Project Name:MyMoviePlan

Source code;

Make a spring starter project and add dependencies like: Lombok,MySql,Spring web and Spring Data JPA.

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.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 {

//
    }
}
```

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);
    }

    /*
     *      ============================== 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_i
d}")
    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/{movi
e_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_sho
w_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());

      }

}
```

MovieController.java

```
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);
	}
}
```

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.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);
    }
}
```

ShowController.java

```
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("8099531318").get().getId(
));
            booking.setMovieShow(moveShow);
            return this.booking.save(booking);
        }


@PutMapping("{show_id}/movie-show/{movie_show_id}/booking/upd
ate")
        @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());
	}
}
```

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);
	}
}
```

Make : **package com.MyMoviePlan.entity**

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;
    }
}
```

AuditoriumEntity.java

```
package com.MyMoviePlan.entity;

import lombok.*;

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

```
//@JsonIdentityInfo(generator =
ObjectIdGenerators.PropertyGenerator.class,
//                property = "id", scope = ShowEntity.class)
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
@Table(name = "auditoriums")
public class AuditoriumEntity implements Serializable {

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

    private String name;

    @Column(length = Integer.MAX_VALUE,
columnDefinition="TEXT")
    private String image;
```

```
    private String email;

    @Column(name = "customer_care_no")
    private String customerCareNo;

    private String address;

    @Column(name = "seat_capacity")
    private int seatCapacity;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ElementCollection
    @CollectionTable(name = "auditorium_facilities", joinColumns =
@JoinColumn(name = "auditorium_id"))
    @Column(name = "facility")
    private List<String> facilities;

    @ToString.Exclude
```

```
    @EqualsAndHashCode.Exclude
    @ElementCollection
    @CollectionTable(name = "auditorium_safeties", joinColumns =
@JoinColumn(name = "auditorium_id"))
    @Column(name = "safety")
    private List<String> safeties;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @JoinColumn(name = "auditorium_id", referencedColumnName =
"id")
    @OneToMany(targetEntity = ShowEntity.class, cascade =
CascadeType.ALL)
//      @JoinTable(name = "auditorium_shows",
//                  joinColumns = @JoinColumn(name = "auditorium_id",
unique = false),
//                  inverseJoinColumns = @JoinColumn(name =
"show_id", unique = false))
    private List<ShowEntity> shows;


    public AuditoriumEntity(String name, String image, String email,
String customerCareNo, String address,
```

```
                                                int seatCapacity, List<String>
facilities, List<String> safeties, List<ShowEntity> shows) {
            this.name = name;
            this.image = image;
            this.email = email;
            this.customerCareNo = customerCareNo;
            this.address = address;
            this.seatCapacity = seatCapacity;
            this.facilities = facilities;
            this.safeties = safeties;
            this.shows = shows;
        }

        public AuditoriumEntity setId(int id) {
            this.id = id;
            return this;
        }

        public AuditoriumEntity setName(String name) {
            this.name = name;
```

```
        return this;
    }

    public AuditoriumEntity setImage(String image) {
        this.image = image;
        return this;
    }

    public AuditoriumEntity setEmail(String email) {
        this.email = email;
        return this;
    }

    public AuditoriumEntity setCustomerCare(String customerCareNo)
{
        this.customerCareNo = customerCareNo;
        return this;
    }

    public AuditoriumEntity setAddress(String address) {
```

```
        this.address = address;
        return this;
    }

    public AuditoriumEntity setSeatCapacity(int seatCapacity) {
        this.seatCapacity = seatCapacity;
        return this;
    }

    public AuditoriumEntity setFacilities(List<String> facilities) {
        this.facilities = facilities;
        return this;
    }

    public AuditoriumEntity setSafeties(List<String> safeties) {
        this.safeties = safeties;
        return this;
    }

    public AuditoriumEntity setShows(List<ShowEntity> shows) {
```

```
            this.shows = shows;
            return this;
        }
}
```

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;
	}
}
```

BookingEntity.java

```
package com.MyMoviePlan.entity;

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

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

@Entity
@Data
```

```
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode
@Table(name = "bookings")
public class BookingEntity implements Serializable {

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

	private double amount;

	@Column(name = "total_seats")
	private int totalSeats;

	@Column(name = "booked_on")
	@Temporal(TemporalType.DATE)
	private Date bookedOn;

	@Column(name = "date_of_booking")
```

```
    @Temporal(TemporalType.DATE)
    private Date dateOfBooking;

    @Column(name = "user_id")
    private String userId;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ElementCollection
    @CollectionTable(name = "booked_seats", joinColumns =
@JoinColumn(name = "booking_id"))
    @Column(name = "seat_numbers")
    private List<String> seatNumbers;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToOne(targetEntity = PaymentEntity.class, cascade =
CascadeType.ALL)
    @JoinColumn(name = "payment_id")
    private PaymentEntity payment;
```

```
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToOne(targetEntity = BookingDetailsEntity.class, cascade =
CascadeType.ALL)
    @JoinColumn(name = "booking_details_id")
    private BookingDetailsEntity bookingDetails;


    @JsonIgnore
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(targetEntity = MovieShowsEntity.class)
    private MovieShowsEntity movieShow;


    public BookingEntity(double amount, int totalSeats, Date
bookedOn, Date dateOfBooking, List<String> seatNumbers,
                                    PaymentEntity payment, String userId,
MovieShowsEntity movieShow) {
        this.amount = amount;
        this.totalSeats = totalSeats;
        this.bookedOn = bookedOn;
```

```
        this.dateOfBooking = dateOfBooking;
        this.seatNumbers = seatNumbers;
        this.payment = payment;
        this.userId = userId;
        this.movieShow = movieShow;
    }

    public BookingEntity setMovieShow(MovieShowsEntity
movieShow) {
        this.movieShow = movieShow;
        return this;
    }

    public BookingEntity setId(int id) {
        this.id = id;
        return this;
    }

    public BookingEntity setAmount(double amount) {
        this.amount = amount;
```

```
        return this;
    }

    public BookingEntity setTotalSeats(int totalSeats) {
        this.totalSeats = totalSeats;
        return this;
    }

    public BookingEntity setStatus(Date bookedOn) {
        this.bookedOn = bookedOn;
        return this;
    }

    public BookingEntity setDateOfBooking(Date dateOfBooking) {
        this.dateOfBooking = dateOfBooking;
        return this;
    }

    public BookingEntity setSeatNumbers(List<String> seatNumbers) {
        this.seatNumbers = seatNumbers;
```

```
        return this;
    }

    public BookingEntity setPayment(PaymentEntity payment) {
        this.payment = payment;
        return this;
    }

    public BookingEntity setUserId(String userId) {
        this.userId = userId;
        return this;
    }
}
```

MovieEntity.java

```
package com.MyMoviePlan.entity;

import lombok.*;
```

```
import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import java.util.List;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode
@Table(name = "movies")
public class MovieEntity implements Serializable {

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

    private String name;

    @Column(length = Integer.MAX_VALUE, columnDefinition =
```

"TEXT")

private String image;

@Column(name = "bg_image", length = Integer.MAX_VALUE, columnDefinition="TEXT")

private String bgImage;

@Column(length = 9000)

private String story;

private String year;

private String duration;

private String caption;

@Column(name = "added_on")

@Temporal(TemporalType.DATE)

private Date addedOn;

```
    @Temporal(TemporalType.DATE)
    private Date release;

    private String language;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ElementCollection
    @CollectionTable(name = "movie_genres", joinColumns =
@JoinColumn(name = "movie_id"))
    @Column(name = "genre")
    private List<String> genres;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(targetEntity = ActorEntity.class, cascade =
CascadeType.ALL)
    @JoinColumn(name = "movie_id", referencedColumnName = "id")
    private List<ActorEntity> casts;
```

```
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(targetEntity = ActorEntity.class, cascade =
CascadeType.ALL)
    @JoinColumn(name = "movie_id", referencedColumnName = "id")
    private List<ActorEntity> crews;


    public MovieEntity(String name, String image, String bgImage,
String story, String year,
                       String duration, String caption, Date
addedOn, Date release, String language,
                       List<String> genres, List<ActorEntity>
casts, List<ActorEntity> crews) {
        this.name = name;
        this.image = image;
        this.bgImage = bgImage;
        this.story = story;
        this.year = year;
        this.duration = duration;
        this.caption = caption;
        this.addedOn = addedOn;
```

```
        this.release = release;
        this.language = language;
        this.genres = genres;
        this.casts = casts;
        this.crews = crews;
    }

    public MovieEntity setId(int id) {
        this.id = id;
        return this;
    }

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

    public MovieEntity setImage(String image) {
        this.image = image;
        return this;
```

```
}

public MovieEntity setBgImage(String bgImage) {
        this.bgImage = bgImage;
        return this;
}

public MovieEntity setStory(String story) {
        this.story = story;
        return this;
}

public MovieEntity setYear(String year) {
        this.year = year;
        return this;
}

public MovieEntity setDuration(String duration) {
        this.duration = duration;
        return this;
```

```
}

public MovieEntity setCaption(String caption) {
        this.caption = caption;
        return this;
}

public MovieEntity setAddedOn(Date addedOn) {
        this.addedOn = addedOn;
        return this;
}

public MovieEntity setRelease(Date release) {
        this.release = release;
        return this;
}

public MovieEntity setLanguages(String language) {
        this.language = language;
        return this;
```

```
	}

	public MovieEntity setGenres(List<String> genres) {
		this.genres = genres;
		return this;
	}

	public MovieEntity setCasts(List<ActorEntity> casts) {
		this.casts = casts;
		return this;
	}

	public MovieEntity setCrews(List<ActorEntity> crews) {
		this.crews = crews;
		return this;
	}
}
```

MovieShowsEntity.java

```
package com.MyMoviePlan.entity;
```

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

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

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode
@Table(name = "movie_shows")
public class MovieShowsEntity implements Serializable {

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

```
@Temporal(TemporalType.DATE)
@Column(name = "show_start")
private Date start;

@Temporal(TemporalType.DATE)
@Column(name = "show_end")
private Date end;

@Column(name = "movie_id")
private int movieId;

@JsonIgnore
@ToString.Exclude
@EqualsAndHashCode.Exclude
@ManyToOne(targetEntity = ShowEntity.class)
private ShowEntity show;

@ToString.Exclude
@EqualsAndHashCode.Exclude
@JoinColumn(name = "movie_show_id", referencedColumnName
```

```
= "id")
    @OneToMany(targetEntity = BookingEntity.class, cascade =
CascadeType.ALL)
//      @JoinTable(name = "movie_show_bookings",
//                  joinColumns = @JoinColumn(name =
"movie_show_id", unique = false),
//                  inverseJoinColumns = @JoinColumn(name =
"booking_id", unique = false))
    private List<BookingEntity> bookings;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToOne(targetEntity = PriceEntity.class, cascade =
CascadeType.ALL)
    @JoinColumn(name = "price_id")
    private PriceEntity price;


    public MovieShowsEntity(int id, Date start, Date end,
List<BookingEntity> bookings, int movieId) {
        this.id    = id;
        this.start = start;
```

```
        this.end = end;
        this.bookings = bookings;
        this.movieId = movieId;
    }

    public MovieShowsEntity setId(int id) {
        this.id = id;
        return this;
    }

    public MovieShowsEntity setStart(Date start) {
        this.start = start;
        return this;
    }

    public MovieShowsEntity setEnd(Date end) {
        this.end = end;
        return this;
    }
```

```
	public MovieShowsEntity setShow(ShowEntity show) {
		this.show = show;
		return this;
	}

	public MovieShowsEntity setMovieId(int movieId) {
		this.movieId = movieId;
		return this;
	}
}
```

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;
    }
}
```

UserEntity.java

```
package com.MyMoviePlan.entity;

import com.MyMoviePlan.model.UserRole;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;
```

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

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode
@Table(name = "users")
public class UserEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator
= "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    private String id;

    @Column(length = 50)
    private String name;
```

```
@Column(nullable = false, length = 50, unique = true)
private String email;

@Column(nullable = false, length = 10, unique = true)
private String mobile;

@Column(length = 60)
private String gender;

private String password;

private Boolean terms;

@Column(name = "is_account_non_expired")
private Boolean isAccountNonExpired;

@Column(name = "is_account_non_locked")
private Boolean isAccountNonLocked;

@Column(name = "is_credentials_non_expired")
```

```
    private Boolean isCredentialsNonExpired;


    @Column(name = "is_enabled")
    private Boolean isEnabled;


    @Column(name = "user_role", length = 20)
    @Enumerated(EnumType.STRING)
    private UserRole userRole;


    public UserEntity(String name, String email, String mobile, String
gender, String password, Boolean terms,
                      Boolean isAccountNonExpired, Boolean
isAccountNonLocked,
                      Boolean isCredentialsNonExpired, Boolean
isEnabled, UserRole userRole) {
        this.name = name;
        this.email = email;
        this.mobile = mobile;
        this.gender = gender;
        this.password = password;
        this.terms = terms;
```

```
        this.isAccountNonExpired = isAccountNonExpired;
        this.isAccountNonLocked = isAccountNonLocked;
        this.isCredentialsNonExpired = isCredentialsNonExpired;
        this.isEnabled = isEnabled;
        this.userRole = userRole;
    }

    public UserEntity setId(String id) {
        this.id = id;
        return this;
    }

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

    public UserEntity setEmail(String email) {
        this.email = email;
        return this;
```

```
}

public UserEntity setMobile(String mobile) {
        this.mobile = mobile;
        return this;
}

public UserEntity setGender(String gender) {
        this.gender = gender;
        return this;
}

public UserEntity setPassword(String password) {
        this.password = password;
        return this;
}

public UserEntity setActive(Boolean active) {
        terms = active;
        return this;
```

```
    }

    public UserEntity setAccountNonExpired(Boolean
accountNonExpired) {
        isAccountNonExpired = accountNonExpired;
        return this;
    }

    public UserEntity setAccountNonLocked(Boolean
accountNonLocked) {
        isAccountNonLocked = accountNonLocked;
        return this;
    }

    public UserEntity setCredentialsNonExpired(Boolean
credentialsNonExpired) {
        isCredentialsNonExpired = credentialsNonExpired;
        return this;
    }

    public UserEntity setEnabled(Boolean enabled) {
```

```
        isEnabled = enabled;
        return this;
    }

    public UserEntity setUserRole(UserRole userRole) {
        this.userRole = userRole;
        return this;
    }

    public UserEntity setTerms(Boolean terms) {
        this.terms = terms;
        return this;
    }

}
```

Make: package com.MyMoviePlan.exception

AuditoriumNotFoundException.java

```
package com.MyMoviePlan.exception;
```

```
public class AuditoriumNotFoundException extends RuntimeException {

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

BookingNotFoundException.java

```
package com.MyMoviePlan.exception;

public class BookingNotFoundException extends RuntimeException{

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

GlobalExceptionHandler.java

```
package com.MyMoviePlan.exception;

import com.MyMoviePlan.model.HttpResponse;
```

```
import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import
org.springframework.web.servlet.mvc.method.annotation.ResponseEnt
ityExceptionHandler;


import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


@ControllerAdvice

public class GlobalExceptionHandler extends
ResponseEntityExceptionHandler {


    @ExceptionHandler(Exception.class)

    public ResponseEntity<HttpResponse> handleException(final
Exception exception,


final HttpServletRequest request,


final HttpServletResponse response) {
```

```
        Integer statusCode = (Integer) request
                .getAttribute("javax.servlet.error.status_code");

        final int status = response.getStatus();

        final String exceptionMessage = exception.getMessage();
        if (statusCode == null || statusCode == 0) {
            statusCode = status;
            if
(HttpStatus.valueOf(status).getReasonPhrase().equals("OK"))
                statusCode = 403;
        }

        final HttpStatus httpStatus = HttpStatus.valueOf(statusCode);

        final HttpResponse httpResponse =
                new HttpResponse(statusCode,
httpStatus.getReasonPhrase(), exceptionMessage);
        return new ResponseEntity<HttpResponse>(httpResponse,
httpStatus);
    }
```

```
}
```

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);
	}
}
```

ShowNotFoundException.java

```
package com.MyMoviePlan.exception;

public class ShowNotFoundException extends RuntimeException{

	public ShowNotFoundException(String message) {
		super(message);
	}
}
```

UnAuthorizedException.java

```
package com.MyMoviePlan.exception;

public class UnAuthorizedException extends RuntimeException{

	public UnAuthorizedException(String message) {
		super(message);
	}



}
```

UserNotFoundException.java

```
package com.MyMoviePlan.exception;


public class UserNotFoundException extends RuntimeException {


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

Make: package com.MyMoviePlan.filter

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.UsernamePasswordAuthe
```

```
nticationToken;

import
org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import
org.springframework.security.web.authentication.WebAuthenticationD
etailsSource;

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,
                                    HttpServletResponse response,
                                    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);


                    if (jwtUtil.validateToken(token, userDetails)) {
                        UsernamePasswordAuthenticationToken
authenticationToken
                                = new
UsernamePasswordAuthenticationToken(userDetails,
                                null, userDetails.getAuthorities());


                        authenticationToken.setDetails(
                                new
WebAuthenticationDetailsSource().buildDetails(request)
                        );

SecurityContextHolder.getContext().setAuthentication(authenticationT
```

```
oken);
                        }
                    }
                    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();
            }
        }
}
```

Make: package com.MyMoviePlan.model

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;
}
```

HttpResponse.java

```
package com.MyMoviePlan.model;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class HttpResponse {

    private int statusCode;
```

```
	private String error;
	private String message;

	public String covertToJson() throws JsonProcessingException {
		if (this == null)
			return null;

		ObjectMapper mapper = new ObjectMapper();
		mapper.registerModule(new JavaTimeModule());

mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);

		return mapper.writeValueAsString(this);
	}
}
```

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;
}
```

Token.java

```
package com.MyMoviePlan.model;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class Token {

	private String token;
}
```

UserPermission.java

```
package com.MyMoviePlan.model;


import lombok.AllArgsConstructor;
import lombok.Getter;
```

```
@Getter
@AllArgsConstructor
public enum UserPermission {
	READ,
	WRITE,
	UPDATE,
	DELETE
}
```

UserRole.java

```
package com.MyMoviePlan.model;

import lombok.AllArgsConstructor;
import lombok.Getter;

import java.util.Arrays;
import java.util.List;

import static com.MyMoviePlan.model.UserPermission.*;

@Getter
```

```
@AllArgsConstructor
public enum UserRole {

    ROLE_USER(Arrays.asList(READ)),
    ROLE_MANAGER(Arrays.asList(READ, WRITE)),
    ROLE_ADMIN(Arrays.asList(READ, WRITE, UPDATE)),
    ROLE_SUPER_ADMIN(Arrays.asList(READ, WRITE, UPDATE,
DELETE));

    private final List<UserPermission> permissions;
}
```

ActorRepository.java

```
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> {

}
```

AuditoriumRepository.java

```
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> {

}
```

BookingRepository.java

```
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);

}
```

MovieRepository.java

```
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> {

}
```

MovieShowsRepository.java

```
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/h
tml/#repositories.limit-query-result

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

	//
https://stackoverflow.com/questions/32079084/how-to-find-distinct-r
```

ows-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);

}
```

PaymentRepository.java

```
ackage 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> {

}
```

PriceRepository.java

```
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> {

}
```

ShowRepository.java

```
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> {
}
```

UserRepository.java

```
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);
}
```

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.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationPr
ovider;
import
org.springframework.security.config.annotation.authentication.builders
.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.
EnableGlobalMethodSecurity;
import
```

```
org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

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

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

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.CorsConfigurationSource;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;


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);
        }

//                          .antMatchers(HttpMethod.POST,
"/user/authenticate", "/user/sign-up")
//                          .permitAll()
```

```
//                      .antMatchers(HttpMethod.PUT,
"/user/forgot-password")
//                      .permitAll()
//                      .antMatchers(HttpMethod.GET,
"/auditorium/**", "/movie/**", "/show/**", "/user/check/**")
//                      .permitAll()


    @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(passwordEncoder);

authenticationProvider.setUserDetailsService(userDetailsService);
        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();
        }
}
```

ApplicationUserDetails.java

```
package com.MyMoviePlan.security;
```

```
import com.MyMoviePlan.entity.UserEntity;
import lombok.AllArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

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

@AllArgsConstructor
public class ApplicationUserDetails implements UserDetails {

    private final UserEntity user;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        final List<SimpleGrantedAuthority> authorities =
```

```
user.getUserRole()
                        .getPermissions()
                        .stream()
                        .map(permission -> new
SimpleGrantedAuthority(permission.name()))
                        .collect(Collectors.toList());
        authorities.add(new
SimpleGrantedAuthority(user.getUserRole().name()));
        return authorities;
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getEmail();
    }
```

```
@Override
public boolean isAccountNonExpired() {
	return user.getIsAccountNonExpired();
}

@Override
public boolean isAccountNonLocked() {
	return user.getIsAccountNonLocked();
}

@Override
public boolean isCredentialsNonExpired() {
	return user.getIsCredentialsNonExpired();
}

@Override
public boolean isEnabled() {
	return user.getIsEnabled();
}
```

}

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.UserDetails;

import
org.springframework.security.core.userdetails.UserDetailsService;

import
org.springframework.security.core.userdetails.UsernameNotFoundExce
ption;

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);
    }
}
```

ServletInitializer.java

```
package com.MyMoviePlan;


import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitiali
zer;


public class ServletInitializer extends SpringBootServletInitializer {


    @Override
    protected SpringApplicationBuilder
```

```
configure(SpringApplicationBuilder application) {
		return application.sources(MyMoviePlanApplication.class);
	}

}
```

MyMoviePlanApplicationTests.java

```
package com.MyMoviePlan;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class MyMoviePlanApplicationTests {

	@Test
	void contextLoads() {
	}

}
```