**My MoviePlan**

1.BackendCode

MovieFinalApplication.java

package com.cg.mts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2

@SpringBootApplication

@ComponentScan("com.cg")

public class MovieFinalApplication {

    public static void main(String[] args) {

        SpringApplication.run(MovieFinalApplication.class, args);

    }

}

AdminController.java

package com.cg.mts.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

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

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

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

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

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

import com.cg.mts.service.IAdminService;

/\*\*

 \*

 \* @author Vidya

 \* @category Login

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/admin")

public class AdminController {

    Logger logger = LoggerFactory.getLogger(AdminController.class);

    @Autowired

    IAdminService adminService;

    /\*\*

     \* takes input of Admin username and password. creates an admin.

     \*

     \* @param username

     \* @param password

     \* @return HTTP status

     \* @throws Exception

     \*/

    @PostMapping("/registeradmin/{username}/{password}")

    public HttpStatus registerAdmin(@PathVariable String username, @PathVariable String password) throws Exception {

        logger.info("-------------------Admin created-----------------");

        adminService.registerAdmin(username, password);

        return HttpStatus.CREATED;

    }

}

BookingController.java

package com.cg.mts.controller;

import java.time.LocalDate;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.format.annotation.DateTimeFormat;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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.RequestMapping;

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

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

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.Screen;

import com.cg.mts.exception.AccessForbiddenException;

import com.cg.mts.exception.BookingNotFoundException;

import com.cg.mts.exception.ScreenNotFoundException;

import com.cg.mts.service.IBookingService;

/\*\*

 \*

 \* @author JyothiKinkar

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/booking")

public class BookingController {

    Logger logger = LoggerFactory.getLogger(BookingController.class);

    @Autowired

    private IBookingService bookingService;

    @Autowired

    LoginController loginController;

    /\*\*

     \*

     \* @param t

     \* @param customerId

     \* @return booking details

     \* @throws AccessForbiddenException

     \* @throws BookingNotFoundException

     \*/

    @PostMapping(value = "/insert", consumes = MediaType.APPLICATION\_JSON\_VALUE)

    public ResponseEntity<Booking> addTicketBooking(@RequestBody Booking t,

            @RequestParam(required = false) Integer customerId,@RequestParam(required = false) Integer showId)

            throws AccessForbiddenException, BookingNotFoundException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Login");

         \*/

        //logger.info("-------Booking For Customer " + customerId + "Added---------");

        return ResponseEntity.ok(bookingService.addBooking(t, customerId,showId));

    }

    /\*\*

     \*

     \* @return bookingList

     \* @throws AccessForbiddenException

     \* @throws BookingNotFoundException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Booking>> viewAllBookings() throws AccessForbiddenException, BookingNotFoundException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------List Of Bookings Fetched Successfully---------");

        return ResponseEntity.ok(bookingService.viewBookingList());

    }

    /\*\*

     \*

     \* @param t

     \* @return updated booking details

     \* @throws BookingNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @PutMapping(value = "/update", consumes = MediaType.APPLICATION\_JSON\_VALUE)

    public Booking updateTicketBooking(@RequestBody Booking t)

            throws BookingNotFoundException, AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------Booking Updated Successfully---------");

        return bookingService.updateBooking(t);

    }

    /\*\*

     \*

     \* @param bookingId

     \* @return deleted booking details

     \* @throws BookingNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @DeleteMapping("ticketbooking/{bookingId}")

    public Booking deleteTicketBookingById(@PathVariable int bookingId)

            throws BookingNotFoundException, AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------Booking With Booking Id " + bookingId + " Deleted Successfully---------");

        return bookingService.cancelBooking(bookingId);

    }

    @GetMapping("/viewbooking/{bookingId}")

    public ResponseEntity<Booking> viewBooking(@PathVariable int bookingId)

            throws BookingNotFoundException {

        ResponseEntity<Booking> response = null;

        try {

            Booking booking = bookingService.viewBooking(bookingId);

            response = new ResponseEntity<>(booking, HttpStatus.OK);

            logger.info("-------Screen Found---------");

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

            throw new BookingNotFoundException("Booking dosen't exist");

        }

        return response;

    }

    /\*\*

     \*

     \* @param movieId

     \* @return bookingList by movie id

     \* @throws AccessForbiddenException

     \* @throws BookingNotFoundException

     \*/

    @GetMapping("/byMovie/{movieId}")

    public ResponseEntity<List<Booking>> viewMovieByMovieId(@PathVariable int movieId)

            throws AccessForbiddenException, BookingNotFoundException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------Bookings With MovieId " + movieId + " Fetched Successfully---------");

        return ResponseEntity.ok(bookingService.showAllBookings(movieId));

    }

    /\*\*

     \*

     \* @param date

     \* @return bookingList by date

     \* @throws AccessForbiddenException

     \* @throws BookingNotFoundException

     \*/

    @GetMapping("/byDate/{date}")

    public ResponseEntity<List<Booking>> viewMovieByLocalDate(

            @RequestParam("bookingDate") @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate date)

            throws AccessForbiddenException, BookingNotFoundException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------Bookings With Date " + date + " Fetched Successfully---------");

        return ResponseEntity.ok(bookingService.showAllBookings(date));

    }

    /\*\*

     \*

     \* @param bookingId

     \* @return total cost of booking

     \* @throws AccessForbiddenException

     \* @throws BookingNotFoundException

     \*/

    @GetMapping("/cost/{bookingId}")

    public double TotalBookingost(@PathVariable int bookingId)

            throws AccessForbiddenException, BookingNotFoundException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin");

         \*/

        logger.info("-------Total Cost Of Booking displayed Successfully---------");

        return bookingService.calculateTotalCost(bookingId);

    }

}

CostomerController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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.RequestMapping;

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

import com.cg.mts.exception.AccessForbiddenException;

import com.cg.mts.exception.CustomerNotFoundException;

import com.cg.mts.exception.MovieNotFoundException;

import com.cg.mts.pojo.Customer;

import com.cg.mts.pojo.Movie;

import com.cg.mts.service.CustomerService;

/\*\*

 \*

 \* @author Ganga Bhavani

 \* @category Customer

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/customer")

public class CustomerController {

    Logger logger = LoggerFactory.getLogger(CustomerController.class);

    @Autowired

    private CustomerService customerService;

    @Autowired

    LoginController loginController;

    /\*\*

     \* Checks the login status if login status is true then adds customer details.

     \*

     \* @param customer

     \* @return response

     \* @throws CustomerNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @PostMapping("/add")

    public ResponseEntity<String> addCustomer(@RequestBody Customer customer)

            throws CustomerNotFoundException, AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Login"); if

         \* (!loginController.getRole().equalsIgnoreCase("ADMIN")) throw new

         \* AccessForbiddenException("You must be Admin to access this...");

         \*/

        ResponseEntity<String> response = null;

        if (customer == null) {

            logger.error("-------------Please Enter Customer Values--------");

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            customer = customerService.addCustomer(customer);

            response = new ResponseEntity<>("Customer is Added", HttpStatus.CREATED);

            logger.info("----------------Customer Created------------------");

        }

        return response;

    }

    /\*\*

     \* checks whether login status is true removes the customer using id

     \*

     \* @param customerId

     \* @return removeCustomer

     \* @throws CustomerNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @DeleteMapping("/delete/{customerId}")

    public Customer removeCustomer(@PathVariable int customerId)

            throws CustomerNotFoundException, AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) { throw new

         \* AccessForbiddenException("Not Logged In/Invalid Login"); } if

         \* (!loginController.getRole().equalsIgnoreCase("ADMIN")) { throw new

         \* AccessForbiddenException("You must be Admin to access this..."); }

         \*/

        logger.info("----------------Customer Deleted Successfully--------------");

        return customerService.removeCustomer(customerId);

    }

    /\*\*

     \* checks weather login status is true updates the customer details.

     \*

     \* @param customer

     \* @return response

     \* @throws CustomerNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @PutMapping("/update")

    public ResponseEntity<Customer> updateCustomer(@RequestBody Customer customer)

            throws CustomerNotFoundException, AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Login");

         \*/

        ResponseEntity<Customer> response = null;

        if (customer == null) {

            logger.error("Enter Customer Details");

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            customer = customerService.updateCustomer(customer);

            response = new ResponseEntity<>(customer, HttpStatus.OK);

            logger.info("--------------Customer Updated Successfully-----------------");

        }

        return response;

    }

    /\*\*

     \* checks whether the login status is true. also verifies whether the role is

     \* admin or not displays the list of customers

     \*

     \* @return CustomerList

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/findall")

    public List<Customer> viewCustomerList() throws AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Login"); if

         \* (!loginController.getRole().equalsIgnoreCase("ADMIN")) throw new

         \* AccessForbiddenException("You must be Admin to access this...");

         \*/

        logger.info("---------------Customer List-----------------");

        return customerService.viewCustomerList();

    }

    @GetMapping("/view/{customerId}")

    public ResponseEntity<Customer> viewACustomer(@PathVariable int customerId) throws CustomerNotFoundException {

        ResponseEntity<Customer> response = null;

        try {

            Customer movie = customerService.viewCustomer(customerId);

            response = new ResponseEntity<>(movie, HttpStatus.OK);

            logger.info("-------Movie With Movie id " + customerId + " Found---------");

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

            throw new CustomerNotFoundException("Customer with " + customerId + " id dosen't exist");

        }

        return response;

        // return ResponseEntity.ok(moviesService.viewMovie(movieId));

    }

}

LoginController.java

package com.cg.mts.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

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

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

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

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

import com.cg.mts.pojo.Login;

import com.cg.mts.service.LoginService;

/\*\*

 \*

 \* @author Vidya

 \* @category Login

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

public class LoginController {

    Logger logger = LoggerFactory.getLogger(LoginController.class);

    @Autowired

    LoginService logServ;

    /\*\*

     \* verifies user name and password.

     \*

     \* @param username

     \* @param password

     \* @return HTTP status

     \*/

    @PostMapping("/login/{username}/{password}")

    public Login loginUser(@PathVariable String username, @PathVariable String password) {

        Login login=new Login();

        try {

            login=logServ.loginWithData(username, password);

        } catch (Exception e) {

            logger.error("------------------LoginFailed---------------");

            return login;

        }

        logger.info("-----------------Login Successful----------------");

        return login;

    }

    /\*\*

     \* logs out the present user.

     \*

     \* @return HTTP status

     \* @throws Exception

     \*/

    @PostMapping("/Logout")

    public HttpStatus logOut() throws Exception {

        if (this.loginStatus()) {

            logServ.logoutPresentUser();

            return HttpStatus.ACCEPTED;

        } else {

            throw new Exception("User Not yet Lo gged In");

        }

    }

    /\*\*

     \* Informs whether the user is logged in.

     \*

     \* @return LoginStatus

     \*/

    public boolean loginStatus() {

        return logServ.loginStatus();

    }

    /\*\*

     \* Obtains the role of the user.

     \*

     \* @return Role

     \*/

    public String getRole() {

        return logServ.getRole();

    }

}

MoviesController.java

package com.cg.mts.controller;

import java.io.IOException;

import java.time.LocalDate;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.format.annotation.DateTimeFormat;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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.RequestMapping;

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

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

import org.springframework.web.multipart.MultipartFile;

import com.cg.mts.service.MoviesService;

import com.cg.mts.exception.MovieNotFoundException;

import com.cg.mts.pojo.Movie;

/\*\*

 \* Movie Controller Administrator can Create,Read,Update,Delete Movie Records

 \* and can also find Movies based on Date and Theater.

 \*

 \* @author Surya

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/movies")

public class MoviesController {

    Logger logger = LoggerFactory.getLogger(MoviesController.class);

    @Autowired

    private MoviesService moviesService;

    /\*\*

     \* Stores a Movie object in the Database.

     \*

     \* @param movie

     \* @return Movie

     \* @throws MovieNotFoundException

     \* @throws IOException

     \* @throws AccessForbiddenException

     \*/

    @PostMapping("/add")

    public ResponseEntity<Movie> addMovie(@RequestBody Movie movie)

            throws MovieNotFoundException, IOException {

        movie = moviesService.addMovie(movie);

        logger.info("-------Movie Added Successfully---------");

        return new ResponseEntity<>(movie, HttpStatus.CREATED);

    }

    /\*\*

     \* Updates a existing Movie record in the database.

     \*

     \* @param movie

     \* @return Movie

     \* @throws MovieNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @PutMapping("/update")

    public ResponseEntity<Movie> updateMovie(@RequestBody Movie movie)

            throws MovieNotFoundException {

        ResponseEntity<Movie> response = null;

        if (movie == null) {

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            movie = moviesService.updateMovie(movie);

            response = new ResponseEntity<>(movie, HttpStatus.OK);

            logger.info("-------Movie Updated Successfully---------");

        }

        return response;

    }

    @PutMapping("/map")

    public ResponseEntity<Movie> addToShow(@RequestBody Movie movie,@RequestParam(required = false) Integer showId)

            throws MovieNotFoundException {

        ResponseEntity<Movie> response = null;

        if (movie == null) {

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            movie = moviesService.addMovieToShow(movie,showId);

            response = new ResponseEntity<>(movie, HttpStatus.OK);

            logger.info("-------Movie Updated Successfully---------");

        }

        return response;

    }

    /\*\*

     \* Return's the List of Movies from the Database

     \*

     \* @return List<Movie>

     \* @throws MovieNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Movie>> viewMovieList() throws MovieNotFoundException {

        logger.info("-------Movie List Fetched---------");

        return ResponseEntity.ok(moviesService.viewMovieList());

    }

    /\*\*

     \* Returns the record from the database using identifier - movieId

     \*

     \* @param movieId

     \* @return Movie

     \* @throws MovieNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/viewMovie/{movieId}")

    public ResponseEntity<Movie> viewMovie(@PathVariable int movieId)

            throws MovieNotFoundException {

        ResponseEntity<Movie> response = null;

        try {

            Movie movie = moviesService.viewMovie(movieId);

            response = new ResponseEntity<>(movie, HttpStatus.OK);

            logger.info("-------Movie With Movie id " + movieId + " Found---------");

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

            throw new MovieNotFoundException("Movie with " + movieId + " id dosen't exist");

        }

        return response;

        // return ResponseEntity.ok(moviesService.viewMovie(movieId));

    }

    /\*\*

     \* Removes persisted Movie instance from the Database.

     \*

     \* @param movieId

     \* @return Movie

     \* @throws MovieNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @DeleteMapping("/delete/{movieId}")

    public ResponseEntity<Movie> removeMovie(@PathVariable int movieId)

            throws MovieNotFoundException {

        ResponseEntity<Movie> response = null;

        Movie movie = moviesService.viewMovie(movieId);

        if (movie == null) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        } else {

            moviesService.removeMovie(movieId);

            response = new ResponseEntity<>(movie, HttpStatus.OK);

            logger.info("-------Movie With Movie id " + movieId + " Deleted---------");

        }

        return response;

    }

    /\*\*

     \* Displays List of movies based on the TheatreId.

     \*

     \* @param theatreId

     \* @return Movie

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/byTheatre/{theatreId}")

    public List<Movie> viewMovieByTheatreId(@PathVariable int theatreId)  {

        logger.info("-------Movies With TheatreId " + theatreId + " Found---------");

        return moviesService.viewMovieList(theatreId);

    }

    /\*\*

     \* Returns the list of Movies based on the Date.

     \*

     \* @param date

     \* @return Movie

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/byDate/{date}")

    public List<Movie> viewMovieByLocalDate(

            @RequestParam("movieDate") @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate date) {

        logger.info("-------Movies With Date " + date + " Found---------");

        return moviesService.viewMovieList(date);

    }

}

ScreenController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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.RequestMapping;

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

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

import com.cg.mts.exception.ScreenNotFoundException;

import com.cg.mts.pojo.Screen;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.service.ScreenService;

/\*\*

 \*

 \* @author Thejesh

 \* @category ScreenController

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/screens")

public class ScreenController {

    Logger logger = LoggerFactory.getLogger(ScreenController.class);

    @Autowired

    private ScreenService screenService;

    /\*\*

     \*

     \* @param screen

     \* @param theatreId

     \* @return addedScreen

     \* @throws AccessForbiddenException

     \* @throws ScreenNotFoundException

     \*/

    @PostMapping("/add")

    public ResponseEntity<Screen> addAScreen(@RequestBody Screen screen,

            @RequestParam(required = false) Integer theatreId)

            throws ScreenNotFoundException {

        logger.info("-------Screen Successfully added into Theatre " + theatreId + "---------");

        return ResponseEntity.ok(screenService.addScreen(screen, theatreId));

    }

    /\*\*

     \*

     \* @return screenList

     \* @throws AccessForbiddenException

     \* @throws ScreenNotFoundException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Screen>> viewScreenList() throws  ScreenNotFoundException {

        logger.info("-------List Of Screens Fetched Successfully---------");

        return ResponseEntity.ok(screenService.viewScreenList());

    }

    @GetMapping("/theatre/{screenId}")

    public ResponseEntity<Theatre>  getTheatreById(@PathVariable int screenId) throws ScreenNotFoundException {

        ResponseEntity<Theatre> response = null;

        try {

            Theatre theatre = screenService.getTheatre(screenId);

            response = new ResponseEntity<>(theatre, HttpStatus.OK);

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

        return response;

    }

    @GetMapping("/viewScreen/{screenId}")

    public ResponseEntity<Screen> viewScreen(@PathVariable int screenId)

            throws ScreenNotFoundException {

        ResponseEntity<Screen> response = null;

        try {

            Screen screen = screenService.viewScreen(screenId);

            response = new ResponseEntity<>(screen, HttpStatus.OK);

            logger.info("-------Screen Found---------");

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

            throw new ScreenNotFoundException("Screen dosen't exist");

        }

        return response;

    }

    /\*\*

     \*

     \* @param s

     \* @param theatreId

     \* @return updatedScreen details

     \* @throws AccessForbiddenException

     \* @throws ScreenNotFoundException

     \*/

    @PutMapping("/update")

    public ResponseEntity<Screen> updateScreen(@RequestBody Screen s, @RequestParam(required = false) Integer theatreId)

            throws  ScreenNotFoundException {

        ResponseEntity<Screen> response = null;

        if (s == null) {

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            s = screenService.updateScreen(s, theatreId);

            response = new ResponseEntity<>(s, HttpStatus.OK);

            logger.info("-------Sceen Updated Successfully---------");

        }

        return response;

    }

}

SeatController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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.RequestMapping;

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

import com.cg.mts.exception.AccessForbiddenException;

import com.cg.mts.exception.SeatNotFoundException;

import com.cg.mts.pojo.Seat;

import com.cg.mts.service.ISeatService;

/\*\*

 \*

 \* @author Surya

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/seats")

public class SeatController {

    Logger logger = LoggerFactory.getLogger(SeatController.class);

    @Autowired

    private ISeatService seatService;

    @Autowired

    LoginController loginController;

    /\*\*

     \*

     \* @param seat

     \* @return seatDetails

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @PostMapping("/add")

    public ResponseEntity<Seat> addASeat(@RequestBody Seat seat)

            throws AccessForbiddenException, SeatNotFoundException {

        seat = seatService.addSeat(seat);

        logger.info("-------Seat Added Successfully---------");

        return new ResponseEntity<>(seat, HttpStatus.CREATED);

    }

    /\*\*

     \*

     \* @return listOfSeats

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Seat>> viewSeatList() throws AccessForbiddenException, SeatNotFoundException {

        logger.info("-------List of Seats Fetched Successfully---------");

        return ResponseEntity.ok(seatService.viewSeatList());

    }

    /\*\*

     \*

     \* @param seat

     \* @return updated seat

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @PutMapping("/update")

    public ResponseEntity<Seat> updateSeat(@RequestBody Seat seat)

            throws AccessForbiddenException, SeatNotFoundException {

        ResponseEntity<Seat> response = null;

        if (seat == null) {

            response = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

        } else {

            seat = seatService.updateSeat(seat);

            response = new ResponseEntity<>(seat, HttpStatus.OK);

            logger.info("-------Seat Updated Successfully---------");

        }

        return response;

    }

    /\*\*

     \*

     \* @param seat

     \* @return booked seat

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @PutMapping("/book")

    public ResponseEntity<Seat> BookASeat(@RequestBody Seat seat)

            throws AccessForbiddenException, SeatNotFoundException {

        seat = seatService.bookSeat(seat);

        logger.info("-------Seat booking Successfull---------");

        return new ResponseEntity<>(seat, HttpStatus.OK);

    }

    /\*\*

     \*

     \* @param seat

     \* @return cancelled seat

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @PutMapping("/cancel")

    public ResponseEntity<Seat> CancelASeat(@RequestBody Seat seat)

            throws AccessForbiddenException, SeatNotFoundException {

        seat = seatService.cancelSeatBooking(seat);

        logger.info("-------Seat Cancellation Successfull---------");

        return new ResponseEntity<>(seat, HttpStatus.OK);

    }

    /\*\*

     \*

     \* @param seat

     \* @return blocked seat

     \* @throws AccessForbiddenException

     \* @throws SeatNotFoundException

     \*/

    @PutMapping("/block")

    public ResponseEntity<Seat> BloclASeat(@RequestBody Seat seat)

            throws AccessForbiddenException, SeatNotFoundException {

        seat = seatService.blockSeat(seat);

        logger.info("-------Seat blocking Successfull---------");

        return new ResponseEntity<>(seat, HttpStatus.OK);

    }

}

ShowController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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.RequestMapping;

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

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

import com.cg.mts.exception.ShowNotFoundException;

import com.cg.mts.pojo.Show;

import com.cg.mts.service.ShowService;

/\*\*

 \* Show Controller

 \*

 \* @author Surya

 \* @category Show Module

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/shows")

public class ShowController {

    Logger logger = LoggerFactory.getLogger(ShowController.class);

    @Autowired

    private ShowService showService;

    /\*\*

     \* Stores a Show object in the Database.

     \*

     \* @param show

     \* @param theatreId

     \* @param screenId

     \* @return Show

     \* @throws AccessForbiddenException

     \*/

    @PostMapping("/add")

    public ResponseEntity<Show> addShow(@RequestBody Show show, @RequestParam(required = false) Integer theatreId,

            @RequestParam(required = false) Integer screenId)  {

        showService.addShow(show, theatreId, screenId);

        logger.info("-------Show Added Succesfully--------");

        return new ResponseEntity<>(show, HttpStatus.CREATED);

    }

    /\*\*

     \* Removes persisted Show instance from the Database.

     \*

     \* @param showId

     \* @return Show

     \* @throws AccessForbiddenException

     \*/

    @DeleteMapping("/delete/{showId}")

    public ResponseEntity<Show> removeShow(@PathVariable int showId)  {

        ResponseEntity<Show> response = null;

        Show show = showService.viewShow(showId);

        if (show == null) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        } else {

            showService.removeShow(showId);

            response = new ResponseEntity<>(show, HttpStatus.OK);

            logger.info("-------Show with ShowId " + showId + " Deleted Successfully---------");

        }

        return response;

    }

    /\*\*

     \* Updates a existing Show record in the database.

     \*

     \* @param show

     \* @param theatreId

     \* @param screenId

     \* @return Show

     \* @throws AccessForbiddenException

     \*/

    @PutMapping("/update")

    public ResponseEntity<Show> updateShow(@RequestBody Show show, @RequestParam(required = false) Integer theatreId,

            @RequestParam(required = false) Integer screenId)  {

        ResponseEntity<Show> response = null;

        if (show == null) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        } else {

            showService.updateShow(show, theatreId, screenId);

            response = new ResponseEntity<>(show, HttpStatus.OK);

            logger.info("-------Show Updated Successfully---------");

        }

        return response;

    }

    /\*\*

     \* Returns the record from the database using identifier - showId

     \*

     \* @param showId

     \* @return Show

     \* @throws AccessForbiddenException

     \* @throws ShowNotFoundException

     \*/

    @GetMapping("/view/{showId}")

    public ResponseEntity<Show> viewShow(@PathVariable int showId)

            throws  ShowNotFoundException {

        ResponseEntity<Show> response = null;

        try {

            Show show = showService.viewShow(showId);

            response = new ResponseEntity<>(show, HttpStatus.OK);

            logger.info("-------Show with ShowId " + showId + " Found Successfully---------");

        } catch (Exception e) {

            response = new ResponseEntity<>(HttpStatus.NOT\_FOUND);

            throw new ShowNotFoundException("Show with " + showId + " id dosen't exist");

        }

        return response;

    }

    /\*\*

     \* Return's the List of Shows existing from the Database

     \*

     \* @return List<Show>

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Show>> viewShowList() {

        logger.info("-------List Of Shows Fetched Successfully---------");

        return ResponseEntity.ok(showService.viewAllShows());

    }

    /\*\*

     \*

     \* @param theatreId

     \* @return Show

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/show\_theatre/{theatreId}")

    public ResponseEntity<List<Show>> viewShowByTheatreId(@PathVariable int theatreId) {

        logger.info("-------List Of Shows With TheatreId " + theatreId + " Fetched Successfully---------");

        return ResponseEntity.ok(showService.viewShowList(theatreId));

    }

    /\*\*

     \*

     \* @param date

     \* @return List<Show>

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/date/{date}")

    public ResponseEntity<List<Show>> viewShowByLocalDate(@PathVariable int date) {

        logger.info("-------List Of Shows With Date " + date + " Fetched Successfully---------");

        return ResponseEntity.ok(showService.viewShowList(date));

    }

}

ThreatreController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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.RequestMapping;

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

import com.cg.mts.exception.TheatreNotFoundException;

import com.cg.mts.pojo.Movie;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.service.TheatreService;

/\*\*

 \*

 \* @author Srinivas

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/theatre")

public class TheatreController {

    Logger logger = LoggerFactory.getLogger(TheatreController.class);

    @Autowired

    private TheatreService theatreservice;

    /\*\*

     \*

     \* @return listOfTheatres

     \* @throws AccessForbiddenException

     \* @throws TheatreNotFoundException

     \*/

    @GetMapping("/all")

    public ResponseEntity<List<Theatre>> getAlltheatres() throws  TheatreNotFoundException {

        logger.info("-------Theatre List Fetched---------");

        return ResponseEntity.ok(theatreservice.getAllTheatres());

    }

    /\*\*

     \*

     \* @param t

     \* @return inserted theatre

     \* @throws AccessForbiddenException

     \* @throws TheatreNotFoundException

     \*/

    @PostMapping("/insert")

    public ResponseEntity<Theatre> addTheatre(@RequestBody Theatre t)

            throws TheatreNotFoundException {

        logger.info("-------Theatre Added Successfully---------");

        return new ResponseEntity<>(theatreservice.addTheatre(t), HttpStatus.CREATED);

    }

    /\*\*

     \*

     \* @param t

     \* @return updated theatre

     \* @throws AccessForbiddenException

     \* @throws TheatreNotFoundException

     \*/

    @PutMapping("/update")

    public List<Theatre> updateTheatre(@RequestBody Theatre t)

            throws  TheatreNotFoundException {

        logger.info("-------Theatre Updated Successfully---------");

        return theatreservice.updateTheatre(t);

    }

    /\*\*

     \*

     \* @param theatreId

     \* @return theatre by theatreId

     \* @throws AccessForbiddenException

     \* @throws TheatreNotFoundException

     \*/

    @GetMapping("/find/{theatreId}")

    public ResponseEntity<Theatre> findTheatre(@PathVariable int theatreId)

            throws  TheatreNotFoundException {

        logger.info("-------Theatre Found with Theatre id" + theatreId + "---------");

        return ResponseEntity.ok(theatreservice.findTheatres(theatreId));

    }

    /\*\*

     \*

     \* @param theatreId

     \* @return deleted theatre

     \* @throws AccessForbiddenException

     \* @throws TheatreNotFoundException

     \*/

    @DeleteMapping("/delete/{theatreId}")

    public List<Theatre> deleteMoviesById(@PathVariable int theatreId)

            throws TheatreNotFoundException {

        logger.info("-------Theatre Deleted with Theatre id" + theatreId + "---------");

        return theatreservice.deleteTheatreById(theatreId);

    }

    @GetMapping("/findbyMovie/{movieId}")

    public ResponseEntity<List<Theatre>> findTheatreByMovieId(@PathVariable int movieId)

            throws  TheatreNotFoundException {

        return ResponseEntity.ok(theatreservice.findTheatresByMovie(movieId));

    }

}

TicketController.java

package com.cg.mts.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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.RequestMapping;

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

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

import com.cg.mts.exception.AccessForbiddenException;

import com.cg.mts.exception.TicketNotFoundException;

import com.cg.mts.pojo.Ticket;

import com.cg.mts.service.TicketService;

/\*\*

 \*

 \* @author Yogendra

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/tickets")

public class TicketController {

    Logger logger = LoggerFactory.getLogger(TicketController.class);

    @Autowired

    private TicketService ticketService;

    @Autowired

    LoginController loginController;

    /\*\*

     \*

     \* @param ticket

     \* @return added ticket

     \* @throws AccessForbiddenException

     \* @throws TicketNotFoundException

     \*/

    @PostMapping("/add")

    public ResponseEntity<Ticket> addATicket(@RequestBody Ticket ticket,@RequestParam(required = false) Integer bookingId)

            throws AccessForbiddenException, TicketNotFoundException {

        ticket = ticketService.addTicket(ticket,bookingId);

        /\* System.out.println(ticket.getSeats()); \*/

        logger.info("-------Ticked Created Successfully---------");

        return new ResponseEntity<>(ticket, HttpStatus.CREATED);

    }

    /\*\*

     \*

     \* @return ticketList

     \* @throws AccessForbiddenException

     \* @throws TicketNotFoundException

     \*/

    @GetMapping("/findall")

    public ResponseEntity<List<Ticket>> viewTicketList() throws AccessForbiddenException, TicketNotFoundException {

        logger.info("-------List of Tickets Found Successfully---------");

        return ResponseEntity.ok(ticketService.viewTicketList());

    }

    /\*\*

     \*

     \* @param ticketId

     \* @return ticket by ticketId

     \* @throws TicketNotFoundException

     \* @throws AccessForbiddenException

     \*/

    @GetMapping("/{ticketId}")

    public Ticket findATicket(@PathVariable int ticketId) throws TicketNotFoundException, AccessForbiddenException {

        Ticket t = null;

        try {

            t = ticketService.findTicket(ticketId);

            logger.info("-------Ticket with ticketId " + ticketId + " Foound Successfully---------");

        } catch (Exception e) {

            throw new TicketNotFoundException("Invalid Ticket ID");

        }

        return t;

    }

}

UserController.java

package com.cg.mts.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

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

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

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

import com.cg.mts.exception.CustomerNotFoundException;

import com.cg.mts.exception.AccessForbiddenException;

import com.cg.mts.exception.UserCreationError;

import com.cg.mts.pojo.Customer;

import com.cg.mts.pojo.User;

import com.cg.mts.repository.CustomerRepository;

import com.cg.mts.service.IUserService;

/\*\*

 \*

 \* @author Vidya

 \* @category Login

 \*

 \*/

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/user")

public class UserController {

    Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired

    IUserService userService;

    @Autowired

    CustomerRepository customerRepository;

    /\*\*

     \*

     \* @param user

     \* @return user

     \* @throws AccessForbiddenException

     \* @throws CustomerNotFoundException

     \* @throws UserCreationError

     \*/

    @PostMapping("/adduser")

    public User addUser(@RequestBody User user)

            throws AccessForbiddenException, CustomerNotFoundException, UserCreationError {

        // if(!logCon.loginStatus() & logCon.getRole().equalsIgnoreCase("ADMIN"))

        /\*

         \* if (user.getRole().equalsIgnoreCase("ADMIN") &

         \* !loginController.loginStatus()) throw new

         \* AccessForbiddenException("You must be Admin to access this..."); else

         \*/

        if (user.getRole().equalsIgnoreCase("CUSTOMER")) {

            Customer customer = new Customer(user.getUsername(), null, null, null, user.getPassword());

            logger.info("-----------------Customer Added---------------------");

            customerRepository.saveAndFlush(customer);

            user.setCustomer(customer);

        }

        logger.info("-----------------User Added---------------------");

        return userService.addUser(user);

    }

    /\*\*

     \*

     \* @param user

     \* @return remove user

     \* @throws AccessForbiddenException

     \*/

    @DeleteMapping("/removeuser")

    public User removeUser(@RequestBody User user) throws AccessForbiddenException {

        /\*

         \* if (!loginController.loginStatus()) throw new

         \* AccessForbiddenException("Not Logged In/Invalid Loggin"); if

         \* (!loginController.getRole().equalsIgnoreCase("ADMIN")) throw new

         \* AccessForbiddenException("You must be Admin to access this...");

         \*/

        logger.info("--------------------User Deleted------------------");

        return userService.removeUser(user);

    }

}

ServletInitializer.java

package com.cg.mts;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override

    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

        return application.sources(MovieFinalApplication.class);

    }

}

AccessForbiddenException.java

package com.cg.mts.exception;

public class AccessForbiddenException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public AccessForbiddenException(String message) {

        super(message);

    }

}

BookingNotFoundException.java

package com.cg.mts.exception;

public class BookingNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public BookingNotFoundException() {

    }

    public BookingNotFoundException(String message) {

        super(message);

    }

}

CustomerNotFoundException.java

package com.cg.mts.exception;

public class CustomerNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public CustomerNotFoundException() {

    }

    public CustomerNotFoundException(String message) {

        super(message);

    }

}

ErrorMapper.java

package com.cg.mts.exception;

import java.util.Date;

public class ErrorMapper {

    private String url;

    private String message;

    private Date now;

    public ErrorMapper(String url, String message, Date now) {

        super();

        this.url = url;

        this.message = message;

        this.now = now;

    }

    public String getUrl() {

        return url;

    }

    public void setUrl(String url) {

        this.url = url;

    }

    public String getMessage() {

        return message;

    }

    public void setMessage(String message) {

        this.message = message;

    }

    public Date getNow() {

        return now;

    }

    public void setNow(Date now) {

        this.now = now;

    }

}

MovieNotFoundException.java

package com.cg.mts.exception;

public class MovieNotFoundException extends Exception {

    private static final long serialVersionUID = 1L;

    public MovieNotFoundException() {

    }

    public MovieNotFoundException(String message) {

        super(message);

    }

}

ScreenNotFoundException.java

package com.cg.mts.exception;

public class ScreenNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public ScreenNotFoundException(String msg) {

        super(msg);

    }

}

SeatNotFoundexception.java

package com.cg.mts.exception;

public class SeatNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public SeatNotFoundException() {

    }

    public SeatNotFoundException(String message) {

        super(message);

    }

}

ShowNotFoundException.java

package com.cg.mts.exception;

public class ShowNotFoundException extends Exception {

    private static final long serialVersionUID = 1L;

    public ShowNotFoundException() {

    }

    public ShowNotFoundException(String message) {

        super(message);

    }

}

ThereatureNotFoundException.java

package com.cg.mts.exception;

public class TheatreNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public TheatreNotFoundException(String msg) {

        super(msg);

    }

}

TicketNotFoundException.java

package com.cg.mts.exception;

public class TicketNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public TicketNotFoundException(String msg) {

        super(msg);

    }

}

UserCreationError.java

package com.cg.mts.exception;

public class UserCreationError extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = -2928667618706847791L;

    public UserCreationError(String message) {

        super(message);

    }

}

UserNotFoundException.java

package com.cg.mts.exception;

public class UserNotFoundException extends Exception {

    /\*\*

     \*

     \*/

    private static final long serialVersionUID = 1L;

    public UserNotFoundException(String message) {

        super(message);

    }

}

UsrException.java

package com.cg.mts.exception;

import java.util.Date;

import javax.servlet.http.HttpServletRequest;

import org.springframework.http.HttpStatus;

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

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

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

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

@ControllerAdvice

public class UsrException {

    @ResponseBody

    @ResponseStatus(value = HttpStatus.NOT\_FOUND)

    @ExceptionHandler({ Exception.class })

    public ErrorMapper handleConflict(Exception ex, HttpServletRequest req) {

        String msg = ex.getMessage();

        String uri = req.getRequestURL().toString();

        return new ErrorMapper(uri, msg, new Date());

    }

}

Pojos

Booking.java

package com.cg.mts.pojo;

import java.time.LocalDate;

import javax.persistence.\*;

import com.fasterxml.jackson.annotation.JsonFormat;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import com.fasterxml.jackson.datatype.jsr310.deser.LocalDateDeserializer;

import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateSerializer;

@Entity

public class Booking {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int transactionId;

    @OneToOne(mappedBy = "booking")

    private Show show;

    @JsonDeserialize(using = LocalDateDeserializer.class)

    @JsonSerialize(using = LocalDateSerializer.class)

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")

    private LocalDate bookingDate;

    private String transactionMode;

    private String transactionStatus;

    private double totalCost;

    @JsonIgnore

    @ManyToOne

    private Customer customer;

    @JsonIgnore

    @OneToOne(mappedBy = "booking")

    private Ticket ticket;

    public Booking() {

    }

    public Booking(Show show, LocalDate bookingDate, String transactionMode, String transactionStatus, double totalCost,

            Customer customer, Ticket ticket) {

        super();

        this.show = show;

        this.bookingDate = bookingDate;

        this.transactionMode = transactionMode;

        this.transactionStatus = transactionStatus;

        this.totalCost = totalCost;

        this.customer = customer;

        this.ticket = ticket;

    }

    public int getTransactionId() {

        return transactionId;

    }

    public void setTransactionId(int transactionId) {

        this.transactionId = transactionId;

    }

    public Show getShow() {

        return show;

    }

    public void setShow(Show show) {

        this.show = show;

    }

    public LocalDate getBookingDate() {

        return bookingDate;

    }

    public void setBookingDate(LocalDate bookingDate) {

        this.bookingDate = bookingDate;

    }

    public String getTransactionMode() {

        return transactionMode;

    }

    public void setTransactionMode(String transactionMode) {

        this.transactionMode = transactionMode;

    }

    public String getTransactionStatus() {

        return transactionStatus;

    }

    public void setTransactionStatus(String transactionStatus) {

        this.transactionStatus = transactionStatus;

    }

    public double getTotalCost() {

        return totalCost;

    }

    public void setTotalCost(double totalCost) {

        this.totalCost = totalCost;

    }

    public Customer getCustomer() {

        return customer;

    }

    public void setCustomer(Customer customer) {

        this.customer = customer;

    }

    public Ticket getTicket() {

        return ticket;

    }

    public void setTicket(Ticket ticket) {

        this.ticket = ticket;

    }

}

Customer.java

package com.cg.mts.pojo;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

public class Customer {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int customerId;

    private String customerName;

    private String address;

    private String mobileNumber;

    private String email;

    private String password;

//  @JsonIgnore

//  @OneToOne

//  private User user;

    public Customer() {

    }

public Customer(String customerName, String address, String mobileNumber, String email, String password) {

    super();

    this.customerName = customerName;

    this.address = address;

    this.mobileNumber = mobileNumber;

    this.email = email;

    this.password = password;

}

public int getCustomerId() {

    return customerId;

}

public void setCustomerId(int customerId) {

    this.customerId = customerId;

}

public String getCustomerName() {

    return customerName;

}

public void setCustomerName(String customerName) {

    this.customerName = customerName;

}

public String getAddress() {

    return address;

}

public void setAddress(String address) {

    this.address = address;

}

public String getMobileNumber() {

    return mobileNumber;

}

public void setMobileNumber(String mobileNumber) {

    this.mobileNumber = mobileNumber;

}

public String getEmail() {

    return email;

}

public void setEmail(String email) {

    this.email = email;

}

public String getPassword() {

    return password;

}

public void setPassword(String password) {

    this.password = password;

}

}

Login.java

package com.cg.mts.pojo;

public class Login {

    private boolean loginStatus;

    private User user;

    public Login() {

    }

    public Login(boolean loginStatus, User user) {

        super();

        this.loginStatus = loginStatus;

        this.user = user;

    }

    public boolean isLoginStatus() {

        return loginStatus;

    }

    public void setLoginStatus(boolean loginStatus) {

        this.loginStatus = loginStatus;

    }

    public User getUser() {

        return user;

    }

    public void setUser(User user) {

        this.user = user;

    }

    public String getRole() {

        return user.getRole();

    }

}

Movie.java

package com.cg.mts.pojo;

import java.sql.Blob;

import java.time.LocalDate;

import javax.persistence.\*;

import org.hibernate.annotations.DynamicUpdate;

import com.fasterxml.jackson.annotation.JsonFormat;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import com.fasterxml.jackson.datatype.jsr310.deser.LocalDateDeserializer;

import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateSerializer;

/\*\*

 \*

 \* @author surya

 \*

 \*/

@Entity

@Table(name = "movie")

@DynamicUpdate

public class Movie {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int movieId;

    private String movieName;

    private String movieGenre;

    private String movieHours;

    private String movieLanguage;

    private String movieDescription;

    private String movieRating;

    @JsonDeserialize(using = LocalDateDeserializer.class)

    @JsonSerialize(using = LocalDateSerializer.class)

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")

    private LocalDate movieDate;

    @JsonIgnore

    @OneToOne

    private Show show;

    public Movie() {

    }

    public Movie(String movieName, String movieGenre, String movieHours, String movieLanguage, String movieDescription,

            String movieRating, LocalDate movieDate, Show show) {

        super();

        this.movieName = movieName;

        this.movieGenre = movieGenre;

        this.movieHours = movieHours;

        this.movieLanguage = movieLanguage;

        this.movieDescription = movieDescription;

        this.movieRating = movieRating;

        this.movieDate = movieDate;

        this.show = show;

    }

    public int getMovieId() {

        return movieId;

    }

    public void setMovieId(int movieId) {

        this.movieId = movieId;

    }

    public String getMovieName() {

        return movieName;

    }

    public void setMovieName(String movieName) {

        this.movieName = movieName;

    }

    public String getMovieGenre() {

        return movieGenre;

    }

    public void setMovieGenre(String movieGenre) {

        this.movieGenre = movieGenre;

    }

    public String getMovieHours() {

        return movieHours;

    }

    public void setMovieHours(String movieHours) {

        this.movieHours = movieHours;

    }

    public String getMovieLanguage() {

        return movieLanguage;

    }

    public void setMovieLanguage(String movieLanguage) {

        this.movieLanguage = movieLanguage;

    }

    public String getMovieDescription() {

        return movieDescription;

    }

    public void setMovieDescription(String movieDescription) {

        this.movieDescription = movieDescription;

    }

    public String getMovieRating() {

        return movieRating;

    }

    public void setMovieRating(String movieRating) {

        this.movieRating = movieRating;

    }

    public LocalDate getMovieDate() {

        return movieDate;

    }

    public void setMovieDate(LocalDate movieDate) {

        this.movieDate = movieDate;

    }

    public Show getShow() {

        return show;

    }

    public void setShow(Show show) {

        this.show = show;

    }

}

Screen.java

package com.cg.mts.pojo;

import java.util.List;

import javax.persistence.\*;

import com.fasterxml.jackson.annotation.JsonIgnore;

/\*\*

 \*

 \* @author Thejesh

 \* @category ScreenPojo

 \*

 \*/

@Entity

public class Screen {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int screenId;

    @JsonIgnore

    @ManyToOne

    private Theatre theatre;

    @OneToMany(mappedBy = "screen",cascade = CascadeType.ALL)

    private List<Show> show;

    private String screenName;

    @Column(name = "rowss")

    private int rows;

    @Column(name = "columnss")

    private int columns;

    public Screen() {

    }

    public Screen(Theatre theatre, List<Show> show, String screenName, int rows, int columns) {

        super();

        this.theatre = theatre;

        this.show = show;

        this.screenName = screenName;

        this.rows = rows;

        this.columns = columns;

    }

    public int getScreenId() {

        return screenId;

    }

    public void setScreenId(int screenId) {

        this.screenId = screenId;

    }

    public Theatre getTheatre() {

        return theatre;

    }

    public void setTheatre(Theatre theatre) {

        this.theatre = theatre;

    }

    public List<Show> getShow() {

        return show;

    }

    public void setShow(List<Show> show) {

        this.show = show;

    }

    public String getScreenName() {

        return screenName;

    }

    public void setScreenName(String screenName) {

        this.screenName = screenName;

    }

    public int getRows() {

        return rows;

    }

    public void setRows(int rows) {

        this.rows = rows;

    }

    public int getColumns() {

        return columns;

    }

    public void setColumns(int columns) {

        this.columns = columns;

    }

}

Seat.java

package com.cg.mts.pojo;

import javax.persistence.\*;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

public class Seat {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int seatId;

    private String seatNumber;

    private String type;

    private double price;

    @Enumerated(EnumType.STRING)

    private SeatStatus status;

    @JsonIgnore

    @ManyToOne

    private Ticket ticket;

    public Seat() {

    }

    public Seat(String seatNumber, String type, double price, SeatStatus status, Ticket tickett) {

        super();

        this.seatNumber = seatNumber;

        this.type = type;

        this.price = price;

        this.status = status;

        this.ticket = tickett;

    }

    public int getSeatId() {

        return seatId;

    }

    public void setSeatId(int seatId) {

        this.seatId = seatId;

    }

    public String getSeatNumber() {

        return seatNumber;

    }

    public void setSeatNumber(String seatNumber) {

        this.seatNumber = seatNumber;

    }

    public String getType() {

        return type;

    }

    public void setType(String type) {

        this.type = type;

    }

    public double getPrice() {

        return price;

    }

    public void setPrice(double price) {

        this.price = price;

    }

    public SeatStatus getStatus() {

        return status;

    }

    public void setStatus(SeatStatus status) {

        this.status = status;

    }

    public Ticket getTickett() {

        return ticket;

    }

    public void setTickett(Ticket tickett) {

        this.ticket = tickett;

    }

}

SeatStatus.java

package com.cg.mts.pojo;

public enum SeatStatus {

    AVAILABLE("Available"), BLOCKED("Blocked"), BOOKED("Booked"), CANCELLED("Cancelled");

    private String status;

    private SeatStatus(String status) {

        this.status = status;

    }

    public String getStatus() {

        return status;

    }

    public void setStatus(String status) {

        this.status = status;

    }

}

Show.java

package com.cg.mts.pojo;

import javax.persistence.\*;

import org.hibernate.annotations.DynamicUpdate;

import org.springframework.format.annotation.DateTimeFormat;

import com.fasterxml.jackson.annotation.JsonFormat;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import com.fasterxml.jackson.datatype.jsr310.deser.LocalDateTimeDeserializer;

import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;

import com.fasterxml.jackson.datatype.jsr310.deser.LocalDateDeserializer;

import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateSerializer;

import java.time.\*;

@Entity

@Table(name = "shhow")

@DynamicUpdate

public class Show {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int showId;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)

    @JsonSerialize(using = LocalDateTimeSerializer.class)

    @JsonDeserialize(using = LocalDateTimeDeserializer.class)

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm")

    private LocalDateTime showStartTime;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)

    @JsonSerialize(using = LocalDateTimeSerializer.class)

    @JsonDeserialize(using = LocalDateTimeDeserializer.class)

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm")

    private LocalDateTime showEndTime;

    private String showName;

    @OneToOne(mappedBy = "show")

    private Movie movie;

    @JsonIgnore

    @ManyToOne

    private Screen screen;

    @JsonIgnore

    @ManyToOne

    private Theatre theatre;

    @JsonIgnore

    @OneToOne

    private Booking booking;

    @JsonDeserialize(using = LocalDateDeserializer.class)

    @JsonSerialize(using = LocalDateSerializer.class)

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")

    private LocalDate showDate;

    public Show() {

    }

    public Show(LocalDateTime showStartTime, LocalDateTime showEndTime, String showName, Movie movie, Screen screen,

            Theatre theatre, Booking booking, LocalDate showDate) {

        super();

        this.showStartTime = showStartTime;

        this.showEndTime = showEndTime;

        this.showName = showName;

        this.movie = movie;

        this.screen = screen;

        this.theatre = theatre;

        this.booking = booking;

        this.showDate = showDate;

    }

    public int getShowId() {

        return showId;

    }

    public void setShowId(int showId) {

        this.showId = showId;

    }

    public LocalDateTime getShowStartTime() {

        return showStartTime;

    }

    public void setShowStartTime(LocalDateTime showStartTime) {

        this.showStartTime = showStartTime;

    }

    public LocalDateTime getShowEndTime() {

        return showEndTime;

    }

    public void setShowEndTime(LocalDateTime showEndTime) {

        this.showEndTime = showEndTime;

    }

    public String getShowName() {

        return showName;

    }

    public void setShowName(String showName) {

        this.showName = showName;

    }

    public Movie getMovie() {

        return movie;

    }

    public void setMovie(Movie movie) {

        this.movie = movie;

    }

    public Screen getScreen() {

        return screen;

    }

    public void setScreen(Screen screen) {

        this.screen = screen;

    }

    public Theatre getTheatre() {

        return theatre;

    }

    public void setTheatre(Theatre theatre) {

        this.theatre = theatre;

    }

    public Booking getBooking() {

        return booking;

    }

    public void setBooking(Booking booking) {

        this.booking = booking;

    }

    public LocalDate getShowDate() {

        return showDate;

    }

    public void setShowDate(LocalDate showDate) {

        this.showDate = showDate;

    }

}

Theatre.java

package com.cg.mts.pojo;

import java.util.List;

import javax.persistence.\*;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

@Table(name = "theatre")

public class Theatre {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int theatreId;

    private String theatreName;

    private String theatreCity;

    private String managerName;

    private String managerContact;

    @OneToMany(mappedBy = "theatre",cascade = CascadeType.ALL)

    private List<Screen> screen;

    @JsonIgnore

    @OneToMany(mappedBy = "theatre",cascade = CascadeType.ALL)

    private List<Show> show;

    public Theatre() {

    }

    public Theatre(String theatreName, String theatreCity, String managerName, String managerContact,

            List<Screen> screen, List<Show> show) {

        super();

        this.theatreName = theatreName;

        this.theatreCity = theatreCity;

        this.managerName = managerName;

        this.managerContact = managerContact;

        this.screen = screen;

        this.show = show;

    }

    public int getTheatreId() {

        return theatreId;

    }

    public void setTheatreId(int theatreId) {

        this.theatreId = theatreId;

    }

    public String getTheatreName() {

        return theatreName;

    }

    public void setTheatreName(String theatreName) {

        this.theatreName = theatreName;

    }

    public String getTheatreCity() {

        return theatreCity;

    }

    public void setTheatreCity(String theatreCity) {

        this.theatreCity = theatreCity;

    }

    public String getManagerName() {

        return managerName;

    }

    public void setManagerName(String managerName) {

        this.managerName = managerName;

    }

    public String getManagerContact() {

        return managerContact;

    }

    public void setManagerContact(String managerContact) {

        this.managerContact = managerContact;

    }

    public List<Screen> getScreen() {

        return screen;

    }

    public void setScreen(List<Screen> screen) {

        this.screen = screen;

    }

    public List<Show> getShow() {

        return show;

    }

    public void setShow(List<Show> show) {

        this.show = show;

    }

}

Ticket.java

package com.cg.mts.pojo;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.OneToOne;

@Entity

public class Ticket {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int ticketId;

    private int noOfSeats;

    private boolean ticketStatus;

    //@OneToMany(mappedBy = "ticket")

    @OneToMany

    private List<Seat> seats;

    @OneToOne

    private Booking booking;

    public Ticket() {

    }

    public Ticket(int noOfSeats, boolean ticketStatus, List<Seat> seats, Booking booking) {

        super();

        this.noOfSeats = noOfSeats;

        this.ticketStatus = ticketStatus;

        this.seats = seats;

        this.booking = booking;

    }

    public int getTicketId() {

        return ticketId;

    }

    public void setTicketId(int ticketId) {

        this.ticketId = ticketId;

    }

    public int getNoOfSeats() {

        return noOfSeats;

    }

    public void setNoOfSeats(int noOfSeats) {

        this.noOfSeats = noOfSeats;

    }

    public boolean isTicketStatus() {

        return ticketStatus;

    }

    public void setTicketStatus(boolean ticketStatus) {

        this.ticketStatus = ticketStatus;

    }

    public List<Seat> getSeats() {

        return seats;

    }

    public void setSeats(List<Seat> seats) {

        this.seats = seats;

    }

    public Booking getBooking() {

        return booking;

    }

    public void setBooking(Booking booking) {

        this.booking = booking;

    }

}

User.java

package com.cg.mts.pojo;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import org.hibernate.annotations.DynamicUpdate;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity

@DynamicUpdate

public class User {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int userid;

    private String username;

    private String password;

    private String role;

    @OneToOne

    private Customer customer;

    User() {

    }

    public User(String username, String password, String role, Customer customer) {

        super();

        this.username = username;

        this.password = password;

        this.role = role;

        this.customer = customer;

    }

    public int getUserid() {

        return userid;

    }

    public void setUserid(int userid) {

        this.userid = userid;

    }

    public String getUsername() {

        return username;

    }

    public void setUsername(String username) {

        this.username = username;

    }

    public String getPassword() {

        return password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    public String getRole() {

        return role;

    }

    public void setRole(String role) {

        this.role = role;

    }

    public Customer getCustomer() {

        return customer;

    }

    public void setCustomer(Customer customer) {

        this.customer = customer;

    }

}

QueryClass.java

package com.cg.mts.repoImpl;

import java.util.List;

import javax.persistence.\*;

import org.springframework.stereotype.Repository;

import com.cg.mts.exception.UserNotFoundException;

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.User;

@Repository

public class QueryClass {

    @PersistenceContext

    EntityManager eManager;

    public List<Booking> getAllByMovieId(int movieId) {

        TypedQuery<Booking> qry = eManager

                .createQuery("select b from Booking b join b.show s where s.movie.movieId = :id", Booking.class);

        qry.setParameter("id", movieId);

        return qry.getResultList();

    }

    public User findByUserName(String username) throws UserNotFoundException {

        TypedQuery<User> qry = eManager.createQuery("select u from User u where u.username like :name", User.class);

        qry.setParameter("name", username);

        List<User> user = qry.getResultList();

        if (user.size() == 0)

            throw new UserNotFoundException("User Not Available !!");

        return user.get(0);

    }

}

BookingRepository.java

package com.cg.mts.repository;

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

import org.springframework.stereotype.Repository;

import com.cg.mts.pojo.Booking;

@Repository

public interface BookingRepository extends JpaRepository<Booking, Integer> {

}

CustomerRepository.java

package com.cg.mts.repository;

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

import org.springframework.stereotype.Repository;

import com.cg.mts.pojo.Customer;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    boolean existsByMobileNumber(String mobileNumber);

    boolean existsByEmail(String email);

}

IAdminRepository.java

package com.cg.mts.repository;

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

import com.cg.mts.pojo.User;

public interface IAdminRepository extends JpaRepository<User, Integer> {

}

MoviesRepository.java

package com.cg.mts.repository;

import java.time.LocalDate;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.cg.mts.pojo.Movie;

@Repository

public interface MoviesRepository extends JpaRepository<Movie, Integer> {

    // @Query("select m from Movie m join m.show s where s.theatre.theatreId = :id")

    // List<Movie> getAllByTheatreId(@Param("id") int id);

    List<Movie> getAllBymovieDate(LocalDate date);

}

ScreenRepository.java

package com.cg.mts.repository;

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

import com.cg.mts.pojo.Screen;

/\*\*

 \*

 \* @author Thejesh

 \* @category ScreenRepository

 \*

 \*/

public interface ScreenRepository extends JpaRepository<Screen, Integer> {

}

SeatRepository.java

package com.cg.mts.repository;

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

import org.springframework.stereotype.Repository;

import com.cg.mts.pojo.Seat;

@Repository

public interface SeatRepository extends JpaRepository<Seat, Integer> {

}

ShowRepository.java

package com.cg.mts.repository;

import java.util.List;

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

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

import org.springframework.data.repository.query.Param;

import com.cg.mts.pojo.Show;

public interface ShowRepository extends JpaRepository<Show, Integer> {

    @Query("select s from Show s where s.theatre.theatreId = :id")

    List<Show> getAllByTheatreId(@Param("id") int id);

}

TheatreRepository.java

package com.cg.mts.repository;

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

import com.cg.mts.pojo.Theatre;

public interface TheatreRepository extends JpaRepository<Theatre, Integer> {

}

TicketRepository.java

package com.cg.mts.repository;

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

import com.cg.mts.pojo.Ticket;

public interface TicketRepository extends JpaRepository<Ticket, Integer> {

}

UserRepository.java

package com.cg.mts.repository;

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

import com.cg.mts.pojo.User;

public interface UserRepository extends JpaRepository<User, Integer> {

}

BookingServieceImpl.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

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

import org.springframework.stereotype.Service;

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.Customer;

import com.cg.mts.pojo.Seat;

import com.cg.mts.pojo.Show;

import com.cg.mts.pojo.Ticket;

import com.cg.mts.repoImpl.QueryClass;

import com.cg.mts.exception.BookingNotFoundException;

import com.cg.mts.repository.BookingRepository;

import com.cg.mts.repository.CustomerRepository;

import com.cg.mts.repository.MoviesRepository;

import com.cg.mts.repository.ShowRepository;

import com.cg.mts.repository.TicketRepository;

@Service

public class BookingServiceImpl implements IBookingService {

    @Autowired

    private BookingRepository bookingRepository;

    @Autowired

    MoviesRepository moviesRepository;

    @Autowired

    ShowRepository showRepository;

    @Autowired

    CustomerRepository custoRepository;

    @Autowired

    TicketRepository ticketRepository;

    @Autowired

    QueryClass query;

    @Override

    public Booking addBooking(Booking booking, Integer customerId, Integer showId) throws BookingNotFoundException {

        Customer customer = new Customer();

        Show show=new Show();

        /\*

         \* if (bookingRepository.existsById(booking.getTransactionId())) { throw new

         \* BookingNotFoundException("Booking with this transcationId is already done");

         \* } else {}

         \*/

        if(showId!=null) {

                //customer = custoRepository.getOne(customerId);

                show=showRepository.findById(showId).get();

                show.setBooking(booking);

                //booking.setCustomer(customer);

                booking.setShow(show);

        }

            bookingRepository.saveAndFlush(booking);

            showRepository.saveAndFlush(show);

        return bookingRepository.findById(booking.getTransactionId()).get();

    }

    @Override

    public Booking updateBooking(Booking booking) throws BookingNotFoundException {

        bookingRepository.saveAndFlush(booking);

        return bookingRepository.getOne(booking.getTransactionId());

    }

    @Override

    public Booking cancelBooking(int bookingid) throws BookingNotFoundException {

        Booking b = bookingRepository.getOne(bookingid);

        bookingRepository.delete(b);

        return b;

    }

    @Override

    public List<Booking> showAllBookings(int movieid) throws BookingNotFoundException {

        List<Booking> bk = query.getAllByMovieId(movieid);

        /\*

         \* if (bk.size() == 0) throw new BookingNotFoundException("No bookings found");

         \*/

        return bk;

    }

    @Override

    public List<Booking> showAllBookings(LocalDate bookingdate) throws BookingNotFoundException {

        List<Booking> bkList = new ArrayList<>();

        for (Booking booking : bookingRepository.findAll()) {

            if (booking.getBookingDate() != null && booking.getBookingDate().isEqual(bookingdate)) {

                bkList.add(booking);

            }

        }

        if (bkList.size() == 0)

            throw new BookingNotFoundException("No bookings found");

        else {

            return bkList;

        }

    }

    @Override

    public double calculateTotalCost(int bookingid) {

        List<Ticket> tickets = ticketRepository.findAll();

        Set<Seat> Seats = new HashSet<>();

        for (Ticket ticket : tickets) {

            if (bookingid == ticket.getBooking().getTransactionId()) {

                Seats.addAll(ticket.getSeats());

            }

        }

        double amount = 0;

        for (Seat seat : Seats) {

            amount = amount + seat.getPrice();

        }

        Booking booking = bookingRepository.getOne(bookingid);

        booking.setTotalCost(amount);

        bookingRepository.saveAndFlush(booking);

        return amount;

    }

    @Override

    public List<Booking> viewBookingList() throws BookingNotFoundException {

        List<Booking> bk = bookingRepository.findAll();

        /\*

         \* if (bk.size() == 0) throw new BookingNotFoundException("No bookings found");

         \*/

        return bk;

    }

    @Override

    public Booking viewBooking(int bookingid) throws BookingNotFoundException {

        return bookingRepository.findById(bookingid).get();

    }

}

CustomerServiece.java

package com.cg.mts.service;

import java.util.List;

import com.cg.mts.exception.CustomerNotFoundException;

import com.cg.mts.pojo.Customer;

public interface CustomerService {

    boolean existsById(Integer userId);

    boolean existsByMobileNumber(String mobileNo);

    boolean existsByEmail(String email);

    public Customer addCustomer(Customer customer) throws CustomerNotFoundException;

    public Customer removeCustomer(int customerid);

    public Customer updateCustomer(Customer customer) throws CustomerNotFoundException;

    public Customer viewCustomer(int customerid);

    public List<Customer> viewCustomerList();

}

CustomerSrvieceImpl.java

package com.cg.mts.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.cg.mts.exception.CustomerNotFoundException;

import com.cg.mts.pojo.Customer;

import com.cg.mts.repository.CustomerRepository;

@Service

public class CustomerServiceImpl implements CustomerService {

    private CustomerRepository customerRepository;

    public CustomerServiceImpl(CustomerRepository customerRepository) {

        this.customerRepository = customerRepository;

    }

    @Override

    public Customer addCustomer(Customer customer) throws CustomerNotFoundException {

        if (customer != null) {

            if (customerRepository.existsById(customer.getCustomerId())) {

                throw new CustomerNotFoundException("Customer " + customer.getCustomerId() + " is already Exists");

            } else if (customerRepository.existsByMobileNumber(customer.getMobileNumber())) {

                throw new CustomerNotFoundException(

                        "Customer with number " + customer.getMobileNumber() + " is already Exists");

            } else if (customerRepository.existsByEmail(customer.getEmail())) {

                throw new CustomerNotFoundException(

                        "Customer with email " + customer.getEmail() + " is already Exists");

            } else {

                customerRepository.saveAndFlush(customer);

            }

        }

        return customerRepository.getOne(customer.getCustomerId());

    }

    @Override

    public List<Customer> viewCustomerList() {

        return customerRepository.findAll();

    }

    @Override

    public Customer removeCustomer(int customerid) {

        Customer c = customerRepository.getOne(customerid);

        customerRepository.delete(c);

        return c;

    }

    @Override

    public Customer updateCustomer(Customer customer) throws CustomerNotFoundException {

        /\*

         \* Customer cu=customerRepository.getOne(customer.getCustomerId());

         \* System.out.println(cu.getCustomerName());

         \* System.out.println(cu.getPassword());

         \*/

        System.out.println(customer.getCustomerId());

        customerRepository.saveAndFlush(customer);

        return customerRepository.getOne(customer.getCustomerId());

    }

    @Override

    public Customer viewCustomer(int customerid) {

        return customerRepository.findById(customerid).get();

    }

    @Override

    public boolean existsById(Integer userId) {

        return customerRepository.existsById(userId);

    }

    @Override

    public boolean existsByMobileNumber(String mobileNo) {

        // TODO Auto-generated method stub

        return customerRepository.existsByMobileNumber(mobileNo);

    }

    @Override

    public boolean existsByEmail(String email) {

        // TODO Auto-generated method stub

        return customerRepository.existsByEmail(email);

    }

}

IAdminServiece.java

package com.cg.mts.service;

import org.springframework.stereotype.Service;

//import java.util.List;

//import java.util.Set;

//

//import com.cg.healthcare.entities.Appointment;

//import com.cg.healthcare.entities.DiagnosticCenter;

//import com.cg.healthcare.entities.DiagnosticTest;

//import com.cg.healthcare.entities.Patient;

//import com.cg.healthcare.entities.User;

@Service

public interface IAdminService {

    public void registerAdmin(String username, String password) throws Exception;

}

IAdminServieceImpl.java

package com.cg.mts.service;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.UserCreationError;

import com.cg.mts.pojo.User;

import com.cg.mts.repository.IAdminRepository;

import com.cg.mts.validator.InputValidator;

@Service

public class IAdminServiceImpl implements IAdminService {

    @Autowired

    IAdminRepository adminRepo;

    @Autowired

    InputValidator validate;

    @Override

    public void registerAdmin(String username, String password) throws Exception {

        if (!validate.usernameValidator(username))

            throw new UserCreationError("Check Username !!!!");

        if (!validate.passwordValidator(password))

            throw new UserCreationError("Cannot register this admin with this password");

        User u = new User(username, password, "ADMIN", null);

        adminRepo.saveAndFlush(u);

    }

}

IBookingServiece.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.List;

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.Screen;

import com.cg.mts.exception.BookingNotFoundException;

import com.cg.mts.exception.ScreenNotFoundException;

public interface IBookingService {

    public Booking addBooking(Booking booking, Integer customerId,Integer showId) throws BookingNotFoundException;

    public List<Booking> viewBookingList() throws BookingNotFoundException;

    public Booking updateBooking(Booking booking) throws BookingNotFoundException;

    public Booking cancelBooking(int bookingid) throws BookingNotFoundException;

    public List<Booking> showAllBookings(int movieid) throws BookingNotFoundException;

    public Booking viewBooking(int bookingid) throws BookingNotFoundException;

    public List<Booking> showAllBookings(LocalDate bookingdate) throws BookingNotFoundException;

    public double calculateTotalCost(int bookingid);

}

ISeatServiece.java

package com.cg.mts.service;

import java.util.List;

import com.cg.mts.exception.SeatNotFoundException;

import com.cg.mts.pojo.Seat;

public interface ISeatService {

    public Seat addSeat(Seat seat) throws SeatNotFoundException;

    public List<Seat> viewSeatList() throws SeatNotFoundException;

    public Seat updateSeat(Seat seat);

    public Seat bookSeat(Seat seat);

    public Seat cancelSeatBooking(Seat seat);

    public Seat blockSeat(Seat seat); // not available for any booking

}

IUserService.java

package com.cg.mts.service;

import com.cg.mts.exception.UserCreationError;

import com.cg.mts.pojo.User;

public interface IUserService {

    public User addUser(User user) throws UserCreationError;

    public User removeUser(User user);

}

IUserServieceImpl.java

package com.cg.mts.service;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.UserCreationError;

import com.cg.mts.pojo.User;

import com.cg.mts.repoImpl.QueryClass;

import com.cg.mts.repository.UserRepository;

import com.cg.mts.validator.InputValidator;

@Service

public class IUserServiceImpl implements IUserService {

    @Autowired

    UserRepository userrepo;

    @Autowired

    InputValidator validate;

    @Autowired

    QueryClass qcp;

    @Override

    public User addUser(User user) throws UserCreationError {

        if (!validate.usernameValidator(user.getUsername()))

            throw new UserCreationError("Check Username !!!!");

        if (!validate.passwordValidator(user.getPassword()))

            throw new UserCreationError("Cannot register this User with this password");

        return userrepo.saveAndFlush(user);

    }

    @Override

    public User removeUser(User user) {

        userrepo.delete(user);

        return user;

    }

}

LoginServiece.java

package com.cg.mts.service;

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

import org.springframework.stereotype.Service;

import com.cg.mts.pojo.Login;

import com.cg.mts.pojo.User;

import com.cg.mts.repoImpl.QueryClass;

@Service

public class LoginService {

    @Autowired

    private QueryClass qcp;

    private Login logData = new Login();

    public Login loginWithData(String username, String password) throws Exception {

        User user = qcp.findByUserName(username);

        if (!user.getPassword().equals(password))

            throw new Exception("Login Data Invalid");

        logData.setLoginStatus(true);

        logData.setUser(user);

        return logData;

    }

    public Login logoutPresentUser() {

        if (logData.isLoginStatus()) {

            logData.setLoginStatus(false);

        }

        return logData;

    }

    public boolean loginStatus() {

        return logData.isLoginStatus();

    }

    public String getRole() {

        return logData.getRole();

    }

}

MoviesServies.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.List;

import org.springframework.web.multipart.MultipartFile;

import com.cg.mts.exception.MovieNotFoundException;

import com.cg.mts.pojo.Movie;

public interface MoviesService {

    public Movie addMovie(Movie movie) throws MovieNotFoundException;

    public Movie removeMovie(int movieid) throws MovieNotFoundException;

    public Movie updateMovie(Movie movie) throws MovieNotFoundException;

    public Movie addMovieToShow(Movie movie, Integer showId) throws MovieNotFoundException;

    public Movie viewMovie(int movieid) throws MovieNotFoundException;

    public List<Movie> viewMovieList() throws MovieNotFoundException;

    public List<Movie> viewMovieList(int theatreid);

    public List<Movie> viewMovieList(LocalDate date);

}

MoviesServieceImpl.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

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

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import com.cg.mts.exception.MovieNotFoundException;

import com.cg.mts.pojo.Movie;

import com.cg.mts.pojo.Show;

import com.cg.mts.repoImpl.QueryClass;

import com.cg.mts.repository.MoviesRepository;

import com.cg.mts.repository.ShowRepository;

import com.cg.mts.repository.TheatreRepository;

import org.springframework.util.StringUtils;

import java.io.IOException;

import java.util.Base64;

@Service

public class MoviesServiceImpl implements MoviesService {

    @Autowired

    private MoviesRepository moviesrepository;

    @Autowired

    TheatreRepository theatreRepository;

    @Autowired

    ShowRepository showrepository;

    @Autowired

    QueryClass query;

    @Override

    public Movie addMovie(Movie movie) throws MovieNotFoundException {

        if (movie != null) {

            if (moviesrepository.existsById(movie.getMovieId())) {

                throw new MovieNotFoundException("Movie with this id already exists");

            } else {

                /\*

                String fileName = StringUtils.cleanPath(file.getOriginalFilename());

                if(fileName.contains(".."))

                {

                    System.out.println("not a a valid file");

                }

                try {

                    movie.setMovieImage(Base64.getEncoder().encodeToString(file.getBytes()));

                } catch (IOException e) {

                    e.printStackTrace();

                }\*/

                moviesrepository.saveAndFlush(movie);

            }

        }

        return movie;

    }

    @Override

    public Movie removeMovie(int movieid) {

        Movie m = moviesrepository.findById(movieid).get();

        List<Show> shows = showrepository.findAll();

        for (Show show : shows) {

            if (show.getMovie()!=null && show.getMovie().getMovieId() == movieid) {

                show.setMovie(null);

            }

        }

        moviesrepository.delete(m);

        return m;

    }

    @Override

    public Movie updateMovie(Movie movie) {

        moviesrepository.saveAndFlush(movie);

        return moviesrepository.getOne(movie.getMovieId());

    }

    @Override

    public Movie addMovieToShow(Movie movie,Integer showId) {

        Show show=new Show();

        if (showId != null) {

            show = showrepository.getOne(showId);

            movie.setShow(show);

        }

        moviesrepository.saveAndFlush(movie);

        return moviesrepository.getOne(movie.getMovieId());

    }

    @Override

    public Movie viewMovie(int movieid) {

        return moviesrepository.findById(movieid).get();

    }

    @Override

    public List<Movie> viewMovieList() throws MovieNotFoundException {

        List<Movie> ml = moviesrepository.findAll();

        //if (ml.size() == 0) throw new MovieNotFoundException("Movies dosen't exist");

        return ml;

    }

    @Override

    public List<Movie> viewMovieList(int theatreid) {

        List<Movie> movies = new ArrayList<>();

        List<Show> shows = showrepository.findAll();

        Set<Integer> showIds = new HashSet<>();

        for (Show s : shows) {

            if (s.getTheatre().getTheatreId() == theatreid) {

                showIds.add(s.getShowId());

            }

        }

        for (Integer id : showIds) {

            movies.add(showrepository.getOne(id).getMovie());

        }

        return movies;

    }

    @Override

    public List<Movie> viewMovieList(LocalDate date) {

        List<Movie> mvList = new ArrayList<>();

        for (Movie movie : moviesrepository.findAll()) {

            if (movie.getMovieDate() != null && movie.getMovieDate().isEqual(date)) {

                mvList.add(movie);

            }

        }

        return mvList;

    }

}

ScreenServiece.java

package com.cg.mts.service;

import java.util.List;

import com.cg.mts.exception.ScreenNotFoundException;

import com.cg.mts.pojo.Screen;

import com.cg.mts.pojo.Theatre;

/\*\*

 \*

 \* @author Thejesh

 \* @category Screen Service

 \*

 \*/

public interface ScreenService {

    public Screen addScreen(Screen screen, Integer theatreId) throws ScreenNotFoundException;

    public List<Screen> viewScreenList() throws ScreenNotFoundException;

    public Screen updateScreen(Screen s, Integer theatreId);

    public Screen viewScreen(int screenId) throws ScreenNotFoundException;

    public Theatre getTheatre(int screenId) throws ScreenNotFoundException;

}

ScreenServieceImpl.java

package com.cg.mts.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.ScreenNotFoundException;

import com.cg.mts.pojo.Screen;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.repository.ScreenRepository;

import com.cg.mts.repository.TheatreRepository;

/\*\*

 \*

 \* @author Thejesh

 \* @category Screen Service Implementation

 \*/

@Service

public class ScreenServiceImpl implements ScreenService {

    @Autowired

    private ScreenRepository screenRepository;

    @Autowired

    private TheatreRepository theatreRepository;

    /\*\*

     \* @return screenList

     \*/

    @Override

    public List<Screen> viewScreenList() throws ScreenNotFoundException {

        List<Screen> screen = screenRepository.findAll();

        if (screen.size() == 0)

            throw new ScreenNotFoundException("No screens found");

        return screen;

    }

    /\*\*

     \* @return screen

     \*/

    @Override

    public Screen addScreen(Screen screen, Integer theatreId) throws ScreenNotFoundException {

        Theatre theatre = new Theatre();

        if (theatreId != null) {

            if (screenRepository.existsById(screen.getScreenId())) {

                throw new ScreenNotFoundException("Screen already exists");

            } else {

                theatre = theatreRepository.getOne(theatreId);

                screen.setTheatre(theatre);

            }

            screenRepository.saveAndFlush(screen);

        }

        return screen;

    }

    @Override

    public Screen viewScreen(int screenId) throws ScreenNotFoundException {

        return screenRepository.findById(screenId).get();

        }

    /\*\*

     \* @return updatedScreen

     \*/

    @Override

    public Screen updateScreen(Screen screen, Integer theatreId) {

        Theatre theatre = new Theatre();

        if (theatreId != null) {

            theatre = theatreRepository.getOne(theatreId);

            screen.setTheatre(theatre);

        }

        screenRepository.saveAndFlush(screen);

        return screen;

    }

    @Override

    public Theatre getTheatre(int screenId) throws ScreenNotFoundException {

        Screen screen =screenRepository.findById(screenId).get();

        Theatre theatre=screen.getTheatre();

        return theatre;

    }

}

SeatServieceImpl.java

package com.cg.mts.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.SeatNotFoundException;

import com.cg.mts.pojo.Seat;

import com.cg.mts.pojo.SeatStatus;

import com.cg.mts.repository.SeatRepository;

@Service

public class SeatServiceImpl implements ISeatService {

    @Autowired

    private SeatRepository seatRepository;

    @Override

    public Seat addSeat(Seat seat) throws SeatNotFoundException {

        if (seat != null) {

            if (seatRepository.existsById(seat.getSeatId())) {

                throw new SeatNotFoundException("Seat with this id already exists");

            } else {

                seatRepository.saveAndFlush(seat);

            }

        }

        return seatRepository.getOne(seat.getSeatId());

    }

    @Override

    public List<Seat> viewSeatList() throws SeatNotFoundException {

        List<Seat> li = seatRepository.findAll();

        /\*

         \* if (li.size() == 0) throw new SeatNotFoundException("No seats found");

         \*/

        return li;

    }

    @Override

    public Seat updateSeat(Seat seat) {

        // TODO Auto-generated method stub

        return seatRepository.saveAndFlush(seat);

    }

    @Override

    public Seat bookSeat(Seat seat) {

        seat.setStatus(SeatStatus.BOOKED);

        return seatRepository.saveAndFlush(seat);

    }

    @Override

    public Seat cancelSeatBooking(Seat seat) {

        seat.setStatus(SeatStatus.CANCELLED);

        return seatRepository.saveAndFlush(seat);

    }

    @Override

    public Seat blockSeat(Seat seat) {

        seat.setStatus(SeatStatus.BLOCKED);

        return seatRepository.saveAndFlush(seat);

    }

}

ShowServiece.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.List;

import com.cg.mts.pojo.Show;

public interface ShowService {

    public Show addShow(Show show, Integer theatreId, Integer screenId);

    public Show updateShow(Show show, Integer theatreId, Integer screenId);

    public Show removeShow(int showid);

    public Show viewShow(int showid);

    public List<Show> viewAllShows();

    public List<Show> viewShowList(int theatreid);

    public List<Show> viewShowList(LocalDate date);

}

ShowServieceImpl.java

package com.cg.mts.service;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cg.mts.pojo.Screen;

import com.cg.mts.pojo.Show;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.repository.ScreenRepository;

import com.cg.mts.repository.ShowRepository;

import com.cg.mts.repository.TheatreRepository;

@Service

public class ShowServiceImpl implements ShowService {

    @Autowired

    private ShowRepository showrepository;

    @Autowired

    private TheatreRepository theatreRepository;

    @Autowired

    private ScreenRepository screenRepository;

    @Override

    public Show addShow(Show show, Integer theatreId, Integer screenId) {

        Theatre theatre = new Theatre();

        Screen screen = new Screen();

        if (theatreId != null) {

            theatre = theatreRepository.getOne(theatreId);

            show.setTheatre(theatre);

        }

        if (screenId != null) {

            screen = screenRepository.getOne(screenId);

            show.setScreen(screen);

        }

        showrepository.saveAndFlush(show);

        return show;

    }

    @Override

    public Show updateShow(Show show, Integer theatreId, Integer screenId) {

        Theatre theatre = new Theatre();

        Screen screen = new Screen();

        if (theatreId != null) {

            theatre = theatreRepository.getOne(theatreId);

            show.setTheatre(theatre);

        }

        if (screenId != null) {

            screen = screenRepository.getOne(screenId);

            show.setScreen(screen);

        }

        showrepository.saveAndFlush(show);

        return show;

    }

    @Override

    public Show removeShow(int showid) {

        Show s = showrepository.findById(showid).get();

        showrepository.delete(s);

        return s;

    }

    @Override

    public Show viewShow(int showid) {

        return showrepository.findById(showid).get();

    }

    @Override

    public List<Show> viewAllShows() {

        return showrepository.findAll();

    }

    @Override

    public List<Show> viewShowList(int theatreid) {

        return showrepository.getAllByTheatreId(theatreid);

        // return null;

    }

    @Override

    public List<Show> viewShowList(LocalDate date) {

        List<Show> shList = new ArrayList<>();

        for (Show show : showrepository.findAll()) {

            if (show.getShowDate() != null && show.getShowDate().isEqual(date)) {

                shList.add(show);

            }

        }

        return shList;

    }

}

TheatreServiece.java

package com.cg.mts.service;

import com.cg.mts.exception.TheatreNotFoundException;

import com.cg.mts.pojo.Theatre;

import java.util.List;

public interface TheatreService {

    public List<Theatre> getAllTheatres() throws TheatreNotFoundException;

    public Theatre findTheatres(int theatreId);

    public Theatre addTheatre(Theatre t) throws TheatreNotFoundException;

    public List<Theatre> updateTheatre(Theatre t);

    public List<Theatre> deleteTheatreById(int theatreId);

    public List<Theatre> findTheatresByMovie(Integer movieId) throws TheatreNotFoundException;

}

ThereatureServieceImpl.java

package com.cg.mts.service;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.TheatreNotFoundException;

import com.cg.mts.pojo.Movie;

import com.cg.mts.pojo.Show;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.repository.MoviesRepository;

import com.cg.mts.repository.ScreenRepository;

import com.cg.mts.repository.TheatreRepository;

@Service

public class TheatreServiceImpl implements TheatreService {

    @Autowired

    private TheatreRepository theatrerepository;

    @Autowired

    ScreenRepository screenRepository;

    @Autowired

    private MoviesRepository moviesrepository;

    @Override

    public List<Theatre> getAllTheatres() throws TheatreNotFoundException {

        List<Theatre> the = theatrerepository.findAll();

        //if (the.size() == 0) throw new TheatreNotFoundException("No theatres found");

        return the;

    }

    @Override

    public Theatre findTheatres(int theatreId) {

        // TODO Auto-generated method stub

        if (theatrerepository.findById(theatreId).isPresent()) {

            return theatrerepository.findById(theatreId).get();

        } else

            return null;

    }

    @Override

    public Theatre addTheatre(Theatre m) throws TheatreNotFoundException {

        if (m != null) {

            if (theatrerepository.existsById(m.getTheatreId())) {

                throw new TheatreNotFoundException("Theatre already exists");

            } else {

                theatrerepository.saveAndFlush(m);

            }

        }

        return m;

    }

    @Override

    public List<Theatre> updateTheatre(Theatre m) {

        // TODO Auto-generated method stub

        theatrerepository.saveAndFlush(m);

        return theatrerepository.findAll();

    }

    @Override

    public List<Theatre> deleteTheatreById(int theatreId) {

        // TODO Auto-generated method stub

        theatrerepository.deleteById(theatreId);

        return theatrerepository.findAll();

    }

    @Override

    public List<Theatre> findTheatresByMovie(Integer movieId) throws TheatreNotFoundException {

        List<Theatre> theatreList=new ArrayList<>();

        Movie movie=moviesrepository.findById(movieId).get();

        Integer showwID=movie.getShow().getShowId();

        List<Theatre> theatres = theatrerepository.findAll();

        for(Theatre theatre:theatres) {

            List<Show> shows =theatre.getShow();

            for(Show show:shows){

                if(show.getShowId()==showwID) {

                    theatreList.add(theatre);

                }

            }

        }

        return theatreList;

    }

    /\*

     \* @Override public Theatre addTheatre(Theatre t, List<Integer> screens) { //

     \* TODO Auto-generated method stub

     \* //if(theatrerepository.existsById(m.getTheatreId())) throws new Theatre

     \* List<Screen> preScs=new ArrayList<>(); if(screens!=null) { for(int id:

     \* screens) { Screen sc=screenRepository.getOne(id); preScs.add(sc);

     \* screenRepository.saveAndFlush(sc); } } t.setScreens(preScs);

     \* theatrerepository.saveAndFlush(t); return t; }

     \*

     \* @Override public List<Theatre> updateTheatre(Theatre t, List<Integer>

     \* screenIds) { // TODO Auto-generated method stub return null; }

     \*/

}

TicketServiece.java

package com.cg.mts.service;

import java.util.List;

import com.cg.mts.exception.TicketNotFoundException;

import com.cg.mts.pojo.Ticket;

public interface TicketService {

    public Ticket addTicket(Ticket ticket,Integer bookingId) throws TicketNotFoundException;

    public Ticket findTicket(int ticketId);

    List<Ticket> viewTicketList() throws TicketNotFoundException;

}

TicketServieceImpl.java

package com.cg.mts.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cg.mts.exception.TicketNotFoundException;

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.Seat;

import com.cg.mts.pojo.Ticket;

import com.cg.mts.repository.BookingRepository;

import com.cg.mts.repository.SeatRepository;

import com.cg.mts.repository.TicketRepository;

@Service

public class TicketServiceImpl implements TicketService {

    private TicketRepository ticketRepository;

    public TicketServiceImpl(TicketRepository ticketRepository) {

        this.ticketRepository = ticketRepository;

    }

    @Autowired

    private SeatRepository seatRepository;

    @Autowired

    private BookingRepository bookingRepository;

    @Override

    public Ticket addTicket(Ticket ticket,Integer transactionId) throws TicketNotFoundException {

        Booking booking=new Booking();

        if(transactionId!=null) {

            booking=bookingRepository.findById(transactionId).get();

            booking.setTransactionStatus("Completed");

            ticket.setBooking(booking);

        }

        ticketRepository.saveAndFlush(ticket);

        /\*

         \* for(Seat s:ticket.getSeats()) { s.setTickett(ticket);

         \* seatRepository.saveAndFlush(s); }

         \*/

        return ticket;

    }

    @Override

    public List<Ticket> viewTicketList() throws TicketNotFoundException {

        List<Ticket> ti = ticketRepository.findAll();

        if (ti.size() == 0)

            throw new TicketNotFoundException("No tickets are avaliable");

        return ti;

    }

    @Override

    public Ticket findTicket(int ticketId) {

        // TODO Auto-generated method stub

        return ticketRepository.findById(ticketId).get();

    }

}

InputValidator.java

package com.cg.mts.validator;

public interface InputValidator {

    public boolean nameValidator(String name);

    public boolean contactValidator(String contact);

    public boolean emailValidator(String email);

    public boolean passwordValidator(String password);

    public boolean usernameValidator(String username);

}

InputValidatorImpl.java

package com.cg.mts.validator;

import org.springframework.stereotype.Component;

@Component

public class InputValidationImpl implements InputValidator {

    @Override

    public boolean nameValidator(String name) {

        return name.length() >= 3;

    }

    public boolean usernameValidator(String username) {

        return username.matches("[A-Za-z]{3,20}$");

    }

    @Override

    public boolean passwordValidator(String password) {

        return password.length() >= 3;

    }

    @Override

    public boolean contactValidator(String contact) {

        return contact.matches("[0-9]{10}");

    }

    @Override

    public boolean emailValidator(String email) {

        return email.matches("^(.+)@(.+)$");

    }

}

Application.properties

server.port=8000

// spring.datasource.url=jdbc:mysql://localhost:3306/mymovieplan

spring.datasource.url=jdbc:mysql://my-movie-plan-database.cnivq90iyrhx.ap-south-1.rds.amazonaws.com:3306/mymovieplan

spring.datasource.username=root

spring.datasource.password=abcd

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

logging.level.root=INFO

logging.level.com.capg.mgs.controller=TRACE

Abstracttest.java

package com.cg.mts;

import java.io.IOException;

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

import org.springframework.test.context.web.WebAppConfiguration;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.web.context.WebApplicationContext;

import com.fasterxml.jackson.core.JsonParseException;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

@WebAppConfiguration

public abstract class AbstractTest {

    protected MockMvc mvc;

    @Autowired

    WebApplicationContext webApplicationContext;

    protected void setUp() {

        mvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();

    }

    protected String mapToJson(Object obj) throws JsonProcessingException {

        ObjectMapper objectMapper = new ObjectMapper();

        return objectMapper.writeValueAsString(obj);

    }

    protected <T> T mapFromJson(String json, Class<T> clazz)

            throws JsonParseException, JsonMappingException, IOException {

        ObjectMapper objectMapper = new ObjectMapper();

        return objectMapper.readValue(json, clazz);

    }

}

MovieFinalApplicationTests.java

package com.cg.mts;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.assertNotEquals;

import java.time.LocalDate;

import java.time.LocalDateTime;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.cg.mts.controller.LoginController;

import com.cg.mts.pojo.Booking;

import com.cg.mts.pojo.Customer;

import com.cg.mts.pojo.Movie;

import com.cg.mts.pojo.Screen;

import com.cg.mts.pojo.Seat;

import com.cg.mts.pojo.Show;

import com.cg.mts.pojo.Theatre;

import com.cg.mts.pojo.Ticket;

@SpringBootTest

class MovieFinalApplicationTests extends AbstractTest {

    @Override

    @BeforeEach

    public void setUp() {

        super.setUp();

    }

    @Autowired

    LoginController loginController;

    /\*

     \* @Test public void addCustomer() throws Exception { String uri =

     \* "/customer/add"; Customer cu=new Customer("Surya", "Anantapur", "7569369939",

     \* "surya@gmail.com", "bunny"); String inputJson = super.mapToJson(cu);

     \* System.out.println("======================="+inputJson+

     \* "======================"); MvcResult mvcResult = mvc.perform(

     \* MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE

     \* ).content(inputJson)) .andReturn(); int status =

     \* mvcResult.getResponse().getStatus(); assertEquals(201, status); }

     \*

     \*/

    /\*@Test

    public void viewCustomerList() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/customer/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Customer cu[] = super.mapFromJson(content, Customer[].class);

        assertEquals("vidhya",cu[cu.length-1].getCustomerName());

    }

    @Test

    public void addTheatre() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/theatre/insert";

        Theatre th = new Theatre("Kabuki", "Tokyo", "Vidhya", "998990181", null, null);

        String inputJson = super.mapToJson(th);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(201, status);

        String content = mvcResult.getResponse().getContentAsString();

        Theatre Th = super.mapFromJson(content, Theatre.class);

        assertEquals("Kabuki", Th.getTheatreName());

    }

    @Test

    public void findTheatres() throws Exception {

        String uri = "/theatre/find/4";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Theatre Th = super.mapFromJson(content, Theatre.class);

        assertEquals("Kabuki", Th.getTheatreName());

    }

    @Test

    public void getAlltheatres() throws Exception {

        String uri = "/theatre/all";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Theatre th[] = super.mapFromJson(content, Theatre[].class);

        assertEquals("Kabuki", th[th.length-1].getTheatreName());

    }

      @Test

      public void deleteMoviesById() throws Exception {

          loginController.loginUser("surya", "capgemini@1");

          String uri="/movies/delete/34";

          MvcResult mvcResult=mvc.perform(MockMvcRequestBuilders.delete(uri)).andReturn();

          int status=mvcResult.getResponse().getStatus();

          assertEquals(200, status);

          String content = mvcResult.getResponse().getContentAsString();

          Movie mv =super.mapFromJson(content, Movie.class);

          assertEquals("Avengers",mv.getMovieName());

      }

    @Test

    public void addMovie() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/movies/add";

        Movie mov = new Movie("Avengers", "scifi", "04", "English", "Good", uri, LocalDate.of(2007, 12, 03), null);

        String inputJson = super.mapToJson(mov);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(201, status);

        String content = mvcResult.getResponse().getContentAsString();

        Movie Mv = super.mapFromJson(content, Movie.class);

        assertEquals("Avengers", Mv.getMovieName());

    }

    @Test

    public void viewMovie() throws Exception {

        String uri = "/movies/viewMovie/5";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Movie mv = super.mapFromJson(content, Movie.class);

        assertEquals(5, mv.getMovieId());

    }

    @Test

    public void viewMovieList() throws Exception {

        String uri = "/movies/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Movie mv[] = super.mapFromJson(content, Movie[].class);

        assertEquals("Avengers", mv[mv.length-1].getMovieName());

    }

    @Test

    public void addAScreen() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/screens/add";

        Theatre th = new Theatre("Kabuki", "Tokyo", "Vidhya", "998990181", null, null);

        Screen sc = new Screen(th, null, "screen1", 4, 8);

        String inputJson = super.mapToJson(sc);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Screen scr = super.mapFromJson(content, Screen.class);

        assertEquals("screen1", scr.getScreenName());

    }

    @Test

    public void viewScreenList() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/screens/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Screen sc[] = super.mapFromJson(content, Screen[].class);

        assertEquals("screen2", sc[sc.length-1].getScreenName());

    }

    @Test

    public void addShow() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/shows/add";

        Theatre th = new Theatre("Kabuki", "Tokyo", "Vidhya", "998990181", null, null);

        Screen sc = new Screen(th, null, "screen1", 4, 8);

        Movie mov = new Movie("Avengers", "scifi", "04", "English", "Good", uri, LocalDate.of(2007, 12, 03), null);

        Show sh = new Show(LocalDateTime.of(2020, 10, 29, 7, 4, 2, 0), LocalDateTime.of(2020, 10, 29, 9, 4, 2, 0),

                "MorningShow", null, sc, th, null, LocalDate.of(2021, 04, 28));

        String inputJson = super.mapToJson(sh);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(201, status);

        String content = mvcResult.getResponse().getContentAsString();

        Show sho = super.mapFromJson(content, Show.class);

        assertEquals("MorningShow", sho.getShowName());

    }

    @Test

    public void viewShowList() throws Exception {

        String uri = "/shows/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Show sh[] = super.mapFromJson(content, Show[].class);

        assertEquals("MorningShow", sh[sh.length-1].getShowName());

    }

    @Test

    public void viewAllBookings() throws Exception {

        String uri = "/booking/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Booking bo[] = super.mapFromJson(content, Booking[].class);

        assertEquals("Online", bo[0].getTransactionMode());

    }

    @Test

    public void addASeat() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/seats/add";

        Seat st = new Seat("c12", "front", 25, null, null);

        String inputJson = super.mapToJson(st);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(201, status);

        String content = mvcResult.getResponse().getContentAsString();

        Seat seat = super.mapFromJson(content, Seat.class);

        assertEquals("c12", seat.getSeatNumber());

    }

    @Test

    public void viewSeatList() throws Exception {

        String uri = "/seats/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Seat st[] = super.mapFromJson(content, Seat[].class);

        assertEquals("c12", st[st.length-1].getSeatNumber());

    }

    @Test

    public void addATicket() throws Exception {

        loginController.loginUser("surya", "capgemini@1");

        String uri = "/tickets/add";

        Customer cu = new Customer("Surya", "Anantapur", "7569369939", "surya@gmail.com", "bunny");

        Booking boo = new Booking(null, LocalDate.of(2007, 12, 03), "Online", "pending", 100, null, null);

        Ticket tk = new Ticket(2, true, null, null);

        String inputJson = super.mapToJson(tk);

        System.out.println("=======================" + inputJson + "======================");

        MvcResult mvcResult = mvc.perform(

                MockMvcRequestBuilders.post(uri).contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson))

                .andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(201, status);

        String content = mvcResult.getResponse().getContentAsString();

        Ticket tik = super.mapFromJson(content, Ticket.class);

        assertEquals(2, tik.getNoOfSeats());

    }

    @Test

    public void viewTicketList() throws Exception {

        String uri = "/tickets/findall";

        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)).andReturn();

        int status = mvcResult.getResponse().getStatus();

        assertEquals(200, status);

        String content = mvcResult.getResponse().getContentAsString();

        Ticket tk[] = super.mapFromJson(content, Ticket[].class);

        assertEquals(2, tk[tk.length-1].getNoOfSeats());

    }

\*/

}

**FrontendCode**

**create-booking.component.html**

## <section id="booking">

## <div class="custom-container">

## <h3 style="color: red">{{ errorMessage }}</h3>

## <form [formGroup]="bookingForm" (ngSubmit)="addABooking()">

## <div class="form-row">

## <!-- <div class="form-group">

## <div class="input-data"> -->

## <!-- <input

## type="text"

## class="form-control"

## formControlName="transactionMode"

## required

## />

## <div class="underline"></div>

## <label for="">Transaction Mode</label> -->

## <div class="wrap">

## <input

## type="radio"

## id="rad1"

## value="admin"

## formControlName="transactionMode"

## />

## <label for="rad1">Online</label>

## <input

## type="radio"

## id="rad2"

## formControlName="transactionMode"

## value="offline"

## />

## <label for="rad2">Offline</label>

## <!-- </div>

## </div> -->

## </div>

## </div>

## <div class="form-group">

## <div class="form-row">

## <div class="date">

## <input

## type="date"

## class="form-control"

## formControlName="bookingDate"

## />

## </div>

## </div>

## </div>

## <div>

## <button [routerLink]="['/movie']" type="submit" class="btn btn-danger">

## Cancel

## </button>

## <button type="submit" class="btn btn-primary">

## Proceed <i class="icofont-long-arrow-right"></i>

## </button>

## </div>

## </form>

## </div>

## </section>

## ****create-booking.component.css****

## .custom-container {

## display: flex;

## align-items: center;

## justify-content: center;

## min-height: 90vh;

## background: #13151f;

## width: 100%;

## padding: 25px 40px 10px 40px;

## box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

## }

## form {

## overflow: hidden;

## background: #13151f;

## }

## .custom-container .text {

## text-align: center;

## font-size: 35px;

## -webkit-text-fill-color: transparent;

## }

## .custom-container form {

## padding: 30px 0 0 0;

## }

## .custom-container form .form-row {

## display: flex;

## margin: 32px 0;

## }

## form .form-row .input-data {

## width: 100%;

## height: 40px;

## margin: 0 20px;

## position: relative;

## }

## form .form-row .textarea {

## height: 70px;

## }

## .date {

## padding-left: 1rem;

## }

## .input-data input,

## .textarea textarea,

## .date input {

## display: block;

## width: 100%;

## height: 100%;

## border: none;

## outline: none;

## font-size: 17px;

## border-bottom: 2px solid rgba(0, 0, 0, 0.12);

## }

## .input-data input:focus ~ label,

## .textarea textarea:focus ~ label,

## .input-data input:valid ~ label,

## .textarea textarea:valid ~ label {

## transform: translateY(-20px);

## font-size: 14px;

## color: #3498db;

## }

## .textarea textarea {

## resize: none;

## padding-top: 10px;

## }

## .input-data label {

## position: absolute;

## pointer-events: none;

## bottom: 10px;

## font-size: 16px;

## transition: all 0.3s ease;

## }

## .textarea label {

## width: 100%;

## bottom: 40px;

## background: #13151f;

## }

## .input-data .underline {

## position: absolute;

## bottom: 0;

## height: 2px;

## width: 100%;

## }

## .input-data .underline:before {

## position: absolute;

## content: "";

## height: 2px;

## width: 100%;

## background: #3498db;

## transform: scaleX(0);

## transform-origin: center;

## transition: transform 0.3s ease;

## }

## .input-data input:focus ~ .underline:before,

## .input-data input:valid ~ .underline:before,

## .textarea textarea:focus ~ .underline:before,

## .textarea textarea:valid ~ .underline:before {

## transform: scale(1);

## }

## @media (max-width: 700px) {

## .custom-container .text {

## font-size: 30px;

## }

## .custom-container form {

## padding: 10px 0 0 0;

## }

## .custom-container form .form-row {

## display: block;

## }

## form .form-row .input-data {

## margin: 35px 0 !important;

## }

## }

## .wrap {

## position: absolute;

## top: 50%;

## left: 50%;

## margin-right: -50%;

## margin-bottom: 2rem;

## transform: translate(-50%, -50%);

## }

## input {

## display: none;

## }

## label {

## cursor: pointer;

## margin: 10px 10px;

## margin-right: 2rem;

## color: white;

## }

## label:before {

## content: "";

## display: inline-block;

## height: 30px;

## width: 30px;

## background: #fff;

## border-radius: 50%;

## z-index: 2;

## transition: box-shadow 0.4s ease, background 0.3s ease;

## }

## input:checked + label:before {

## box-shadow: inset 0px 3px 0 2px rgb(255, 255, 255);

## background: #13151f;

## }

## button {

## margin-left: 1rem;

## }

## .date {

## margin-left: 0.2rem;

## }

## ****create-booking.component.spec.ts****

## import { ComponentFixture, TestBed } from '@angular/core/testing';

## import { CreateBookingComponent } from './create-booking.component';

## describe('CreateBookingComponent', () => {

## let component: CreateBookingComponent;

## let fixture: ComponentFixture<CreateBookingComponent>;

## beforeEach(async () => {

## await TestBed.configureTestingModule({

## declarations: [ CreateBookingComponent ]

## })

## .compileComponents();

## });

## beforeEach(() => {

## fixture = TestBed.createComponent(CreateBookingComponent);

## component = fixture.componentInstance;

## fixture.detectChanges();

## });

## it('should create', () => {

## expect(component).toBeTruthy();

## });

## });

## ****create-booking.component.ts****

## import { Component, OnInit } from '@angular/core';

## import { FormBuilder, FormGroup, Validators } from '@angular/forms';

## import { ActivatedRoute, Router } from '@angular/router';

## import { BokingService } from '../boking.service';

## @Component({

## selector: 'app-create-booking',

## templateUrl: './create-booking.component.html',

## styleUrls: ['./create-booking.component.css'],

## })

## /\*\*

## \* @author Jyothi

## \* @Description It is used to perform movie booking by the Customer.

## \*/

## export class CreateBookingComponent implements OnInit {

## bookingForm: FormGroup;

## errorMessage: string;

## showId: number;

## bookingId: number;

## constructor(

## private fb: FormBuilder,

## private bService: BokingService,

## private router: Router,

## private actRouter: ActivatedRoute

## ) {}

## ngOnInit(): void {

## this.showId = this.actRouter.snapshot.params['showId'];

## this.bookingForm = this.fb.group({

## transactionMode: [''],

## transactionStatus: [''],

## bookingDate: [''],

## });

## }

## addABooking() {

## this.bService.addBooking(this.bookingForm.value, this.showId).subscribe(

## (res: any) => {

## console.log(res.transactionId);

## this.router.navigate(['ticket/add', res.transactionId]);

## },

## (error: any) => {

## this.errorMessage = error;

## }

## );

## }

## }

## ****detail-view-booking.component.css****

## #detailMovie {

## color: #b6b7b9;

## background: #13151f;

## position: relative;

## min-height: 90vh;

## }

## .section-title h1 i {

## color: #fece50;

## font-size: 40px;

## margin: 2rem 1rem 0 1rem;

## }

## hr {

## margin: 0;

## border-top: 2px solid #ddd;

## margin-bottom: 3rem;

## }

## .details {

## margin-top: 3rem;

## min-height: 30rem;

## padding: 35px;

## box-shadow: 0 0 20px #000000;

## position: relative;

## width: 45%;

## margin: auto;

## }

## .details .movieDetails-content img {

## display: block;

## position: relative;

## }

## .movieDetails-content > p {

## margin: 12px 0 30px;

## position: relative;

## padding-left: 70px;

## }

## .movieDetails-content > h2 + a.theme-btn {

## margin: 12px 10px 20px 0;

## }

## .movieDetails-content > p::before {

## position: absolute;

## content: "";

## left: 0;

## bottom: 0;

## top: 0;

## width: 55px;

## height: 2px;

## margin: auto;

## background: #eb315a;

## }

## .movieDetail-left {

## color: #eb315a;

## min-width: 100px;

## }

## .movieDetails-content ul li {

## display: flex;

## margin-top: 12px;

## font-weight: 300;

## }

## .movieDetail-right .theme-btn {

## margin: 0 8px;

## padding: 2px 15px;

## }

## .theme-btn {

## background: #eb315a;

## border: 1px solid #eb315a;

## color: #fff;

## border-radius: 35px;

## padding: 5px 25px;

## display: inline-block;

## font-size: 16px;

## cursor: pointer;

## font-weight: 500;

## }

## .theme-btn:hover {

## border: 1px solid #eb315a;

## background: transparent;

## color: #eb315a;

## }

## .details > .theme-btn {

## position: absolute;

## right: 0;

## bottom: 0;

## border-radius: 5px 0 0;

## }

## @media only screen and (max-width: 991px) {

## .details .movieDetails-content img {

## margin: auto;

## }

## }

## @media only screen and (max-width: 576px) {

## .movieDetails-content ul li {

## display: block;

## }

## .movieDetails-content > p::before {

## display: none;

## }

## .movieDetails-content > p {

## padding: 0;

## }

## .movieDetails-content h2 {

## font-size: 24px;

## }

## .details {

## padding: 5px 15px 40px;

## }

## .transformers-content > h2 + a.theme-btn {

## margin: 10px 5px 5px 0;

## }

## }

## ****detail-view-booking.component.html****

## <section id="detailMovie">

## <div class="container">

## <div class="section-title">

## <h1><i class="icofont icofont-film"></i>Booking Summary</h1>

## </div>

## <hr />

## <div class="details">

## <div class="row flexbox-center">

## <div class="col-lg-7">

## <div class="movieDetails-content">

## <h2>Booking Details</h2>

## <br />

## <ul>

## <li>

## <div class="movieDetail-left">Movie:</div>

## <div class="movieDetail-right">

## <a href="#">{{ booking.show.movie.movieName }}</a>

## </div>

## </li>

## <li>

## <div class="movieDetail-left">Show Date:</div>

## <div class="movieDetail-right">{{ booking.show.showDate }}</div>

## </li>

## <li>

## <div class="movieDetail-left">Show Start Time:</div>

## <div class="movieDetail-right">

## {{ booking.show.showStartTime.show.showStartTime }}

## </div>

## </li>

## <li>

## <div class="movieDetail-left">Transaction Mode:</div>

## <div class="movieDetail-right">

## {{ booking.transactionMode }}

## </div>

## </li>

## <li>

## <div class="movieDetail-left">Transaction Status:</div>

## <div class="movieDetail-right">

## {{ booking.transactionStatus }}

## </div>

## </li>

## <li>

## <div class="movieDetail-left">Total Cost:</div>

## <div class="movieDetail-right">

## {{ booking.totalCost }}

## </div>

## </li>

## </ul>

## </div>

## </div>

## </div>

## <a [routerLink]="['/movie']" class="theme-btn"

## ><i class="icofont-ticket"></i>Back to MOVIES</a

## >

## </div>

## </div>

## </section>

## ****detail-view-booking.component.spec.ts****

## import { ComponentFixture, TestBed } from '@angular/core/testing';

## import { DetailViewBookingComponent } from './detail-view-booking.component';

## describe('DetailViewBookingComponent', () => {

## let component: DetailViewBookingComponent;

## let fixture: ComponentFixture<DetailViewBookingComponent>;

## beforeEach(async () => {

## await TestBed.configureTestingModule({

## declarations: [ DetailViewBookingComponent ]

## })

## .compileComponents();

## });

## beforeEach(() => {

## fixture = TestBed.createComponent(DetailViewBookingComponent);

## component = fixture.componentInstance;

## fixture.detectChanges();

## });

## it('should create', () => {

## expect(component).toBeTruthy();

## });

## });

## ****list-view-booking.component.css****

## ul,

## ol {

## list-style-type: none;

## }

## img {

## max-width: 100%;

## max-height: 100%;

## }

## a {

## text-decoration: none;

## color: #242221;

## transition: color 0.3s ease;

## }

## @media (hover: hover) {

## a:hover {

## color: #fc5185;

## }

## }

## button {

## padding: 0.85rem;

## background-color: rgb(255, 254, 254);

## border: 0;

## color: #000000;

## cursor: pointer;

## box-shadow: 0 0 0 1px rgba(145, 116, 216, 0.5);

## transition: color 0.3s ease, background-color 0.3s ease;

## }

## .focus--box-shadow:focus {

## z-index: 1;

## outline: none;

## box-shadow: 0 0 0 3px rgba(145, 116, 216, 0.5);

## }

## .wrapper {

## display: flex;

## justify-content: center;

## width: 100%;

## min-height: 100vh;

## background-color: #e9e8ec;

## font-size: 16px;

## color: #242221;

## }

## @media (max-width: 992px) {

## .wrapper {

## flex-wrap: wrap;

## }

## }

## .main {

## width: calc(100% - 46.125rem);

## margin: 0 3.75rem 6rem 1.875rem;

## }

## @media (max-width: 1800px) {

## .main {

## width: calc(100% - 38.8125rem);

## margin: 0 3rem 6rem 1.5rem;

## }

## }

## @media (max-width: 1400px) {

## .main {

## width: calc(100% - 33.8125rem);

## margin: 0 2rem 6rem 1rem;

## }

## }

## @media (max-width: 1200px) {

## .main {

## width: calc(100% - 27.125rem);

## }

## }

## @media (max-width: 992px) {

## .main {

## width: 100%;

## order: 2;

## margin: 0 1rem 6rem 1rem;

## }

## }

## .section {

## padding-top: 5rem;

## }

## .adminList-item:not(:last-child) {

## margin-bottom: 1.2rem;

## }

## .adminList-link {

## position: relative;

## display: block;

## width: 100%;

## background-color: #f8f8f8;

## padding: 1.5rem 0.7rem;

## border-radius: 10px;

## transition: background-color 0.3s ease;

## }

## button .theme-btn {

## background-color: none;

## }

## @media (hover: hover) {

## .adminList-link:hover .adminList-inform-name {

## color: #fc5185;

## }

## }

## .adminList\_\_item:not(:last-child) {

## margin-bottom: 1.2rem;

## }

## .adminList-wrapper {

## display: flex;

## align-items: center;

## justify-content: space-between;

## width: 100%;

## }

## .adminList-element {

## padding: 0 0.8rem;

## }

## .adminList-inform {

## width: 28%;

## }

## .adminList-setting {

## width: 8%;

## text-align: center;

## }

## .adminList-inform-name {

## font-size: 1.2rem;

## line-height: 1.5;

## color: #242221;

## }

## @media (max-width: 1800px) {

## .adminList-link {

## padding: 1.2rem 0.7rem;

## }

## .adminList-element {

## padding: 0 0.5rem;

## }

## .adminList-inform-name {

## font-size: 1rem;

## }

## }

## @media (max-width: 1400px) {

## .adminList-wrapper {

## flex-wrap: wrap;

## }

## .adminList-inform {

## order: 2;

## width: calc(100% - 90px);

## margin-bottom: 1.5rem;

## padding-right: 4rem;

## }

## .adminList-setting {

## position: absolute;

## top: 1.2rem;

## right: 0.7rem;

## order: 3;

## width: 4rem;

## height: 4rem;

## margin: 0;

## }

## }

## @media (max-width: 992px) {

## .adminList-link {

## padding: 1rem 0.5rem;

## }

## .adminList-element {

## padding: 0 0.3rem;

## }

## .adminList-inform {

## padding-right: 3rem;

## }

## .adminList-setting {

## right: 0.2rem;

## width: 3rem;

## height: 3rem;

## }

## }

## @media (max-width: 768px) {

## .adminList-date {

## order: 5;

## width: 50%;

## margin-bottom: 1.5rem;

## text-align: left;

## }

## .adminList-icon {

## max-width: 50px;

## min-width: 50px;

## }

## .adminList-inform {

## width: calc(100% - 55px);

## }

## }

## .nav {

## display: flex;

## width: 5.6rem;

## height: calc(100vh - 19.25rem);

## min-height: 40rem;

## max-height: 50rem;

## margin: 11.53rem 2.45rem;

## padding: 7.5rem 0.825rem;

## background-color: #f8f8f8;

## border-radius: 20px;

## }

## .nav-list {

## display: flex;

## align-items: center;

## justify-content: center;

## flex-wrap: wrap;

## width: 100%;

## }

## .nav-link {

## display: block;

## padding: 10px;

## font-size: 0;

## border-radius: 10px;

## background-color: #f8f8f8;

## text-align: center;

## transition: background-color 0.3s ease;

## }

## @media (hover: hover) {

## .nav-link:hover {

## background-color: #fc5185;

## }

## .nav-link:hover .nav-icon path {

## fill: #f8f8f8;

## }

## }

## .nav-item-isActive .nav-link {

## background-color: #242221;

## }

## .nav-item-isActive .nav-link .nav-icon path {

## fill: #f8f8f8;

## }

## .nav-item-isActive .nav-link:focus {

## outline: none;

## box-shadow: 0 0 0 3px rgba(36, 34, 33, 0.5);

## }

## .nav-icon {

## width: 100%;

## }

## @media (max-width: 1800px) {

## .nav {

## width: 4.6875rem;

## height: calc(100vh - 16.25rem);

## min-height: 35rem;

## max-height: 45rem;

## margin: 9.625rem 1.5rem;

## padding: 7.5rem 0.625rem;

## }

## }

## @media (max-width: 1400px) {

## .nav {

## height: calc(100vh - 21.25rem);

## margin: 7.625rem 1rem;

## padding: 5.5rem 0.625rem;

## }

## }

## @media (max-width: 1200px) {

## .nav {

## z-index: 5;

## position: fixed;

## bottom: 0;

## left: 0;

## width: calc(100% - 27.125rem);

## height: auto;

## min-height: auto;

## margin: 0 1rem;

## padding: 0.5rem 5rem;

## border-radius: 10px 10px 0 0;

## box-shadow: 0px 3px 10px 4px rgba(36, 34, 33, 0.2);

## }

## .nav-list {

## justify-content: space-around;

## }

## .nav-item {

## width: 3.3rem;

## height: 3.3rem;

## }

## }

## @media (max-width: 992px) {

## .nav {

## width: calc(100% - 2rem);

## }

## .nav-item {

## width: 2.5rem;

## height: 2.5rem;

## }

## }

## @media (max-width: 768px) {

## .nav {

## padding: 0.5rem 1rem;

## }

## }

## .fab-container {

## position: fixed;

## bottom: 50px;

## right: 50px;

## z-index: 999;

## cursor: pointer;

## }

## .fab-icon-holder {

## width: 50px;

## height: 50px;

## border-radius: 100%;

## background: #016fb9;

## box-shadow: 0 6px 20px rgba(0, 0, 0, 0.2);

## }

## .fab-icon-holder:hover {

## opacity: 0.8;

## }

## .fab-icon-holder i {

## display: flex;

## align-items: center;

## justify-content: center;

## height: 100%;

## font-style: 25px;

## color: #fff;

## }

## .fab {

## width: 60px;

## height: 60px;

## background: #eb315a;

## }

## .fab-options {

## list-style-type: none;

## margin: 0;

## position: absolute;

## bottom: 70px;

## right: 0;

## opacity: 0;

## transition: all 0.3s ease;

## transform: scale(0);

## transform-origin: 85% bottom;

## }

## .fab:hover + .fab-options,

## .fab-options:hover {

## opacity: 1;

## transform: scale(1);

## }

## ****list-view-booking.component.html****

## <div class="wrapper">

## <nav class="nav">

## <ul class="nav-list" role="menubar">

## <li class="nav-item nav-item-isActive">

## <a

## [routerLink]="['/home']"

## class="nav-link focus--box-shadow"

## role="menuitem"

## aria-label="Home"

## >

## <svg

## class="nav-icon"

## xmlns="http://www.w3.org/2000/svg"

## viewBox="0 0 24 24"

## role="presentation"

## >

## <path

## fill="#6563ff"

## d="M20,8h0L14,2.74a3,3,0,0,0-4,0L4,8a3,3,0,0,0-1,2.26V19a3,3,0,0,0,3,3H18a3,3,0,0,0,3-3V10.25A3,3,0,0,0,20,8ZM14,20H10V15a1,1,0,0,1,1-1h2a1,1,0,0,1,1,1Zm5-1a1,1,0,0,1-1,1H16V15a3,3,0,0,0-3-3H11a3,3,0,0,0-3,3v5H6a1,1,0,0,1-1-1V10.25a1,1,0,0,1,.34-.75l6-5.25a1,1,0,0,1,1.32,0l6,5.25a1,1,0,0,1,.34.75Z"

## />

## </svg>

## </a>

## </li>

## <li class="nav-item">

## <a

## [routerLink]="['/movie/admin/list']"

## class="nav-link focus--box-shadow"

## role="menuitem"

## aria-label="Favorite projects"

## >

## <img src="assets/icon/movie-film.png" alt="" />

## </a>

## </li>

## <li class="nav-item">

## <a

## [routerLink]="['/theatre']"

## class="nav-link focus--box-shadow"

## role="menuitem"

## aria-label="Informational messages"

## ><img src="assets/icon/screen.png" alt="" />

## </a>

## </li>

## <li class="nav-item">

## <a

## [routerLink]="['/booking']"

## class="nav-link focus--box-shadow"

## role="menuitem"

## aria-label="Analytics"

## >

## <img src="assets/icon/ticket.png" alt="" />

## </a>

## </li>

## </ul>

## </nav>

## <main class="main">

## <section class="section">

## <ul class="adminList">

## <li class="adminList\_\_item">

## <a

## href="#"

## style="pointer-events: none"

## class="adminList-link focus--box-shadow"

## >

## <div class="adminList-wrapper">

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name"

## ><strong>Transaction Mode</strong></span

## >

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name"

## ><strong>Transaction Status</strong>

## </span>

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name"

## ><strong>Total Cost</strong></span

## >

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name"

## ><strong>Booking Date</strong>

## </span>

## </div>

## <div class="adminList-element adminList-setting">

## <span></span>

## </div>

## </div>

## </a>

## </li>

## </ul>

## <hr />

## <ul class="adminList">

## <li class="adminList\_\_item" \*ngFor="let bk of bookings">

## <a

## [routerLink]="['/booking/view', bk.transactionId]"

## class="adminList-link focus--box-shadow"

## >

## <div class="adminList-wrapper">

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name">{{

## bk.transactionMode

## }}</span>

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name">{{

## bk.transactionStatus

## }}</span>

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name">{{ bk.totalCost }}</span>

## </div>

## <div class="adminList-element adminList-inform">

## <span class="adminList-inform-name">{{ bk.bookingDate }}</span>

## </div>

## <div class="adminList-element adminList-setting">

## <button

## class="focus--box-shadow"

## type="button"

## [routerLink]="['/booking/view', bk.transactionId]"

## >

## <i class="icofont-caret-right"></i>

## </button>

## </div>

## </div>

## </a>

## </li>

## </ul>

## </section>

## </main>

## </div>

## ****list-view-booking.component.spec.ts****

## import { ComponentFixture, TestBed } from '@angular/core/testing';

## import { ListViewBookingComponent } from './list-view-booking.component';

## describe('ListViewBookingComponent', () => {

## let component: ListViewBookingComponent;

## let fixture: ComponentFixture<ListViewBookingComponent>;

## beforeEach(async () => {

## await TestBed.configureTestingModule({

## declarations: [ ListViewBookingComponent ]

## })

## .compileComponents();

## });

## beforeEach(() => {

## fixture = TestBed.createComponent(ListViewBookingComponent);

## component = fixture.componentInstance;

## fixture.detectChanges();

## });

## it('should create', () => {

## expect(component).toBeTruthy();

## });

## });

## ****list-view-booking.component.ts****

## import { Component, OnInit } from '@angular/core';

## import { BokingService } from '../boking.service';

## @Component({

## selector: 'app-list-view-booking',

## templateUrl: './list-view-booking.component.html',

## styleUrls: ['./list-view-booking.component.css'],

## })

## /\*\*

## \* @author Jyothi

## \* @Description returns list of bookings done by the customer.

## \*/

## export class ListViewBookingComponent implements OnInit {

## bookings: any[];

## constructor(private bService: BokingService) {}

## ngOnInit(): void {

## this.bService.getAllBookings().subscribe((data: any[]) => {

## this.bookings = data;

## });

## }

## }

## ****boking.service.spec.ts****

## import { TestBed } from '@angular/core/testing';

## import { BokingService } from './boking.service';

## describe('BokingService', () => {

## let service: BokingService;

## beforeEach(() => {

## TestBed.configureTestingModule({});

## service = TestBed.inject(BokingService);

## });

## it('should be created', () => {

## expect(service).toBeTruthy();

## });

## });

## ****boking.service.ts****

## import { Injectable } from '@angular/core';

## import {

## HttpClient,

## HttpErrorResponse,

## HttpHeaders,

## } from '@angular/common/http';

## import { Observable, pipe, throwError } from 'rxjs';

## import { catchError } from 'rxjs/operators';

## import { Booking } from './booking';

## @Injectable({

## providedIn: 'root',

## })

## export class BokingService {

## private url = 'http://localhost:8000';

## httpOptions = {

## headers: new HttpHeaders({

## 'Content-Type': 'application/json',

## }),

## };

## constructor(private httpClient: HttpClient) {}

## getAllBookings(): Observable<Booking[]> {

## return this.httpClient.get<Booking[]>(this.url + '/booking' + '/findall');

## pipe(catchError(this.handleError));

## }

## addBooking(book: any, showId: number): Observable<any> {

## return this.httpClient

## .post<Booking>(

## this.url + '/booking/insert?showId=' + showId,

## JSON.stringify(book),

## this.httpOptions

## )

## .pipe(catchError(this.handleError));

## }

## updateMovie(book: any): Observable<any> {

## return this.httpClient

## .put<Booking>(

## this.url + '/booking/update',

## JSON.stringify(book),

## this.httpOptions

## )

## .pipe(catchError(this.handleError));

## }

## getBookingById(id: any): Observable<Booking> {

## return this.httpClient

## .get<Booking>(this.url + '/booking/viewbooking/' + id)

## .pipe(catchError(this.handleError));

## }

## deleteBooking(bookingId: number): Observable<Booking> {

## return this.httpClient

## .delete<Booking>(this.url + '/booking/ticketbooking/' + bookingId)

## .pipe(catchError(this.handleError));

## }

## getTotalCost(id: any): Observable<Booking> {

## return this.httpClient

## .get<Booking>(this.url + '/booking/cost/' + id)

## .pipe(catchError(this.handleError));

## }

## handleError(eResponse: HttpErrorResponse) {

## if (eResponse.error instanceof ErrorEvent) {

## console.log('Client Side Error =' + eResponse.error.message);

## console.log('Status Code=' + eResponse.status);

## } else {

## console.log('Server Side Error =' + eResponse.error.message);

## console.log('Status Code=' + eResponse.status);

## }

## return throwError(eResponse.error.message);

## }

## }

## ****app-routing.module.ts****

## import { NgModule } from '@angular/core';

## import { RouterModule, Routes } from '@angular/router';

## import { BookingComponent } from './booking/booking.component';

## import { CreateBookingComponent } from './booking/create-booking/create-booking.component';

## import { ListViewBookingComponent } from './booking/list-view-booking/list-view-booking.component';

## import { DetailViewBookingComponent } from './booking/detail-view-booking/detail-view-booking.component';

## import { AdminDetailsComponent } from './movie/admin-details/admin-details.component';

## import { AllMoviesComponent } from './movie/all-movies/all-movies.component';

## import { CreateMovieComponent } from './movie/create-movie/create-movie.component';

## import { DetailViewMovieComponent } from './movie/detail-view-movie/detail-view-movie.component';

## import { MovieDashboardComponent } from './movie/movie-dashboard/movie-dashboard.component';

## import { MovieComponent } from './movie/movie.component';

## import { UpdateMoviesComponent } from './movie/update-movies/update-movies.component';

## import { CreateScreenComponent } from './screen/create-screen/create-screen.component';

## import { DetailViewScreenComponent } from './screen/detail-view-screen/detail-view-screen.component';

## import { ListViewScreenComponent } from './screen/list-view-screen/list-view-screen.component';

## import { ScreenComponent } from './screen/screen.component';

## import { ListViewSeatComponent } from './seat/list-view-seat/list-view-seat.component';

## import { SeatComponent } from './seat/seat.component';

## import { CreateShowComponent } from './show/create-show/create-show.component';

## import { DetaiViewShowComponent } from './show/detai-view-show/detai-view-show.component';

## import { ListViewShowComponent } from './show/list-view-show/list-view-show.component';

## import { ShowComponent } from './show/show.component';

## import { CreateTheatreComponent } from './theatre/create-theatre/create-theatre.component';

## import { DetailViewTheatreComponent } from './theatre/detail-view-theatre/detail-view-theatre.component';

## import { ListViewTheatreComponent } from './theatre/list-view-theatre/list-view-theatre.component';

## import { TheatreComponent } from './theatre/theatre.component';

## import { TheatresBookingComponent } from './theatre/theatres-booking/theatres-booking.component';

## import { UpdateTheatreComponent } from './theatre/update-theatre/update-theatre.component';

## import { CreateTicketComponent } from './ticket/create-ticket/create-ticket.component';

## import { TicketComponent } from './ticket/ticket.component';

## import { LoginComponent } from './login/login.component';

## import { CreateUserComponent } from './user/create-user/create-user.component';

## import { AuthGuard } from './auth.guard';

## import { CustomerComponent } from './customer/customer.component';

## import { CreateCustomerComponent } from './customer/create-customer/create-customer.component';

## import { ListViewCustomerComponent } from './customer/list-view-customer/list-view-customer.component';

## import { HomeComponent } from './home/home.component';

## import { UpdateScreensComponent } from './screen/update-screens/update-screens.component';

## import { UpdateCustomerComponent } from './customer/update-customer/update-customer.component';

## import { CreateSeatComponent } from './seat/create-seat/create-seat.component';

## const routes: Routes = [

## { path: '', redirectTo: 'home', pathMatch: 'full' },

## { path: 'home', component: HomeComponent },

## { path: 'signIn', component: LoginComponent },

## { path: 'signUp', component: CreateUserComponent },

## {

## path: 'movie',

## component: MovieComponent,

## children: [

## { path: '', component: AllMoviesComponent },

## { path: 'admin/list', component: MovieDashboardComponent },

## { path: 'view/:movieId', component: DetailViewMovieComponent },

## { path: 'admin/details/:movieId', component: AdminDetailsComponent },

## { path: 'add', component: CreateMovieComponent },

## { path: 'update/:movieId', component: UpdateMoviesComponent },

## ],

## },

## {

## path: 'theatre',

## component: TheatreComponent,

## children: [

## { path: '', component: ListViewTheatreComponent },

## { path: 'view/:theatreId', component: DetailViewTheatreComponent },

## { path: 'add', component: CreateTheatreComponent },

## { path: 'update/:theatreId', component: UpdateTheatreComponent },

## { path: 'viewbyMovie/:movieId', component: TheatresBookingComponent },

## ],

## },

## {

## path: 'screen',

## component: ScreenComponent,

## children: [

## { path: '', component: ListViewScreenComponent },

## { path: 'view/:screenId', component: DetailViewScreenComponent },

## { path: 'add/:theatreId', component: CreateScreenComponent },

## { path: 'update/:screenId', component: UpdateScreensComponent },

## ],

## },

## {

## path: 'show',

## component: ShowComponent,

## children: [

## { path: '', component: ListViewShowComponent },

## { path: 'view/:showId', component: DetaiViewShowComponent },

## { path: 'add/:theatreId/:screenId', component: CreateShowComponent },

## ],

## },

## {

## path: 'booking',

## component: BookingComponent,

## canActivate: [AuthGuard],

## children: [

## { path: '', component: ListViewBookingComponent },

## { path: 'view/:bookingId', component: DetailViewBookingComponent },

## { path: 'add/:showId', component: CreateBookingComponent },

## ],

## },

## {

## path: 'ticket',

## component: TicketComponent,

## children: [{ path: 'add/:bookingId', component: CreateTicketComponent }],

## },

## {

## path: 'seat',

## component: SeatComponent,

## children: [

## { path: '', component: ListViewSeatComponent },

## { path: 'add', component: CreateSeatComponent },

## ],

## },

## {

## path: 'customer',

## component: CustomerComponent,

## children: [

## { path: '', component: ListViewCustomerComponent },

## { path: 'update/:customerId', component: UpdateCustomerComponent },

## { path: 'add', component: CreateCustomerComponent },

## ],

## },

## ];

## @NgModule({

## imports: [RouterModule.forRoot(routes)],

## exports: [RouterModule],

## })

## export class AppRoutingModule {}

## ****app.component.ts****

## import { Component, ViewChild } from '@angular/core';

## import { LoginComponent } from './login/login.component';

## @Component({

## selector: 'app-root',

## templateUrl: './app.component.html',

## styleUrls: ['./app.component.css'],

## })

## export class AppComponent {

## title = 'MTB-App';

## // @ViewChild(LoginComponent) login: any;

## // ngAfterViewInit() {

## // console.log(this.login);

## // }

## }

## ****app.module.ts****

## import { NgModule } from '@angular/core';

## import { BrowserModule } from '@angular/platform-browser';

## import { AppRoutingModule } from './app-routing.module';

## import { AppComponent } from './app.component';

## import { MovieComponent } from './movie/movie.component';

## import { AllMoviesComponent } from './movie/all-movies/all-movies.component';

## import { FormsModule, ReactiveFormsModule } from '@angular/forms';

## import { HttpClientModule } from '@angular/common/http';

## import { CreateMovieComponent } from './movie/create-movie/create-movie.component';

## import { UpdateMoviesComponent } from './movie/update-movies/update-movies.component';

## import { DetailViewMovieComponent } from './movie/detail-view-movie/detail-view-movie.component';

## import { HomeComponent } from './home/home.component';

## import { TheatreComponent } from './theatre/theatre.component';

## import { ListViewTheatreComponent } from './theatre/list-view-theatre/list-view-theatre.component';

## import { DetailViewTheatreComponent } from './theatre/detail-view-theatre/detail-view-theatre.component';

## import { CreateTheatreComponent } from './theatre/create-theatre/create-theatre.component';

## import { UpdateTheatreComponent } from './theatre/update-theatre/update-theatre.component';

## import { ScreenComponent } from './screen/screen.component';

## import { ListViewScreenComponent } from './screen/list-view-screen/list-view-screen.component';

## import { CreateScreenComponent } from './screen/create-screen/create-screen.component';

## import { ShowComponent } from './show/show.component';

## import { ListViewShowComponent } from './show/list-view-show/list-view-show.component';

## import { CreateShowComponent } from './show/create-show/create-show.component';

## import { DetailViewScreenComponent } from './screen/detail-view-screen/detail-view-screen.component';

## import { NavbarComponent } from './navbar/navbar.component';

## import { MovieDashboardComponent } from './movie/movie-dashboard/movie-dashboard.component';

## import { BookingComponent } from './booking/booking.component';

## import { DetailViewBookingComponent } from './booking/detail-view-booking/detail-view-booking.component';

## import { ListViewBookingComponent } from './booking/list-view-booking/list-view-booking.component';

## import { SearchMoviePipe } from './movie/search-movie.pipe';

## import { DetaiViewShowComponent } from './show/detai-view-show/detai-view-show.component';

## import { AdminDetailsComponent } from './movie/admin-details/admin-details.component';

## import { CreateBookingComponent } from './booking/create-booking/create-booking.component';

## import { TicketComponent } from './ticket/ticket.component';

## import { CreateTicketComponent } from './ticket/create-ticket/create-ticket.component';

## import { SeatComponent } from './seat/seat.component';

## import { ListViewSeatComponent } from './seat/list-view-seat/list-view-seat.component';

## import { TheatresBookingComponent } from './theatre/theatres-booking/theatres-booking.component';

## import { UpdateScreensComponent } from './screen/update-screens/update-screens.component';

## import { UpdateShowComponent } from './show/update-show/update-show.component';

## import { UserComponent } from './user/user.component';

## import { CreateUserComponent } from './user/create-user/create-user.component';

## import { LoginComponent } from './login/login.component';

## import { CustomerComponent } from './customer/customer.component';

## import { CreateCustomerComponent } from './customer/create-customer/create-customer.component';

## import { ListViewCustomerComponent } from './customer/list-view-customer/list-view-customer.component';

## import { UpdateCustomerComponent } from './customer/update-customer/update-customer.component';

## import { CreateSeatComponent } from './seat/create-seat/create-seat.component';

## @NgModule({

## declarations: [

## AppComponent,

## MovieComponent,

## AllMoviesComponent,

## CreateMovieComponent,

## UpdateMoviesComponent,

## DetailViewMovieComponent,

## HomeComponent,

## TheatreComponent,

## ListViewTheatreComponent,

## DetailViewTheatreComponent,

## CreateTheatreComponent,

## UpdateTheatreComponent,

## ScreenComponent,

## ListViewScreenComponent,

## CreateScreenComponent,

## ShowComponent,

## ListViewShowComponent,

## CreateShowComponent,

## DetailViewScreenComponent,

## NavbarComponent,

## MovieDashboardComponent,

## BookingComponent,

## DetailViewBookingComponent,

## ListViewBookingComponent,

## SearchMoviePipe,

## DetaiViewShowComponent,

## AdminDetailsComponent,

## CreateBookingComponent,

## TicketComponent,

## CreateTicketComponent,

## SeatComponent,

## ListViewSeatComponent,

## TheatresBookingComponent,

## UpdateScreensComponent,

## UpdateShowComponent,

## UserComponent,

## CreateUserComponent,

## LoginComponent,

## CustomerComponent,

## CreateCustomerComponent,

## ListViewCustomerComponent,

## UpdateCustomerComponent,

## CreateSeatComponent,

## ],

## imports: [

## BrowserModule,

## AppRoutingModule,

## FormsModule,

## ReactiveFormsModule,

## HttpClientModule,

## ],

## providers: [],

## bootstrap: [AppComponent],

## })

## export class AppModule {}

## ****auth.guard.ts****

## import { Injectable } from '@angular/core';

## import {

## CanActivate,

## ActivatedRouteSnapshot,

## RouterStateSnapshot,

## UrlTree,

## Router,

## } from '@angular/router';

## import { Observable } from 'rxjs';

## import { LogginService } from './login/loggin.service';

## @Injectable({

## providedIn: 'root',

## })

## export class AuthGuard implements CanActivate {

## constructor(private lService: LogginService, private router: Router) {}

## canActivate(

## route: ActivatedRouteSnapshot,

## state: RouterStateSnapshot

## ):

## | Observable<boolean | UrlTree>

## | Promise<boolean | UrlTree>

## | boolean

## | UrlTree {

## if (!this.lService.getLoginStatus()) {

## return this.router.parseUrl('signIn');

## }

## return this.lService.getLoginStatus();

## }

## }