MYMOVIE PLAN CAPSTONE

PROJECT SOURCE CODE

***MavenWrapperDownloader.java***



import java.net.\*; import java.io.\*;

import java.nio.channels.\*; import java.util.Properties;

public class MavenWrapperDownloader {

private static final String WRAPPER\_VERSION = "0.5.6";

private static final String DEFAULT\_DOWNLOAD\_URL = "https://repo.maven.apache.org/maven2/io/takari/maven- wrapper/"

+ WRAPPER\_VERSION + "/maven-wrapper-" + WRAPPER\_VERSION + ".jar";

private static final String MAVEN\_WRAPPER\_PROPERTIES\_PATH =

".mvn/wrapper/maven-wrapper.properties"; private static final String MAVEN\_WRAPPER\_JAR\_PATH =

".mvn/wrapper/maven-wrapper.jar"; private static final String

PROPERTY\_NAME\_WRAPPER\_URL = "wrapperUrl"; public static void main(String args[]) {

System.out.println("- Downloader started"); File baseDirectory = new File(args[0]); System.out.println("- Using base directory: " +

baseDirectory.getAbsolutePath());

File mavenWrapperPropertyFile = new File(baseDirectory, MAVEN\_WRAPPER\_PROPERTIES\_PATH); String url = DEFAULT\_DOWNLOAD\_URL;

if(mavenWrapperPropertyFile.exists()) {

FileInputStream mavenWrapperPropertyFileInputStream = null;



try { mavenWrapperPropertyFileInputStream =

new FileInputStream(mavenWrapperPropertyFile);

Properties mavenWrapperProperties = new

Properties();

mavenWrapperProperties.load(mavenWrapperPropertyFileInpu tStream);

url = mavenWrapperProperties.getProperty(PROPERTY\_NAME\_WRAPPER

\_URL, url);

} catch (IOException e) { System.out.println("- ERROR loading '" +

MAVEN\_WRAPPER\_PROPERTIES\_PATH + "'");

} finally {

try { if(mavenWrapperPropertyFileInputStream != null) {

mavenWrapperPropertyFileInputStream.close();

}

} catch (IOException e) {

// Ignore ...

}

}

}

System.out.println("- Downloading from: " +

url);

File outputFile = new File(baseDirectory.getAbsolutePath(), MAVEN\_WRAPPER\_JAR\_PATH);



if(!outputFile.getParentFile().exists()) { if(!outputFile.getParentFile().mkdirs()) {

System.out.println(

"- ERROR creating output

directory '" + outputFile.getParentFile().getAbsolutePath() + "'");

}

}

System.out.println("- Downloading to: " + outputFile.getAbsolutePath());

try {

downloadFileFromURL(url, outputFile); System.out.println("Done"); System.exit(0);

} catch (Throwable e) {

System.out.println("- Error downloading"); e.printStackTrace();

System.exit(1);

}

}

private static void downloadFileFromURL(String urlString, File destination) throws Exception {

if (System.getenv("MVNW\_USERNAME") != null && System.getenv("MVNW\_PASSWORD") != null) {

String username = System.getenv("MVNW\_USERNAME");

char[] password = System.getenv("MVNW\_PASSWORD").toCharArray();



Authenticator.setDefault(new Authenticator()

{

@Override

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(username, password);

}

});

}

URL website = new URL(urlString); ReadableByteChannel rbc;

rbc = Channels.newChannel(website.openStream()); FileOutputStream fos = new

FileOutputStream(destination); fos.getChannel().transferFrom(rbc, 0,

Long.MAX\_VALUE);

fos.close();

rbc.close();

}

}

## InitialData.java

package com.MyMoviePlan.config;

import com.MyMoviePlan.service.UserService; import

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

import org.springframework.boot.CommandLineRunner;



import org.springframework.security.crypto.password.PasswordEnc oder;

import org.springframework.stereotype.Component;

@Component

public class InitialData implements CommandLineRunner {

@Autowired

private UserService service; @Autowired

private PasswordEncoder passwordEncoder;

@Override

public void run(String... args) throws Exception {

}

}

***AuditoriumController.java***

package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.\*; import

com.MyMoviePlan.exception.AuditoriumNotFoundException; import com.MyMoviePlan.exception.BookingNotFoundException; import com.MyMoviePlan.exception.MovieShowNotFoundException; import com.MyMoviePlan.exception.ShowNotFoundException;

import com.MyMoviePlan.model.TicketDetails;

import com.MyMoviePlan.model.UserRole; import com.MyMoviePlan.repository.\*; import com.MyMoviePlan.service.UserService; import lombok.AllArgsConstructor;



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

;

import org.springframework.web.bind.annotation.\*; import java.util.List;

import java.util.stream.Collectors; @CrossOrigin

@RestController @RequestMapping("/auditorium") @AllArgsConstructor

public class AuditoriumController { private final ShowRepository show; private final UserService service;

private final BookingRepository booking; private final MovieRepository movie;

private final MovieShowsRepository movieShow; private final AuditoriumRepository auditorium; @GetMapping({"/", "all"})

public List<AuditoriumEntity> findAllAuditoriums() { return this.auditorium.findAll();

}

@GetMapping("{auditorium\_id}") @PreAuthorize("hasAuthority('WRITE')") public AuditoriumEntity

findAuditoriumById(@PathVariable final int auditorium\_id) {

return this.auditorium.findById(auditorium\_id)

.orElseThrow(() -> new



AuditoriumNotFoundException("Auditorium with id: " + auditorium\_id + " not found."));

}

@PostMapping("add") @PreAuthorize("hasAuthority('WRITE')")

public AuditoriumEntity saveAuditorium(@RequestBody final AuditoriumEntity auditorium) {

return this.auditorium.save(auditorium);

}

@PutMapping("update") @PreAuthorize("hasAuthority('UPDATE')") public AuditoriumEntity

updateAuditorium(@RequestBody final AuditoriumEntity auditorium) {

return this.auditorium.save(auditorium);

}

@DeleteMapping("delete/{auditorium\_id}") @PreAuthorize("hasAuthority('DELETE')")

public void deleteAuditorium(@PathVariable final int auditorium\_id) {

this.auditorium.deleteById(auditorium\_id);

}

@GetMapping("{auditorium\_id}/show/{show\_id}") public ShowEntity findShowById(@PathVariable final

int auditorium\_id,

@PathVariable final

int show\_id) {

return

this.findAuditoriumById(auditorium\_id).getShows()

.stream()



.filter(show -> show.getId() == show\_id)

.findFirst()

.orElseThrow(() ->

new ShowNotFoundException("Show with Id: " + show\_id + " not found"));

}

@GetMapping("{auditorium\_id}/show/all")

public List<ShowEntity> findAllShows(@PathVariable final int auditorium\_id) {

return this.findAuditoriumById(auditorium\_id).getShows();

}

@PostMapping("{auditorium\_id}/show/add") @PreAuthorize("hasAuthority('WRITE')")

public ShowEntity saveShow(@PathVariable final int auditorium\_id,

@RequestBody final

ShowEntity show) {

final AuditoriumEntity auditorium = this.findAuditoriumById(auditorium\_id);

show.setAuditorium(auditorium); return this.show.save(show);

}

@PutMapping("{auditorium\_id}/show/update") @PreAuthorize("hasAuthority('UPDATE')")

public ShowEntity updateShow(@PathVariable final int auditorium\_id,

@RequestBody final

ShowEntity show) {

final AuditoriumEntity auditorium = this.findAuditoriumById(auditorium\_id);

show.setAuditorium(auditorium); return this.show.save(show);

}

@DeleteMapping("{auditorium\_id}/show/delete/{show\_id}") @PreAuthorize("hasAuthority('DELETE')")

public void deleteShow(@PathVariable final int auditorium\_id,

@PathVariable final int

show\_id) {

final ShowEntity show = this.findShowById(auditorium\_id, show\_id);

this.show.deleteById(show.getId());

}

@GetMapping("movie/{movieId}") public List<AuditoriumEntity>

findAuditoriumsByMovieId(@PathVariable final int movieId) {

return this.findAllAuditoriums().stream()

.filter(halls -> halls.getShows()

.stream()

.anyMatch(show ->

show.getMovieShows()

.stream()

.anyMatch(m\_show ->

m\_show.getMovieId() == movieId)))



.collect(Collectors.toList());

}

@GetMapping("{auditorium\_id}/movie/{movieId}")

public List<ShowEntity> findShowsByMovieId(@PathVariable final int auditorium\_id, @PathVariable final int movieId) {



return this.findAllShows(auditorium\_id).stream()

.filter(show -> show.getMovieShows()

.stream()

.anyMatch(m\_show -> m\_show.getMovieId() == movieId))

.collect(Collectors.toList());

}

@GetMapping("{auditorium\_id}/show/{show\_id}/movie- show/all")

public List<MovieShowsEntity> findAllMovieShows(@PathVariable final int auditorium\_id,

@PathVariable final int show\_id) {

return this.findShowById(auditorium\_id, show\_id)

.getMovieShows();

}

@GetMapping("{auditorium\_id}/show/{show\_id}/movie- show/{movie\_show\_id}")

public MovieShowsEntity findMovieShowById(@PathVariable final int auditorium\_id,

@PathVariable final int show\_id, @PathVariable final int movie\_show\_id) {

return this.findShowById(auditorium\_id, show\_id)

.getMovieShows()

.stream()

.filter(movie\_show -> movie\_show.getId()

== movie\_show\_id)

.findFirst()

.orElseThrow(

() -> new MovieShowNotFoundException("Movie Show with id: "

+ movie\_show\_id + " not

found"));

}

@PutMapping("{auditorium\_id}/show/{show\_id}/movie- show/update")

@PreAuthorize("hasAuthority('UPDATE')") public MovieShowsEntity

updateMovieShow(@PathVariable final int auditorium\_id,

@PathVariable final int show\_id,

final MovieShowsEntity movieShow) { final ShowEntity show =

this.findShowById(auditorium\_id, show\_id); movieShow.setShow(show);

@RequestBody

return this.movieShow.save(movieShow);



}

@DeleteMapping("{auditorium\_id}/show/{show\_id}/movie- show/delete/{movie\_show\_id}")

@PreAuthorize("hasAuthority('DELETE')")

public void deleteMovieShow(@PathVariable final int auditorium\_id,

@PathVariable final int

show\_id,

@PathVariable final int



movie\_show\_id) {

final MovieShowsEntity movieShow = this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id);

this.movieShow.deleteById(movieShow.getMovieId());

}

@PathVariable

final int booking\_id) {

final MovieShowsEntity movieShow = this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id);

return movieShow.getBookings()

.stream().filter(booking -> booking.getId() == booking\_id)

.findFirst()

.orElseThrow(() -> new BookingNotFoundException("Booking with id: "

+ booking\_id + " not found."));

}

final UserEntity user =

this.service.getLoggedInUser(); if

(user.getUserRole().equals(UserRole.ROLE\_ADMIN) || user.getUserRole().equals(UserRole.ROLE\_SUPER\_ADMIN))

return this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id).getBookings();

else

return this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id).getBookings()

.stream().filter(booking -> booking.getUserId().equals(user.getId()))

.collect(Collectors.toList());

}

@RequestBody final BookingEntity booking) { final MovieShowsEntity moveShow =

this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id);

booking.setUserId(this.service.getLoggedInUser().getId()

);

//

booking.setUserId(this.service.findByMobile("8099531318 ").get().getId());

booking.setMovieShow(moveShow); booking.setBookingDetails(new

BookingDetailsEntity(auditorium\_id, show\_id, movie\_show\_id, moveShow.getMovieId()));

return this.booking.save(booking);

}

@PutMapping("{auditorium\_id}/show/{show\_id}/movie- show/{movie\_show\_id}/booking/update")

@PreAuthorize("hasAuthority('UPDATE')")

public BookingEntity updateBooking(@PathVariable final int auditorium\_id,



final int show\_id, final int movie\_show\_id,

@PathVariable @PathVariable

@RequestBody

final BookingEntity booking) {

final MovieShowsEntity moveShow = this.findMovieShowById(auditorium\_id, show\_id, movie\_show\_id);

booking.setMovieShow(moveShow); return this.booking.save(booking);

}

@DeleteMapping("{auditorium\_id}/show/{show\_id}/movie- show/{movie\_show\_id}/booking/delete/{booking\_id}")

@PreAuthorize("hasAuthority('READ')")

public void deleteBookingById(@PathVariable final int auditorium\_id,

int show\_id,

int movie\_show\_id, int booking\_id) {

@PathVariable final @PathVariable final @PathVariable final

final BookingEntity booking = this.findBookingById(auditorium\_id, show\_id, movie\_show\_id, booking\_id);



this.booking.deleteById(booking.getId());

}

@GetMapping("ticket-details/{booking\_id}") @PreAuthorize("hasAuthority('READ')")

public TicketDetails getMovieDetails(@PathVariable final int booking\_id) {

final PaymentEntity payment = this.booking.findById(booking\_id).get().getPayment();



final MovieShowsEntity movieShow = this.movieShow.findAll().stream().filter(m\_show -> m\_show.getBookings()

.stream().anyMatch(booking -> booking.getId() == booking\_id)).findFirst().get();

final MovieEntity movie = this.movie.findById(movieShow.getMovieId()).get();

final ShowEntity showEntity = show.findAll().stream()

.filter(show -> show.getMovieShows()

.stream().anyMatch(m\_show -> m\_show.getId() == movieShow.getId())).findFirst().get();

final AuditoriumEntity auditorium = this.auditorium.findAll().stream().filter(hall -> hall.getShows()

.stream().anyMatch(show -> show.getId()

== showEntity.getId())).findFirst().get();

return new TicketDetails(auditorium.getName(), showEntity.getName(), showEntity.getStartTime(), payment.getAmount(), movie.getName(), movie.getImage(), movie.getBgImage());

}

}

## BookingController.java



package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.BookingDetailsEntity; import com.MyMoviePlan.entity.BookingEntity; import com.MyMoviePlan.entity.UserEntity;

import com.MyMoviePlan.exception.BookingNotFoundException; import com.MyMoviePlan.model.UserRole;

import com.MyMoviePlan.repository.BookingRepository; import com.MyMoviePlan.service.UserService;

import lombok.AllArgsConstructor; import

org.springframework.security.access.prepost.PreAuthorize

;

import org.springframework.web.bind.annotation.\*; import java.util.List;

@CrossOrigin @RestController @RequestMapping("/booking") @AllArgsConstructor

public class BookingController {

private final BookingRepository repository; private final UserService service; @GetMapping("{id}") @PreAuthorize("hasAuthority('READ')")

public BookingEntity findById(@PathVariable final int id) {

return repository.findById(id)



.orElseThrow(() -> new BookingNotFoundException("Booking with id: " + id + " not found."));

}

@GetMapping("all") @PreAuthorize("hasAuthority('READ')") public List<BookingEntity> allBookings() {

final UserEntity user = service.getLoggedInUser();

if (user.getUserRole().equals(UserRole.ROLE\_ADMIN) || user.getUserRole().equals(UserRole.ROLE\_SUPER\_ADMIN))

return repository.findAll(); else return

repository.findAllByUserIdOrderByBookedOnAsc(user.getId(

));

}

@GetMapping("{username}/all") @PreAuthorize("hasAuthority('READ')") public List<BookingEntity>

findAllByUserId(@PathVariable String username) { if (!(username.contains("-") &&

username.length() > 10))

username = service.getUser(username).getId();

return repository.findAllByUserIdOrderByBookedOnAsc(username);

}



@DeleteMapping("delete/{id}") @PreAuthorize("hasAuthority('DELETE')")

public void deleteBooking(@PathVariable final int id) {

repository.deleteById(id);

}

@GetMapping("{id}/details") @PreAuthorize("hasAuthority('READ')") public BookingDetailsEntity

findByDetailsId(@PathVariable final int id) {

return this.findById(id).getBookingDetails();

}

}

## MovieShowController.java

package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.BookingEntity; import com.MyMoviePlan.entity.MovieShowsEntity; import

com.MyMoviePlan.exception.MovieShowNotFoundException; import com.MyMoviePlan.model.BookedSeats;

import com.MyMoviePlan.repository.MovieShowsRepository; import lombok.AllArgsConstructor;

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

;

import org.springframework.web.bind.annotation.\*;

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



import java.util.stream.Collectors;

@CrossOrigin @RestController

@RequestMapping("/movie-show") @AllArgsConstructor

public class MovieShowController {

private final MovieShowsRepository repository;

@PostMapping("add") @PreAuthorize("hasAuthority('WRITE')") public MovieShowsEntity save(@RequestBody

MovieShowsEntity movieShow) {

return repository.save(movieShow);

}

@GetMapping("up-coming") @PreAuthorize("hasAuthority('READ')") public List<MovieShowsEntity>

upComing(@RequestParam(value = "records", required = false) Optional<String> records) {

if (records.isPresent()) return

repository.findFewUpComing(Integer.parseInt(records.get(

)));

return repository.findAllUpComing();

}



@GetMapping("now-playing") public List<MovieShowsEntity>

nowPlaying(@RequestParam(value = "records", required = false) Optional<String> records) {

if (records.isPresent()) return

repository.findFewNowPlaying(Integer.parseInt(records.ge t()));

return repository.findAllNowPlaying();

}

@GetMapping("now-playing-up-coming") public List<MovieShowsEntity>

nowPlayingAndUpComing() {

return repository.findAllNowPlayingAndUpComing();

}

@GetMapping("not-playing") @PreAuthorize("hasAuthority('WRITE')") public List<MovieShowsEntity> notPlaying() {

return repository.findAllNotPlaying();

}

@GetMapping("all")

public List<MovieShowsEntity> findAllMovieShows() { return repository.findAll();

}

@GetMapping("{movie\_show\_id}")

public MovieShowsEntity findMovieShowById(@PathVariable final int movie\_show\_id)



{

return repository.findById(movie\_show\_id)

.orElseThrow(

() -> new MovieShowNotFoundException("Movie Show with id: " + movie\_show\_id + " not found")

);

}

@DeleteMapping("delete/{movie\_show\_id}") @PreAuthorize("hasAuthority('DELETE')")

public void deleteMovieShow(@PathVariable final int movie\_show\_id) {

repository.deleteById(movie\_show\_id);

}

/\*

\* ============================= Booking Controller ==========================

\*/

@GetMapping("{movie\_show\_id}/booked-seats/{on}") @PreAuthorize("hasAuthority('READ')")

public BookedSeats bookedSeats(@PathVariable final int movie\_show\_id, @PathVariable final String on) {

final List<BookingEntity> bookings = this.findMovieShowById(movie\_show\_id).getBookings()

.stream().filter(m\_show -> m\_show.getDateOfBooking().toString().equals(on))



.collect(Collectors.toList());

int count = 0;

List<String> seats = new ArrayList<>(); for (BookingEntity booking : bookings) { count += booking.getTotalSeats();

seats.addAll(booking.getSeatNumbers());

}

return new BookedSeats(count, seats);

}

}

## UserController.java

package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.UserEntity; import com.MyMoviePlan.model.Credentials; import com.MyMoviePlan.model.HttpResponse; import com.MyMoviePlan.model.Token;

import com.MyMoviePlan.service.UserService; import lombok.AllArgsConstructor;

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

;

import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletRequest; import java.util.List;

@CrossOrigin

@RestController @RequestMapping("/user") @AllArgsConstructor



public class UserController {

private final UserService service; private final HttpServletRequest request;

@GetMapping("/") public String index() {

return "Welcome " + service.getUserName();

}

@PostMapping("authenticate")

public Token authenticate(@RequestBody final Credentials credentials) {

return service.authenticate(credentials);

}

@GetMapping("check/{username}")

public Token checkUniqueness(@PathVariable final String username) {

return service.checkUniqueness(username);

}

@GetMapping("get-user") @PreAuthorize("hasAuthority('READ')") public UserEntity user() {

return service.getLoggedInUser()

.setPassword(null);

}



@GetMapping("all") @PreAuthorize("hasAuthority('WRITE')") public List<UserEntity> allUsers() {

return service.findAll();

}

@PutMapping("update/{username}") @PreAuthorize("hasAuthority('READ')")

public UserEntity updateUser(@RequestBody final UserEntity userEntity,

@PathVariable final

String username) {

return service.update(userEntity, username);

}

@PostMapping("sign-up")

public HttpResponse signUp(@RequestBody final UserEntity userEntity) {

return service.register(userEntity);

}

@PutMapping("change-password") @PreAuthorize("hasAuthority('READ')")

public HttpResponse changePassword(@RequestBody final Credentials credentials) {

return service.changePassword(credentials);



}

@PutMapping("forgot-password")

public HttpResponse forgotPassword(@RequestBody final Credentials credentials) {

return service.forgotPassword(credentials);

}

@DeleteMapping("delete/{username}") @PreAuthorize("hasAuthority('DELETE')") public HttpResponse delete(@PathVariable final

String username) {

return service.deleteById(username);

}

}

## ActorEntity.java

package com.MyMoviePlan.entity;

import com.fasterxml.jackson.annotation.JsonIgnore; import lombok.\*;

import javax.persistence.\*; import java.io.Serializable;

@Entity @Data

@AllArgsConstructor

@NoArgsConstructor @EqualsAndHashCode

@Table(name = "actors")



public class ActorEntity implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int id;

@Column(name = "is\_cast") private String isCast;

private String name; private String role;

@Column(length = Integer.MAX\_VALUE, columnDefinition="TEXT")

private String image;

@JsonIgnore @ToString.Exclude @EqualsAndHashCode.Exclude

@ManyToOne(targetEntity = MovieEntity.class) private MovieEntity movie;

public ActorEntity(String name, String role, String image) {

this.name = name; this.role = role; this.image = image;

}

}



## BookingDetailsEntity.java

package com.MyMoviePlan.entity;

import lombok.AllArgsConstructor; import lombok.Data;

import lombok.EqualsAndHashCode; import lombok.NoArgsConstructor;

import javax.persistence.\*; import java.io.Serializable;

@Entity @Data

@AllArgsConstructor @NoArgsConstructor @EqualsAndHashCode

@Table(name = "booking\_details")

public class BookingDetailsEntity implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int id;

@Column(name = "auditorium\_id") private int auditoriumId;

@Column(name = "show\_id") private int showId;



@Column(name = "movie\_show\_id") private int movieShowId;

@Column(name = "movie\_id") private int movieId;

public BookingDetailsEntity(int auditoriumId, int showId, int movieShowId, int movieId) {

this.auditoriumId = auditoriumId; this.showId = showId; this.movieShowId = movieShowId; this.movieId = movieId;

}

}

## PaymentEntity.java

package com.MyMoviePlan.entity;

import lombok.AllArgsConstructor; import lombok.Data;

import lombok.EqualsAndHashCode; import lombok.NoArgsConstructor;

import javax.persistence.\*; import java.io.Serializable; import java.util.Date;

@Entity

@Data @AllArgsConstructor @NoArgsConstructor @EqualsAndHashCode @Table(name = "payments")



public class PaymentEntity implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int id;

private double amount;

@Column(name = "payment\_date") @Temporal(TemporalType.DATE) private Date paymentDate;

@Column(name = "card\_number", length = 20) private String cardNumber;

@Column(name = "card\_expiry\_month", length = 5) private String cardExpiryMonth;

@Column(name = "card\_expiry\_year", length = 5) private String cardExpiryYear;

@Column(name = "card\_cvv", length = 5) private String cardCVV;

public PaymentEntity(double amount, Date paymentDate, String cardNumber, String cardExpiryMonth,

String cardExpiryYear, String



cardCVV) {

this.amount = amount; this.paymentDate = paymentDate; this.cardNumber = cardNumber;

this.cardExpiryMonth = cardExpiryMonth; this.cardExpiryYear = cardExpiryYear; this.cardCVV = cardCVV;

}

public PaymentEntity setId(int id) { this.id = id;

return this;

}

public PaymentEntity setAmount(double amount) { this.amount = amount;

return this;

}

public PaymentEntity setPaymentDate(Date paymentDate) {

this.paymentDate = paymentDate; return this;

}

public PaymentEntity setCardNumber(String cardNumber) {

this.cardNumber = cardNumber; return this;

}



public PaymentEntity setCardExpiryMonth(String cardExpiryMonth) {

this.cardExpiryMonth = cardExpiryMonth; return this;

}

public PaymentEntity setCardExpiryYear(String cardExpiryYear) {

this.cardExpiryYear = cardExpiryYear; return this;

}

public PaymentEntity setCardCVV(String cardCVV) { this.cardCVV = cardCVV;

return this;

}

}

## PriceEntity.java

package com.MyMoviePlan.entity;

import lombok.AllArgsConstructor; import lombok.Data;

import lombok.EqualsAndHashCode; import lombok.NoArgsConstructor; import javax.persistence.\*; import java.io.Serializable; @Entity

@Data



@AllArgsConstructor @NoArgsConstructor @EqualsAndHashCode @Table(name = "prices")

public class PriceEntity implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int id;

private double general; private double silver; private double gold;

public PriceEntity(double general, double silver, double gold) {

this.general = general; this.silver = silver; this.gold = gold;

}

}

***ShowEntity.java***

package com.MyMoviePlan.entity;

import com.fasterxml.jackson.annotation.JsonIgnore; import lombok.\*;

import javax.persistence.\*; import java.io.Serializable; import java.util.List;



@Entity @Data

@AllArgsConstructor @NoArgsConstructor @EqualsAndHashCode @Table(name = "shows")

public class ShowEntity implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int id;

private String name;

@Column(name = "start\_time") private String startTime;

@JsonIgnore @ToString.Exclude @EqualsAndHashCode.Exclude

@ManyToOne(targetEntity = AuditoriumEntity.class) private AuditoriumEntity auditorium;

// @JsonManagedReference @ToString.Exclude @EqualsAndHashCode.Exclude

@OneToMany(targetEntity = MovieShowsEntity.class, cascade = CascadeType.ALL)



@JoinColumn(name = "show\_id", referencedColumnName = "id")

private List<MovieShowsEntity> movieShows;

public ShowEntity(String name, String startTime, List<MovieShowsEntity> movieShows) {

this.name = name; this.startTime = startTime; this.movieShows = movieShows;

}

public ShowEntity setId(int id) { this.id = id;

return this;

}

public ShowEntity setName(String name) { this.name = name;

return this;

}

public ShowEntity setStartTime(String startTime) { this.startTime = startTime;

return this;

}

public ShowEntity setAuditorium(AuditoriumEntity auditorium) {

this.auditorium = auditorium;



return this;

}

public ShowEntity setMovieShows(List<MovieShowsEntity> movieShows) {

this.movieShows = movieShows; return this;

}

}

## BookingNotFoundException.java

package com.MyMoviePlan.exception;

public class BookingNotFoundException extends RuntimeException{

public BookingNotFoundException(String message) { super(message);

}

}

***AuditoriumNotFoundException.java***

package com.MyMoviePlan.exception;

public class AuditoriumNotFoundException extends RuntimeException {

public AuditoriumNotFoundException(String message) { super(message);



}

}

***MovieNotFoundException.java***

package com.MyMoviePlan.exception;

public class MovieNotFoundException extends RuntimeException {

public MovieNotFoundException(String message) { super(message);

}

}

***MovieShowNotFoundException.java***

package com.MyMoviePlan.exception;

public class MovieShowNotFoundException extends RuntimeException {

public MovieShowNotFoundException(String message) { super(message);

}

}

***UserNotFoundException.java***

package com.MyMoviePlan.exception;



public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String message) { super(message);

}

}

***---------------------------------------------------------------------------------------------------------***

## JWTFilter.java

package com.MyMoviePlan.filter;

import com.MyMoviePlan.model.HttpResponse; import

com.MyMoviePlan.security.ApplicationUserDetailsService; import com.MyMoviePlan.util.JWTUtil;

import io.jsonwebtoken.JwtException; import

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

import org.springframework.security.authentication.UsernamePass wordAuthenticationToken;

import org.springframework.security.core.context.SecurityContex tHolder;

import org.springframework.security.core.userdetails.UserDetail s;

import org.springframework.security.web.authentication.WebAuthe nticationDetailsSource;

import org.springframework.stereotype.Component; import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain; import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.io.IOException;

@Component()

public class JWTFilter extends OncePerRequestFilter {

@Autowired

private JWTUtil jwtUtil;

@Autowired

private ApplicationUserDetailsService userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request,

response,

HttpServletResponse

FilterChain

filterChain) throws ServletException, IOException {



try {

String authorization = request.getHeader("Authorization");

String token = null; String userName = null;

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

token = authorization.substring(7); userName =

jwtUtil.getUsernameFromToken(token);

}

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

== null) {

UserDetails userDetails

=

userDetailsService.loadUserByUsername(userName);

userDetails)) {

if (jwtUtil.validateToken(token,

UsernamePasswordAuthenticationToken

authenticationToken



= new

UsernamePasswordAuthenticationToken(userDetails,

null, userDetails.getAuthorities());

authenticationToken.setDetails(

new WebAuthenticationDetailsSource().buildDetails(request)



);

SecurityContextHolder.getContext().setAuthentication(aut henticationToken);

}

}

filterChain.doFilter(request, response);

} catch (JwtException exception) { setErrorResponse(HttpStatus.NOT\_ACCEPTABLE,

response, exception);

} catch (Exception exception) {

setErrorResponse(HttpStatus.INTERNAL\_SERVER\_ERROR, response, exception);

}

}

private void setErrorResponse(HttpStatus status, HttpServletResponse response, Exception exception) {

response.setStatus(status.value()); response.setContentType("application/json"); final HttpResponse httpResponse =

new HttpResponse(status.value(),

HttpStatus.valueOf(status.value()).getReasonPhrase(),

exception.getMessage());

try {

final String json = httpResponse.covertToJson();



response.getWriter().write(json);

} catch (IOException e) { e.printStackTrace();

}

}

}

## BookedSeats.java

package com.MyMoviePlan.model;

import lombok.AllArgsConstructor; import lombok.Data;

import lombok.NoArgsConstructor; import java.util.List;

@Data @AllArgsConstructor @NoArgsConstructor

public class BookedSeats {

private int count;

private List<String> seats;

}

***Credentials.java***

package com.MyMoviePlan.model;



import lombok.AllArgsConstructor; import lombok.Data;

import lombok.NoArgsConstructor;

@Data @AllArgsConstructor @NoArgsConstructor

public class Credentials {

private String username; private String password;

}

# TicketDetails.java

package com.MyMoviePlan.model;

import lombok.AllArgsConstructor; import lombok.NoArgsConstructor;

@AllArgsConstructor @NoArgsConstructor

public class TicketDetails { private String auditoriumName; private String showName; private String showTiming;



private double amount; private String movieName; private String movieImage;

private String movieBgImage;

}

# UserPermission.java

package com.MyMoviePlan.model;

import lombok.AllArgsConstructor; import lombok.Getter;

@Getter @AllArgsConstructor

public enum UserPermission { READ,

WRITE, UPDATE, DELETE

}

***ApplicationSecurity.java***

package com.MyMoviePlan.security;

import com.MyMoviePlan.filter.JWTFilter; import



org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.Authenticati onManager;

import org.springframework.security.authentication.dao.DaoAuthe nticationProvider;

import org.springframework.security.config.annotation.authentic ation.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.co nfiguration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.build ers.HttpSecurity;

import org.springframework.security.config.annotation.web.build ers.WebSecurity;

import org.springframework.security.config.annotation.web.confi guration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.confi guration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreation Policy;



import org.springframework.security.crypto.password.PasswordEnc oder;

import org.springframework.security.web.authentication.Username PasswordAuthenticationFilter;

import org.springframework.web.cors.CorsConfiguration; import org.springframework.web.cors.CorsConfigurationSource; import org.springframework.web.cors.UrlBasedCorsConfigurationSo urce;

import java.util.Arrays;

@Configuration @EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true) public class ApplicationSecurity extends WebSecurityConfigurerAdapter {

@Autowired

private ApplicationUserDetailsService userDetailsService;

@Autowired

private PasswordEncoder passwordEncoder;

@Autowired



private JWTFilter jwtFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.headers().frameOptions().sameOrigin(); http.csrf().disable().cors().disable()

.cors().configurationSource(corsConfigurationSource())

.and()

.authorizeRequests()

.antMatchers(HttpMethod.OPTIONS, "/\*\*")

.permitAll()

.anyRequest()

.fullyAuthenticated()

.and()

.httpBasic()

.and()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS); http.addFilterBefore(jwtFilter,

UsernamePasswordAuthenticationFilter.class);

}

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring()

.antMatchers("/h2-console/\*\*", "/auditorium/\*\*", "/movie/\*\*", "/show/\*\*", "/user/\*\*",



"/user/forgot-password", "/user/authenticate", "/movie-show/\*\*",

"/booking/\*\*", "/logout");

}

@Override protected void

configure(AuthenticationManagerBuilder auth) throws Exception {

auth.authenticationProvider(authenticationProvider());

}

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authenticationProvider

= new DaoAuthenticationProvider();

authenticationProvider.setPasswordEncoder(passwordEncode r);

authenticationProvider.setUserDetailsService(userDetails Service);

return authenticationProvider;

}

@Bean

CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = new CorsConfiguration();



configuration.setAllowedOrigins(Arrays.asList("\*"));

configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"));

configuration.setAllowCredentials(true);

//the below three lines will add the relevant CORS response headers

configuration.addAllowedOrigin("\*"); configuration.addAllowedHeader("\*"); configuration.addAllowedMethod("\*"); UrlBasedCorsConfigurationSource source = new

UrlBasedCorsConfigurationSource(); source.registerCorsConfiguration("/\*\*",

configuration);

return source;

}

@Override @Bean

protected AuthenticationManager authenticationManager() throws Exception {

return super.authenticationManager();

}

}

## ApplicationUserDetailsService.java

package com.MyMoviePlan.security;



import com.MyMoviePlan.entity.UserEntity;

import com.MyMoviePlan.exception.UserNotFoundException; import com.MyMoviePlan.service.UserService;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.security.core.userdetails.UserDetail s;

import org.springframework.security.core.userdetails.UserDetail sService;

import org.springframework.security.core.userdetails.UsernameNo tFoundException;

import org.springframework.stereotype.Service; @Service

public class ApplicationUserDetailsService implements UserDetailsService {

@Autowired

private UserService service; @Override

public UserDetails loadUserByUsername(final String username) throws UsernameNotFoundException {

final UserEntity userEntity = service.getUser(username);

return new ApplicationUserDetails(userEntity);

}

}



## MyMoviePlanApplication.java

package com.MyMoviePlan;

import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplica tion;

@SpringBootApplication

public class MyMoviePlanApplication { public static void main(String[] args) {

SpringApplication.run(MyMoviePlanApplication.class, args);

}

}

***Dockerfile***

# Start with a base image containing Java runtime FROM openjdk:8-jdk-alpine

# Add a volume pointing to /tmp VOLUME /tmp

# Make port 8080 available to the world outside this container



# EXPOSE 8080

# Add the application's jar to the container ADD target/my-movie-plan.jar my-movie-plan.jar

# Run the jar file

ENTRYPOINT ["java", "-jar", "my-movie-plan.jar"]

***Dockerfile-mysql***

pipeline {

agent any stages {

stage('docker run MySQL image') { steps {

sh "docker run --name mysql- my\_movie\_plan -e MYSQL\_ROOT\_PASSWORD=admin@root -e MYSQL\_DATABASE=my\_movie\_plan -e MYSQL\_USER=mysql -e MYSQL\_PASSWORD=root -d mysql:latest"

}

}

stage('docker images') { steps {

sh "docker images"

}

}

}

}

***Pom.xml***

<?xml version="1.0" encoding="UTF-8"?>



<project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter- parent</artifactId>

<version>2.4.2</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.MyMoviePlan</groupId>

<artifactId>my-movie-plan-backend</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<name>My Movie Plan</name>

<description>

My Movie Plan is an online movie ticket booking web application with a rich and user-friendly interface.

This is the backend code for my-movie-plan application. The frontend code of this application is developed using

Angular.

</description>



<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data- jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter- security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter- web</artifactId>

</dependency>

<!-- <dependency>-->

<!--

<groupId>org.springframework.boot</groupId>-->

<!-- <artifactId>spring-boot- devtools</artifactId>-->

<!-- <scope>runtime</scope>-->

<!-- <optional>true</optional>-->



<!-- </dependency>-->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector- java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>



<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter- tomcat</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter- test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security- test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>



<!--

https://stackoverflow.com/questions/42390860/configure- active-profile-in-springboot-via-maven -->

<!-- mvn package -P dev -->

<!-- java -jar - Dspring.profiles.active=test employee-management.jar

-->

<profiles>

<profile>

<id>dev</id>

<properties>

<maven.test.skip>true</maven.test.skip>

<activatedProperties>dev</activatedProperties>

</properties>

</profile>

<profile>

<id>test</id>

<properties>

<activatedProperties>test</activatedProperties>

</properties>

</profile>

<profile>

<id>prod</id>

<properties>

<maven.test.skip>true</maven.test.skip>



<activatedProperties>prod</activatedProperties>

</properties>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

</profiles>

<build>

<finalName>my-movie-plan</finalName>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven- plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>



<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-help- plugin</artifactId>

<version>3.2.0</version>

<executions>

<execution>

<id>show-profiles</id>

<phase>compile</phase>

<goals>

<goal>active-profiles</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

==============================X================================