package com.MyMoviePlan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MyMoviePlanApplication {

public static void main(String[] args) {

SpringApplication.run(MyMoviePlanApplication.class, args);

}

}

package com.MyMoviePlan;

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(MyMoviePlanApplication.class);

}

}

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.PasswordEncoder;

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 {

}

}

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);

}

/\*

\* ============================= Show Controller ==========================

\*/

@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());

}

/\*

\* ============================= Movie Show Controller ==========================

\*/

@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"));

}

@PostMapping("{auditorium\_id}/show/{show\_id}/movie-show/add")

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

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

@PathVariable final int show\_id,

@RequestBody final MovieShowsEntity movieShow) {

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

final int movieId = movieShow.getMovieId();

movieShow.setShow(show);

movieShow.setMovieId(this.movie.findById(movieId).get().getId());

return this.movieShow.save(movieShow);

}

@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,

@RequestBody final MovieShowsEntity movieShow) {

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

movieShow.setShow(show);

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());

}

/\*

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

\*/

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

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

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

@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@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."));

}

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

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

public List<BookingEntity> allBookings(@PathVariable final int auditorium\_id,

@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id) {

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());

}

@PostMapping("{auditorium\_id}/show/{show\_id}/movie-show/{movie\_show\_id}/booking/add")

// @PreAuthorize("hasAuthority('WRITE')")

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

@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@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,

@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@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,

@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@PathVariable final int booking\_id) {

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());

}

}

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();

}

}

package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.MovieEntity;

import com.MyMoviePlan.exception.MovieNotFoundException;

import com.MyMoviePlan.repository.MovieRepository;

import com.MyMoviePlan.repository.MovieShowsRepository;

import lombok.AllArgsConstructor;

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

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

import java.util.\*;

@CrossOrigin

@RestController

@RequestMapping("/movie")

@AllArgsConstructor

public class MovieController {

private final MovieRepository movieRepository;

private final MovieShowsRepository movieShowsRepository;

@GetMapping({"/", "all"})

public List<MovieEntity> findAll() {

return movieRepository.findAll();

}

@GetMapping("{movie\_id}")

public MovieEntity findById(@PathVariable final int movie\_id) {

return movieRepository.findById(movie\_id)

.orElseThrow(() -> new MovieNotFoundException("Movie with movie\_id: " + movie\_id + " not found."));

}

@GetMapping("up-coming")

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

List<MovieEntity> movies;

List<MovieEntity> allMovies;

if (records.isPresent()) {

movies = new ArrayList<>();

allMovies = this.findAll();

movieShowsRepository.findFewUpComing(Integer.parseInt(records.get()))

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> (movie.getId() == m\_show.getMovieId() && movie.getRelease().getTime() > new Date().getTime()))

.findFirst().orElse(null)));

} else {

movies = new ArrayList<>();

allMovies = this.findAll();

movieShowsRepository.findAllUpComing()

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> movie.getId() == m\_show.getMovieId() && movie.getRelease().getTime() > new Date().getTime())

.findFirst().orElse(null)));

}

// return (movies.size() > 0 && !movies.contains(null)) ? movies : new ArrayList<>();

movies.removeAll(Collections.singletonList(null));

return movies;

}

@GetMapping("now-playing")

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

List<MovieEntity> movies;

List<MovieEntity> allMovies;

if (records.isPresent()) {

movies = new ArrayList<>();

allMovies = this.findAll();

movieShowsRepository.findFewNowPlaying(Integer.parseInt(records.get()))

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> movie.getId() == m\_show.getMovieId())

.findFirst().orElse(null)));

} else {

movies = new ArrayList<>();

allMovies = this.findAll();

movieShowsRepository.findAllNowPlaying()

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> movie.getId() == m\_show.getMovieId())

.findFirst().orElse(null)));

}

movies.removeAll(Collections.singletonList(null));

return movies;

}

@GetMapping("now-playing-up-coming")

public List<MovieEntity> nowPlayingAndUpComing() {

final List<MovieEntity> movies = new ArrayList<>();

final List<MovieEntity> allMovies = this.findAll();

movieShowsRepository.findAllNowPlayingAndUpComing()

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> movie.getId() == m\_show.getMovieId())

.findFirst().orElse(null)));

movies.removeAll(Collections.singletonList(null));

return movies;

}

@GetMapping("not-playing")

public List<MovieEntity> notPlaying() {

final List<MovieEntity> movies = new ArrayList<>();

final List<MovieEntity> allMovies = this.findAll();

movieShowsRepository.findAllNotPlaying()

.forEach(m\_show -> movies.add(allMovies.stream()

.filter(movie -> movie.getId() == m\_show.getMovieId())

.findFirst().orElse(null)));

movies.removeAll(Collections.singletonList(null));

return movies;

}

@PostMapping("add")

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

public MovieEntity saveMovie(@RequestBody final MovieEntity movie) {

return movieRepository.save(movie);

}

@PutMapping("update")

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

public MovieEntity updateMovie(@RequestBody final MovieEntity movie) {

return movieRepository.save(movie);

}

@DeleteMapping("delete/{movie\_id}")

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

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

movieRepository.deleteById(movie\_id);

}

}

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.get()));

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);

}

}

package com.MyMoviePlan.controller;

import com.MyMoviePlan.entity.BookingEntity;

import com.MyMoviePlan.entity.MovieShowsEntity;

import com.MyMoviePlan.entity.ShowEntity;

import com.MyMoviePlan.exception.BookingNotFoundException;

import com.MyMoviePlan.exception.MovieShowNotFoundException;

import com.MyMoviePlan.exception.ShowNotFoundException;

import com.MyMoviePlan.repository.BookingRepository;

import com.MyMoviePlan.repository.MovieRepository;

import com.MyMoviePlan.repository.MovieShowsRepository;

import com.MyMoviePlan.repository.ShowRepository;

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("/show")

@AllArgsConstructor

public class ShowController {

private final ShowRepository show;

private final MovieShowsRepository movieShow;

private final MovieRepository movie;

private final UserService service;

private final BookingRepository booking;

@GetMapping("{show\_id}")

public ShowEntity findShowById(@PathVariable final int show\_id) {

return this.show.findById(show\_id)

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

}

@GetMapping({"/", "all"})

public List<ShowEntity> findAllShows() {

return this.show.findAll();

}

@DeleteMapping("delete/{show\_id}")

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

public void deleteShow(@PathVariable final int show\_id) {

this.show.deleteById(show\_id);

}

/\*

\* ============================= Movie Show Controller ==========================

\*/

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

public List<MovieShowsEntity> findAllMovieShows(@PathVariable final int show\_id) {

return this.findShowById(show\_id)

.getMovieShows();

}

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

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

@PathVariable final int movie\_show\_id) {

return this.findShowById(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"));

}

@PostMapping("{show\_id}/movie-show/add")

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

public MovieShowsEntity saveMovieShow(@PathVariable final int show\_id,

@RequestBody final MovieShowsEntity movieShow) {

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

final int movieId = movieShow.getMovieId();

movieShow.setShow(show);

movieShow.setMovieId(this.movie.findById(movieId).get().getId());

return this.movieShow.save(movieShow);

}

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

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

public MovieShowsEntity updateMovieShow(@PathVariable final int show\_id,

@RequestBody final MovieShowsEntity movieShow) {

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

movieShow.setShow(show);

return this.movieShow.save(movieShow);

}

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

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

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

@PathVariable final int movie\_show\_id) {

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

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

}

/\*

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

\*/

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

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

public BookingEntity findBookingById(@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@PathVariable final int booking\_id) {

final MovieShowsEntity movieShow = this.findMovieShowById(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."));

}

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

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

public List<BookingEntity> allBookings(@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id) {

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

}

@PostMapping("{show\_id}/movie-show/{movie\_show\_id}/booking/add")

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

public BookingEntity saveBooking(@PathVariable final int show\_id,

@PathVariable final int movie\_show\_id,

@RequestBody final BookingEntity booking) {

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

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

booking.setUserId(this.service.findByMobile("8318152817").get().getId());

booking.setMovieShow(moveShow);

return this.booking.save(booking);

}

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

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

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

@PathVariable final int movie\_show\_id,

@RequestBody final BookingEntity booking) {

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

booking.setMovieShow(moveShow);

return this.booking.save(booking);

}

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

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

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

@PathVariable final int movie\_show\_id,

@PathVariable final int booking\_id) {

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

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

}

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);

}

}

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.ActorEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface ActorRepository extends JpaRepository<ActorEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.AuditoriumEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface AuditoriumRepository extends JpaRepository<AuditoriumEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.BookingEntity;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface BookingRepository extends JpaRepository<BookingEntity, Integer> {

List<BookingEntity> findAllByUserIdOrderByBookedOnAsc(final String userId);

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.MovieEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface MovieRepository extends JpaRepository<MovieEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.MovieShowsEntity;

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

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

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface MovieShowsRepository extends JpaRepository<MovieShowsEntity, Integer> {

// https://docs.spring.io/spring-data/commons/docs/current/reference/html/#repositories.limit-query-result

// https://stackoverflow.com/questions/11401229/how-to-use-select-distinct-with-random-function-in-postgresql

// https://stackoverflow.com/questions/32079084/how-to-find-distinct-rows-with-field-in-list-using-jpa-and-spring

// https://dev.to/golovpavel/make-a-request-with-sub-condition-for-child-list-via-spring-data-jpa-4inn

// https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

// @Query(value = "SELECT DISTINCT ON(movie\_id) \* FROM movie\_shows WHERE start >= CURRENT\_DATE", nativeQuery = true)

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_start > CURRENT\_DATE", nativeQuery = true)

List<MovieShowsEntity> findAllUpComing();

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_start <= CURRENT\_DATE AND ms.show\_end >= CURRENT\_DATE", nativeQuery = true)

List<MovieShowsEntity> findAllNowPlaying();

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_end >= CURRENT\_DATE", nativeQuery = true)

List<MovieShowsEntity> findAllNowPlayingAndUpComing();

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_end < CURRENT\_DATE", nativeQuery = true)

List<MovieShowsEntity> findAllNotPlaying();

// @Query("FROM MovieShowsEntity ms LEFT JOIN ms.bookings b WHERE ms.id = ?1 AND b.dateOfBooking = ?2")

// @Query(value = "SELECT \* FROM movie\_shows ms INNER JOIN bookings b ON ms.id = :id and b.date\_of\_booking = ':dateOfBooking'", nativeQuery = true)

// Optional<MovieShowsEntity> findByIdAndDateOfBooking(@Param("id") final int id, @Param("dateOfBooking") final String dateOfBooking);

// SELECT \* FROM (SELECT DISTINCT movie\_id FROM movie\_shows WHERE start > CURRENT\_DATE) ms ORDER BY random() LIMIT :records

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_start > CURRENT\_DATE LIMIT :records", nativeQuery = true)

List<MovieShowsEntity> findFewUpComing(@Param("records") final int records);

@Query(value = "SELECT DISTINCT ON(ms.movie\_id) \* FROM movie\_shows ms WHERE ms.show\_start <= CURRENT\_DATE AND ms.show\_end >= CURRENT\_DATE LIMIT :records", nativeQuery = true)

List<MovieShowsEntity> findFewNowPlaying(@Param("records") final int records);

// @Query(value = "SELECT ms.id, ms.show\_end, ms.movie\_id, ms.show\_start, ms.show\_id, ms.price\_id FROM movie\_shows ms INNER JOIN bookings b ON b.id = :bookingId", nativeQuery = true)

// MovieShowsEntity findByBookingId(@Param("bookingId") final int bookingId);

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.PaymentEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PaymentRepository extends JpaRepository<PaymentEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.PriceEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PriceRepository extends JpaRepository<PriceEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.ShowEntity;

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

import org.springframework.stereotype.Repository;

@Repository

public interface ShowRepository extends JpaRepository<ShowEntity, Integer> {

}

package com.MyMoviePlan.repository;

import com.MyMoviePlan.entity.UserEntity;

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

import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository

public interface UserRepository extends JpaRepository<UserEntity, String> {

Optional<UserEntity> findByEmail(final String email);

Optional<UserEntity> findByMobile(final String mobile);

}