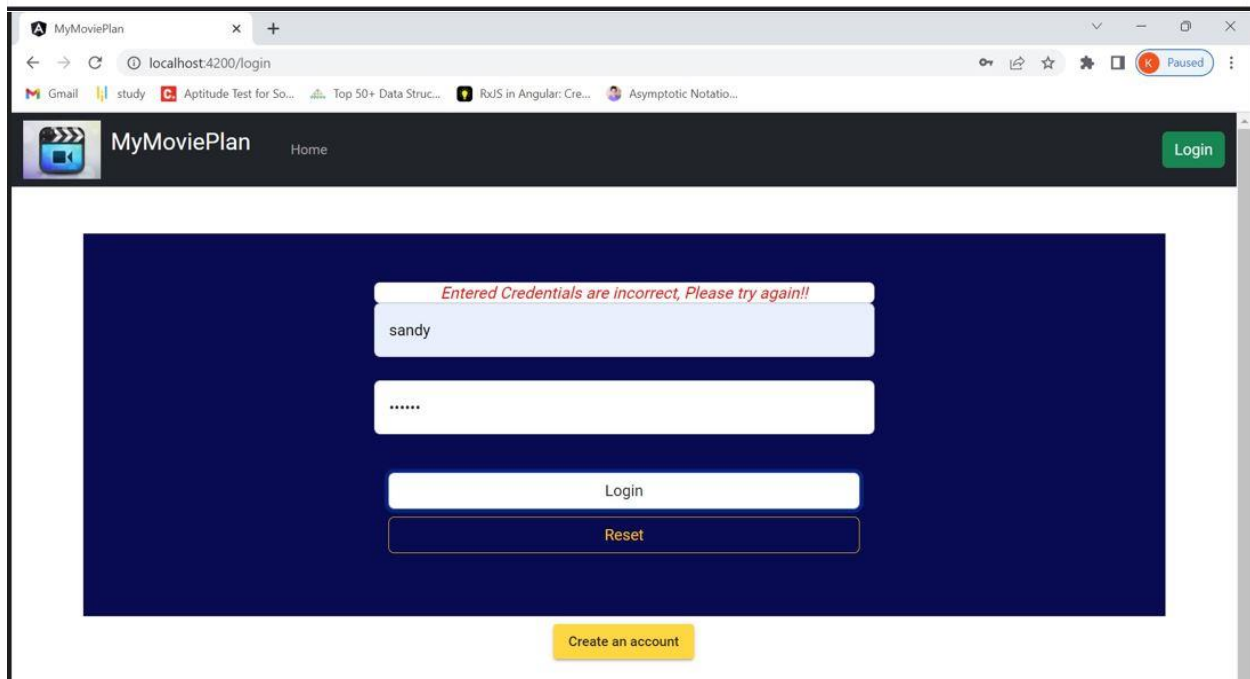
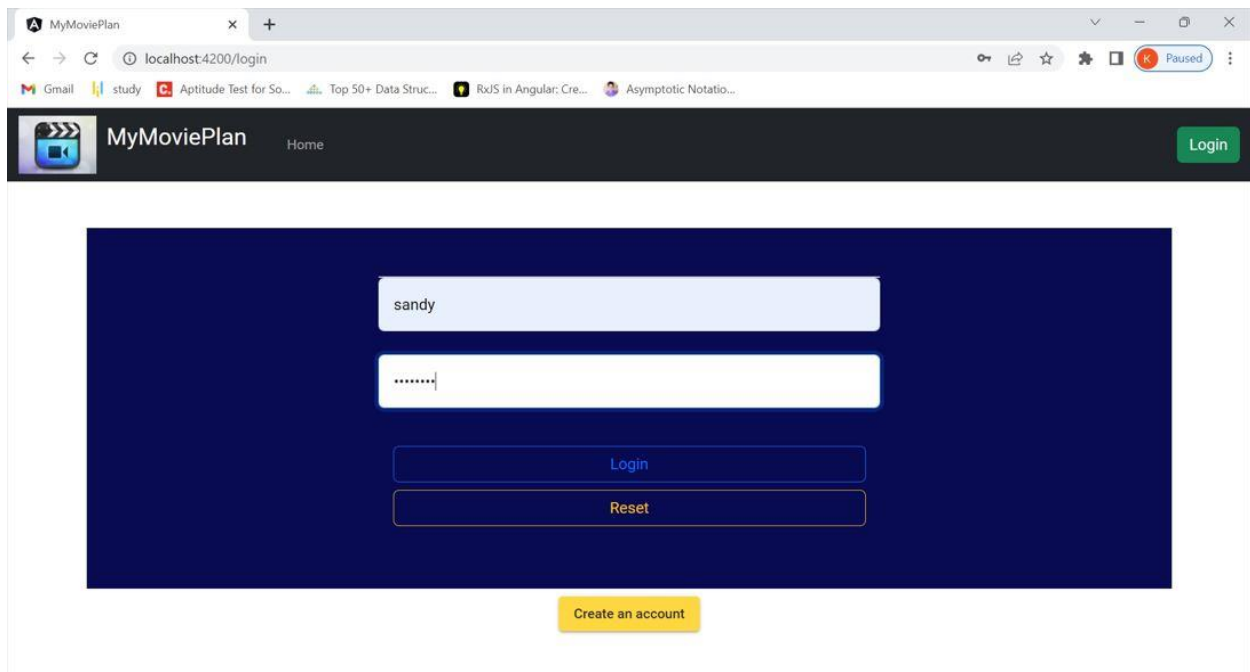
Main Actors: Nani, Keerthy Suresh, Shine Tom Chacko

₹350.00 ₹480.00

View Details







MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

mymovieplan_db1

Tables

cart

hibernate_sequence

movie_poster

movie_tickets

order_detail

role

ticket_posters

user_role

users

Views

Administration Schemas

No object selected

Query 1 x

```

1 • use mymovieplan_db1;
2 • select * from order_detail;
3 • select * from users;
4 • select * from cart;
5 • select * from movie_tickets;

```

Result Grid

ticket_id	actors	actual_price	description	discounted_price	genre	movie
5	Dave Bautista, Jonathan Groff, Ben Aldridge	520	While vacationing, a girl and her parents are ta...	480	Horror-Mystery-Thriller	Knock
6	Rishab Shetty, Kishore Kumar, G. Adhyuth Kumar	450	It involves culture of Kambala and Bhootha Kola...	400	Action-Adventure-Thriller-Drama	Kantar
7	Vijay Sethupathi, Varalaxmi Sarathkumar, Divya...	450	The gang fighting over places. A youngster dec...	400	Action-Crime-Drama	Michae
9	Nani, Keerthy Suresh, Shine Tom Chacko	480	Set in the backdrop of Singareni coal mines near...	350	Action-Adventure-Drama	Dasari
14	Shahrukh Khan, Deepika Padukone, John Abrah...	420	An Indian agent races against a doomsday doc...	395	Action-Adventure-Drama	Pathar
16	Ranbir Kapoor, Shraddha Kapoor, Hasleen Kaur	480	Madness ensues when a 'player' in the world of ...	380	comedy_Romance	Tu Jhc
18	Kartik Aaryan, Kriti Sanon, Paresh Rawal	480	Bantu is hated by his father Valmiki since he wa...	410	Action-Comedy-Drama	Shehz
62	Sam Worthington, Zoe Saldana, Sigourney Wea...	620	Jake Sully lives with his newfound family formed...	540	Action-Adventure-Fantasy	Avata

movie_tickets 5 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2	13:36:35	select * from order_detail LIMIT 0, 1000	20 row(s) returned	0.016 sec / 0.000 sec
3	16:32:17	select * from order_detail LIMIT 0, 1000	23 row(s) returned	0.000 sec / 0.000 sec
4	17:35:26	use mymovieplan_db1	0 row(s) affected	0.000 sec
5	17:35:28	select * from order_detail LIMIT 0, 1000	29 row(s) returned	0.000 sec / 0.000 sec
6	17:36:00	select * from cart LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
7	17:36:07	select * from movie_tickets LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

employee

kitchen_story

medicare

medicare_datastorage

medicare_web

Tables

Views

Stored Procedures

Functions

mymovieplan_db1

Tables

cart

Administration Schemas

No object selected

Query 1 x

```

1 • use mymovieplan_db1;
2 • select * from order_detail;
3 • select * from users;
4 • select * from cart;
5 • select * from movie_tickets;
6

```

Result Grid

user_name	full_name	userpassword
admin123	admin	\$2a\$10\$XyFbyYn6jYnYqmb3bpuoGhtYa.nKx...
harsh		\$2a\$10\$3.cPF.K3ZnZl2pnzWxFS10SjCtRBPz...
harshita	Harshita Sahu	\$2a\$10\$2Dq4tWHg8/7D7gA79Wc.PeRs9Ro48...
raj123	Raj Malhotra	\$2a\$10\$OpLxs4e1.DV/68MdOAYQWuEcDwnUe...
sandy	Sandeep Malhotra	\$2a\$10\$4cm.TbW/8F/vfE7XD2G01.69nNP1K7K...
varika	Varika Awasthi	\$2a\$10\$7c27bD5WdzAceFFKCb.9QYndunYY9...

users 2 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	19:50:00	use mymovieplan_db1	0 row(s) affected	0.000 sec
2	19:50:02	select * from users LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
3	19:53:41	select * from users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec

Query Completed

MyMoviePlan

localhost:4200/userReg

MyMoviePlan Home Login

Username

User Full Name:

Password

Register

MyMoviePlan

localhost:4200/userReg

MyMoviePlan Home Login

Username

vartika

User Full Name:

Vartika Awasthi

Password

Register

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

mymovieplan_db1

Tables

- hibernate_sequence
- movie_poster
- movie_tickets
- order_detail
- role
- ticket_posters
- user_role
- users

Views

Administration Schemas

Information

No object selected

Query 1 x

```

1 use mymovieplan_db1;
2 select * from order_detail;
3 select * from users;
4 select * from cart;
5 select * from movie_tickets;

```

Result Grid

order_id	order_alternate_number	order_amount	order_contact_number	order_full_address	order_fullname	order_status	movie_ticket_ticket_id	user_user_
26	8800994455	380	6789760000	Astak Heights, 120-h, Mumbai	manjit	Placed	16	raj123
27	667777889	1185	997777009	Fanaloo Heights, 112-8, Pune	Adira Jaamwal	Placed	14	harshita
32	6655447788	480	7766889900	Hyderabad	manjit	Placed	5	harshita
33	6655447788	380	7766889900	Hyderabad	manjit	Placed	16	harshita
34	6655447788	1200	7766889900	Hyderabad	manjit	Placed	7	harshita
35	8800994455	480	7766558899	Hyderabad	Dulquer Salman	Placed	5	harshita
36	8800994455	380	7766558899	Hyderabad	Dulquer Salman	Placed	16	harshita
37	8800994455	400	7766558899	Hyderabad	Dulquer Salman	Placed	7	harshita
38	6655447788	480	8800998877	Nanatak colony, 18-H, Pune	manjit	Placed	5	harshita
39	6655447788	380	8800998877	Nanatak colony, 18-H, Pune	manjit	Placed	16	harshita

order_detail 3 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	13:36:32	use mymovieplan_db1	0 row(s) affected	0.000 sec
2	13:36:35	select * from order_detail LIMIT 0, 1000	20 row(s) returned	0.016 sec / 0.000 sec
3	16:32:17	select * from order_detail LIMIT 0, 1000	23 row(s) returned	0.000 sec / 0.000 sec
4	17:35:26	use mymovieplan_db1	0 row(s) affected	0.000 sec
5	17:35:28	select * from order_detail LIMIT 0, 1000	29 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

(229) 55. Payment Integration in x MyMoviePlan x 2.8 million+ Stunning Free Image x +

localhost:4200/buyTicket;isSingleTicketCheckout=false;id=0

Gmail study Aptitude Test for So... Top 50+ Data Struc... RxJS in Angular: Cre... Asymptotic Notatio...

MyMoviePlan Home User

Home Cart My Orders

Full Name: Manas Tiwari

Full Address: 112/B Red cross society, Bhopal

Contact Number: 9089786756

Alternate Contact Number: 9182736455

MyMoviePlan_Pay...

Pay With UPI QR

Scan the QR using any UPI app on your phone.

QR Code is valid for 11:36 minutes

Pay Using UPI ID

UPI ID

Enter your UPI ID

success@razorpay

Account Secured by Razorpay

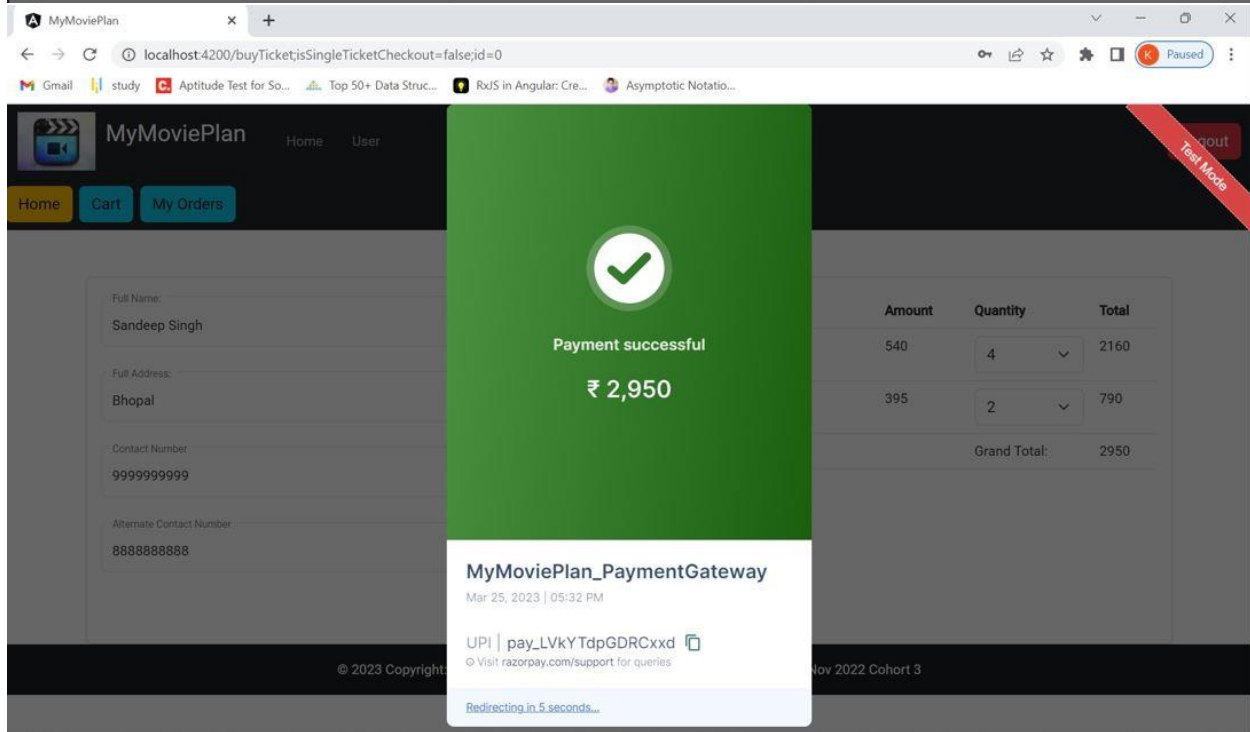
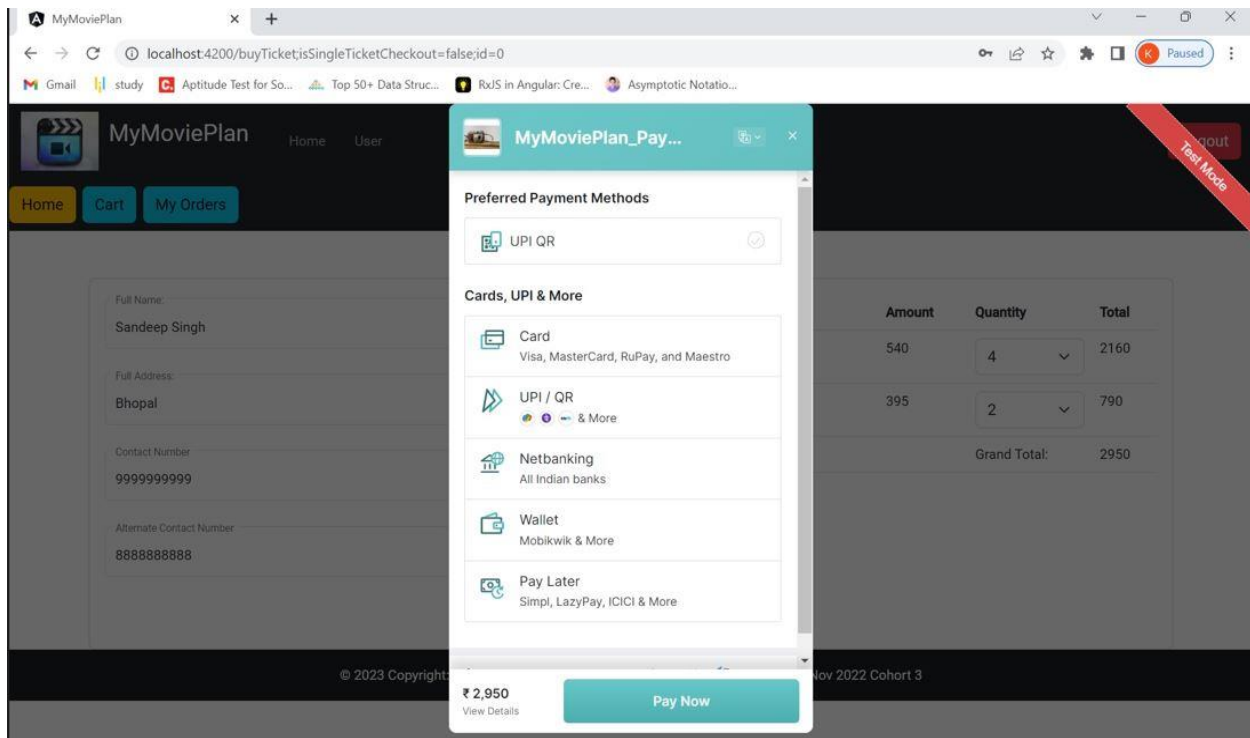
₹ 2,800

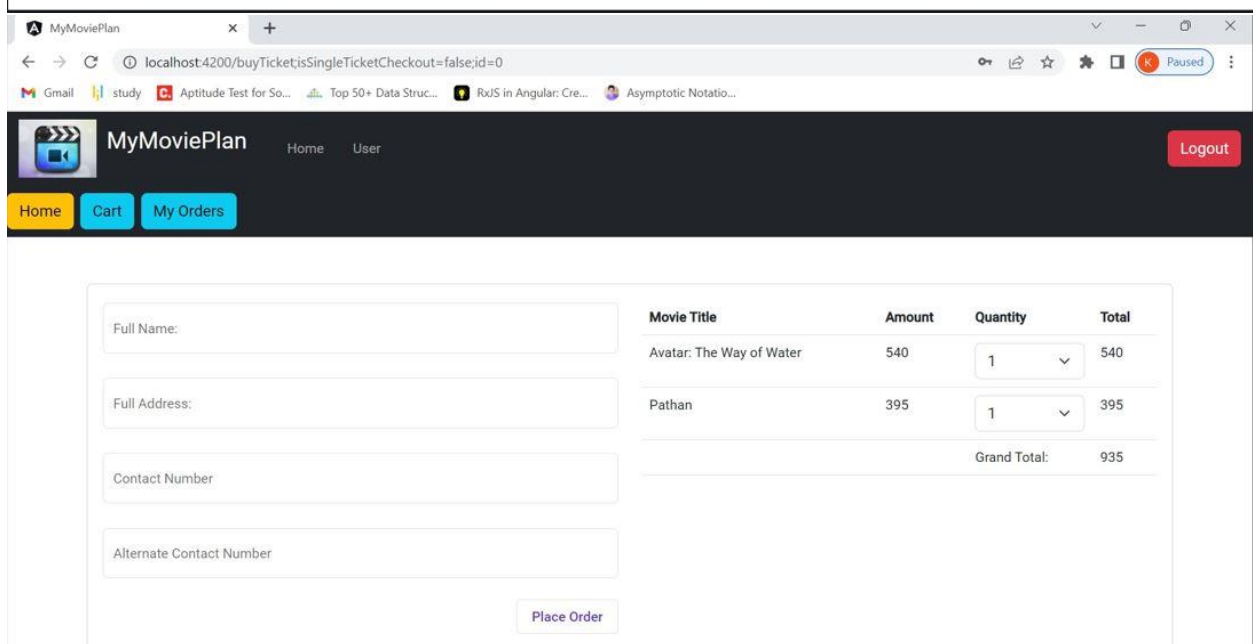
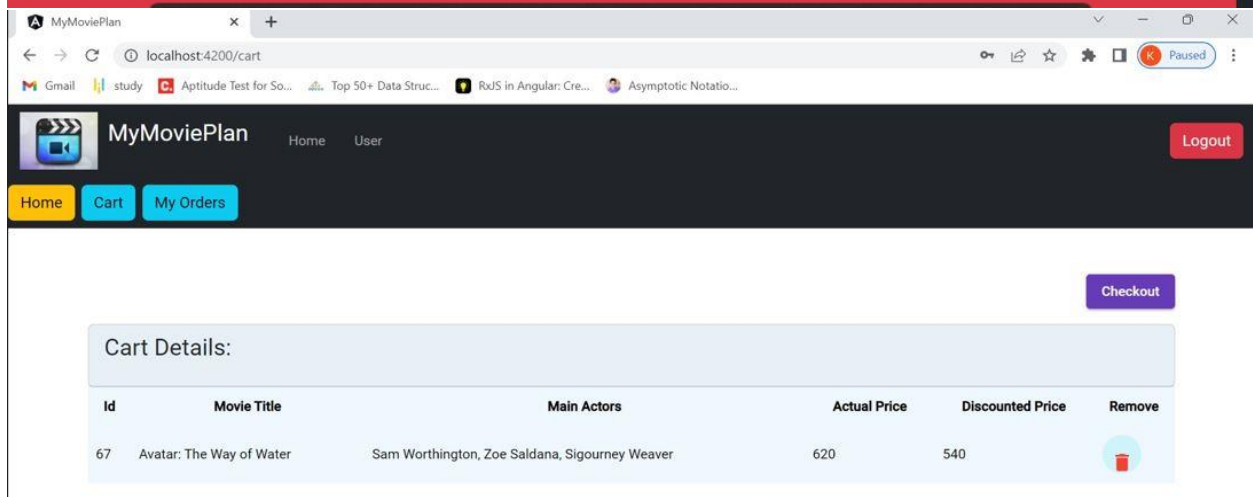
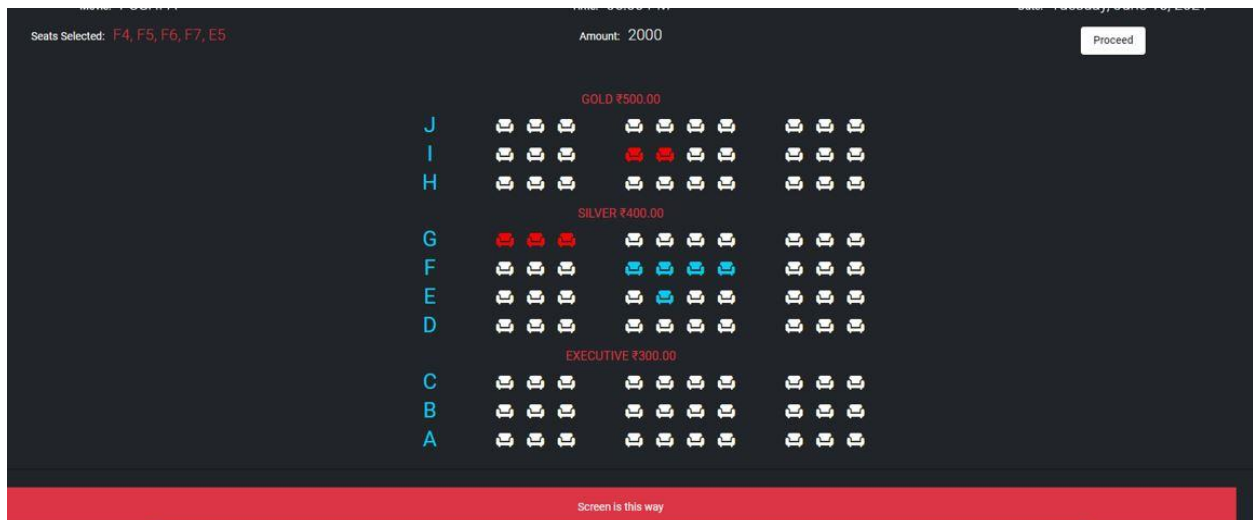
View Details

Pay Now

Grand Total: 2800

© 2023 Copyright Nov 2022 Cohort 3





MyMoviePlan

HomeUser

Logout

HomeCartMy Orders

Full Name:
Sandeep Singh

Full Address:
Bhopal

Contact Number:
9999999999

Alternate Contact Number:
8888888888

Movie Title	Amount	Quantity	Total
Avatar: The Way of Water	540	1	540
Pathan	395	1	395
Grand Total:			935

Place Order

MyMoviePlan

HomeUser

Logout

HomeCartMy Orders

Full Name:
Sandeep Singh

Full Address:
Bhopal

Contact Number:
9999999999

Alternate Contact Number:
8888888888

Movie Title	Amount	Quantity	Total
Avatar: The Way of Water	540	4	2160
Pathan	395	2	790
Grand Total:			2950

Place Order

MyMoviePlan

HomeUser

Logout



Home

Cart

My Orders

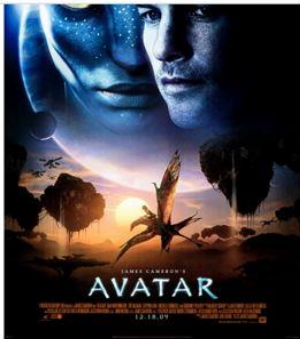
Checkout

Cart Details:

Id	Movie Title	Main Actors	Actual Price	Discounted Price	Remove
68	Avatar: The Way of Water	Sam Worthington, Zoe Saldana, Sigourney Weaver	620	540	
71	Pathan	Shahrukh Khan, Deepika Padukone, John Abraham	480	395	

MyMoviePlan

localhost:4200/ticketViewDetails;ticketId=62



home.


Main Actors:
Sam Worthington, Zoe Saldana, Sigourney Weaver


Director:
James Cameron

₹540.00 ₹620.00

Buy Now

Add To Cart

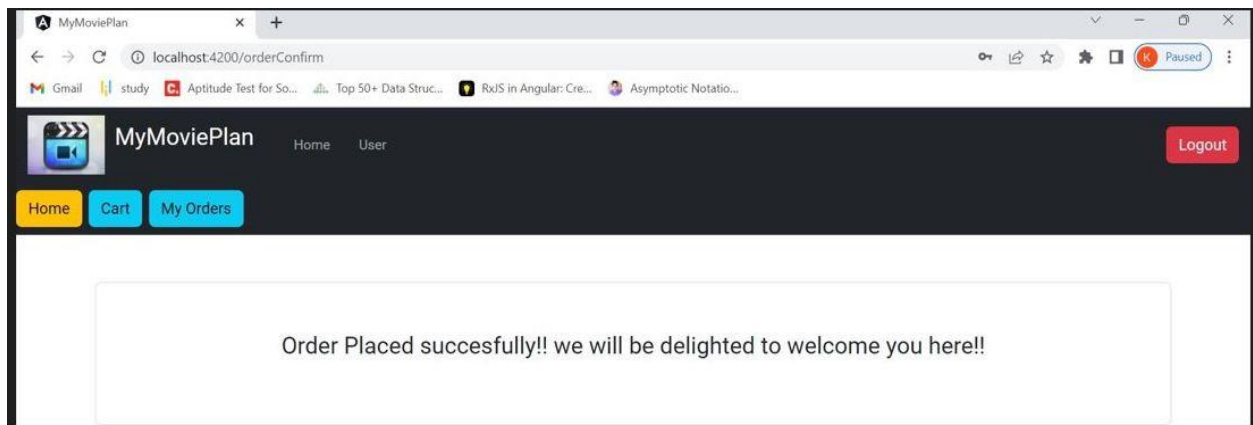


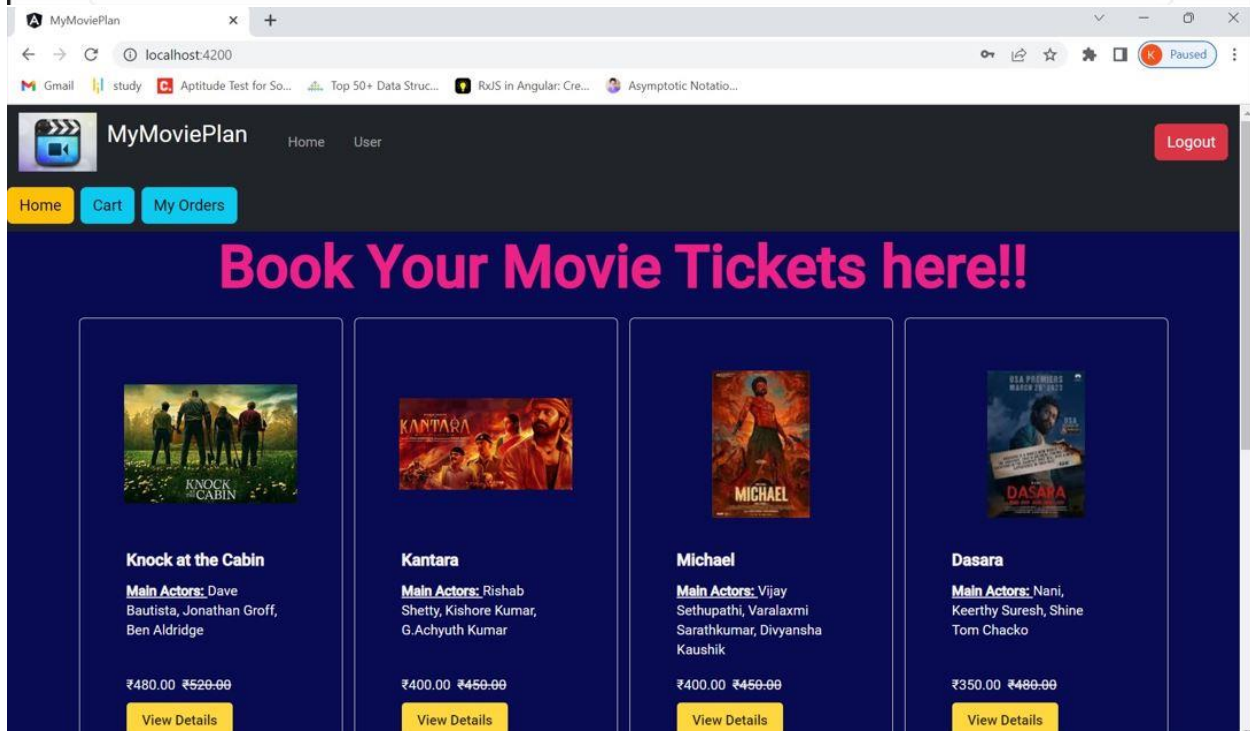
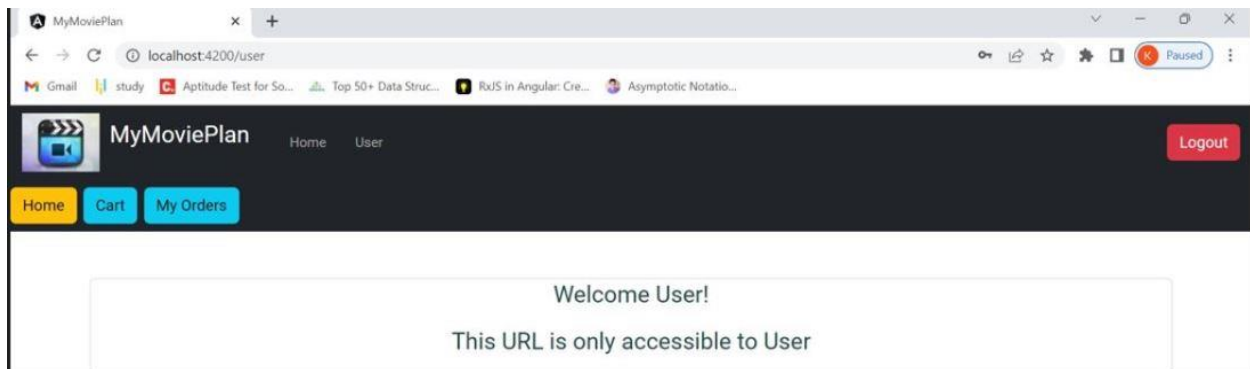


Checkout

Cart Details:					
Id	Movie Title	Main Actors	Actual Price	Discounted Price	Remove
68	Avatar: The Way of Water	Sam Worthington, Zoe Saldana, Sigourney Weaver	620	540	
69	Pathan	Shahrukh Khan, Deepika Padukone, John Abraham	480	395	
70	Michael	Vijay Sethupathi, Varalaxmi Sarathkumar, Divyansha Kaushik	450	400	

Order Details:				
Name	Contact Number	Order Status	Order Amount	Movie Title
Manan Godbole	7766558899	Placed	1920	Knock at the Cabin
Adira Jaamwal	7766558899	Watched	1600	Kantara
Harleen kaur	6699880077	Watched	1520	Tu Jhoothi Main Makkaar
Manas Tiwari	9089786756	Placed	1580	Pathan
Manas Tiwari	9089786756	Placed	820	Shehzada
Manas Tiwari	9089786756	Placed	400	Michael
Sandeep Singh	9999999999	Placed	2160	Avatar: The Way of Water
Sandeep Singh	9999999999	Placed	790	Pathan





MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

mymovieplan_db1

- Tables
 - cart
 - hibernate_sequence
 - movie_poster
 - movie_tickets
 - order_detail
 - role
 - ticket_posters
 - user_role
 - users
- Views

Administration Schemas

Information

No object selected

Query 1 x

```
1 • use mymovieplan_db1;
2 • select * from order_detail;
3 • select * from users;
4 • select * from cart;
5 • select * from movie_tickets;
```

Result Grid

	user_name	full_name	userpassword
▶	admin123	admin	\$2a\$10\$3zseVCo2ArU8H78jKGHNie81H.ZzceRtt...
	harsh		\$2a\$10\$3LdPF.K32NZI2pnzWXFs1OSjSCT/RBFz...
	harshita	Harshita Sahu	\$2a\$10\$2DqITWHg8/7D7gA79Wc.PeRs5Ro48...
	raj123	Raj Malhotra	\$2a\$10\$QpLx54e1.DV/68MdOATQWUEc0wnUe...
	sandy	Sandeep Malhotra	\$2a\$10\$vkcm.TbiW8f/gYE7KD2G01.6j9ghVP.kZk...
*	ROLE	ROLE	ROLE

users 5 x

Apply Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
3	16:32:17	select * from order_detail LIMIT 0, 1000	23 row(s) returned	0.000 sec / 0.000 sec
4	17:35:26	use mymovieplan_db1	0 row(s) affected	0.000 sec
5	17:35:28	select * from order_detail LIMIT 0, 1000	29 row(s) returned	0.000 sec / 0.000 sec
6	17:36:00	select * from cart LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
7	17:36:07	select * from movie_tickets LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec
8	17:36:28	select * from users LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

