



Dokumentacja projektu Kina

Wykonanego w technologii Spring
oraz React

Autor: Paweł Pauszek

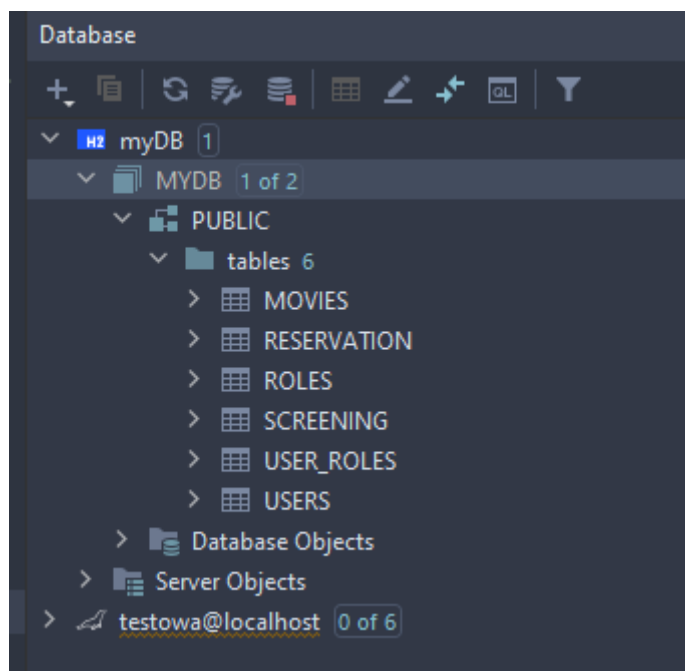
Celem projektu było stworzenie aplikacji full stack z wykorzystaniem Springa jako backend oraz Reacta jako frontend. Z racji tego, iż projekt jest ogromny i dokumentacja każdej części zajęła by mi ponad 100 stron, postanowiłem udokumentować postęp swojej pracy w mniej szczegółowy sposób.

Baza danych postawiona jest na H2. Zarządzanie bazą odbywa się poprzez JPA w Springu.

```
1 #spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
2 # App Properties
3 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
4 app.jwtSecret= MySecret
5 app.jwtExpirationMs= 86400000
6 spring.datasource.url=jdbc:h2:file:./data/myDB
7 spring.datasource.username=
8 spring.datasource.password=
9 spring.datasource.driverClassName=org.h2.Driver
10 spring.jpa.show-sql=true
11 spring.jpa.hibernate.ddl-auto=update
```

Jako sposób autoryzacji postanowiłem posłużyć się tokenami JWT. Token generowany jest w backendzie i zwracany do front endu, gdzie zapisuję go do localStorage (można także zapisać go do cookies, by miał termin ważności).

Tak prezentują się encje w bazie danych:



Poniżej zamieszczam jak wyglądają entity w JPA

Tabela User:

```
User.java
11  @Entity
12  @Table(name = "users",
13        uniqueConstraints = {
14            @UniqueConstraint(columnNames = "username"),
15            @UniqueConstraint(columnNames = "email")
16        })
17  public class User {
18      2 usages
19      @Id
20      @GeneratedValue(strategy = GenerationType.IDENTITY)
21      private Long id;
22      3 usages
23      @NotBlank
24      @Size(max = 20)
25      private String username;
26      3 usages
27      @NotBlank
28      @Size(max = 50)
29      @Email
30      private String email;
31      3 usages
32      @NotBlank
33      @Size(max = 120)
34      private String password;
35      2 usages
36      @ManyToMany(fetch = FetchType.LAZY)
37      @JoinTable( name = "user_roles",
38                joinColumns = @JoinColumn(name = "user_id"),
39                inverseJoinColumns = @JoinColumn(name = "role_id"))
40      private Set<Role> roles = new HashSet<>();
41      public User() {
42      }
43      public User(String username, String email, String password) {
44          this.username = username;
45          this.email = email;
46          this.password = password;
47      }
48      public Long getId() {
49          return id;
50      }
51  }
```

Tabela Role:

```
1 package com.kino.springjwt.models;
2
3 import javax.persistence.*;
4
5 12 usages  no.body
6 @Entity
7 @Table(name = "roles")
8 public class Role {
9     2 usages
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Integer id;
13
14     3 usages
15     @Enumerated(EnumType.STRING)
16     @Column(length = 20)
17     private ERole name;
18
19     no usages  no.body
20     public Role() {
21
22     }
23
24     no usages  no.body
25     public Role(ERole name) { this.name = name; }
26
27     no.body
28     public Integer getId() { return id; }
29
30     no.body
31     public void setId(Integer id) { this.id = id; }
32
33     no.body
34     public ERole getName() { return name; }
35
36     no.body
37     public void setName(ERole name) { this.name = name; }
38
39 }
```

Tabela ERole (enum):

```
1 package com.kino.springjwt.models;
2
3 11 usages  no.body
4 public enum ERole {
5     2 usages
6     ROLE_USER,
7     1 usage
8     ROLE_MODERATOR,
9     1 usage
10    ROLE_ADMIN
11 }
12
```

Tabela Movie:

```
Movie.java
5  @Entity
6  @Table(name = "MOVIES")
7  public class Movie {
8      2 usages
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Column(name = "ID", nullable = false)
12     private Integer id;
13
14     2 usages
15     @Column(name = "NAME")
16     private String name;
17
18     2 usages
19     @Column(name = "DESCRIPTION")
20     private String description;
21
22     2 usages
23     @Column(name = "RATING")
24     private Double rating;
25
26     2 usages
27     @Column(name = "PEGI")
28     private Integer pegi;
29
30     2 usages
31     @Column(name = "IMAGE")
32     private String image;
33
34     no body
35     public Integer getId() { return id; }
36
37     no body
38     public void setId(Integer id) { this.id = id; }
39
40     no body
41     public String getName() { return name; }
42
43     no body
44     public void setName(String name) { this.name = name; }
45
46     no body
47     public String getDescription() { return description; }
48
49     no body
50     public void setDescription(String description) { this.description = description; }
51
52     no body
53     public Double getRating() { return rating; }
54
55     no body
56     public void setRating(Double rating) { this.rating = rating; }
57
58
59
```

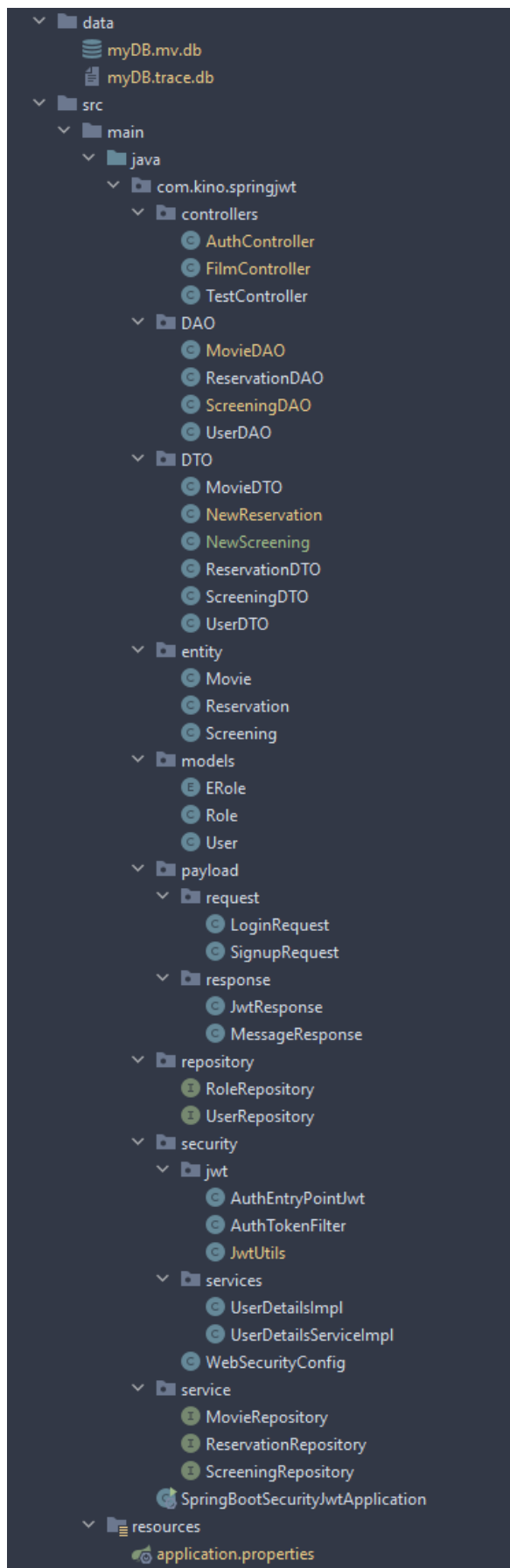
Tabela Screening (Seansów)

```
Screening.java
8  @Entity
9  @Table(name = "SCREENING")
10 public class Screening {
11     2 usages
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Column(name = "ID_SCREENING", nullable = false)
15     private Integer id;
16
17     2 usages
18     @ManyToOne(fetch = FetchType.EAGER)
19     @JoinColumn(name = "ID_MOVIE")
20     private Movie idMovie;
21
22     2 usages
23     @Column(name = "PRICE")
24     private Double price;
25
26     2 usages
27     @Column(name = "DATE")
28     private Timestamp date;
29
30     2 usages
31     @Column(name = "SEATS")
32     private Integer seats;
33
34     no usages  no.body
35     public Screening(Screening screening) {
36     }
37
38     2 usages  no.body
39     public Screening() {
40     }
41
42     no.body
43     public Integer getId() { return id; }
44
45     no.body
46     public void setId(Integer id) { this.id = id; }
47
48     1 usage  no.body
49     public Movie getIdMovie() { return idMovie; }
50
51     3 usages  no.body
52     public void setIdMovie(Movie idMovie) { this.idMovie = idMovie; }
53
54     no.body
55     public Double getPrice() { return price; }
56
57     no.body
58     public void setPrice(Double price) { this.price = price; }
59 }
```

Tabela Reservation:

```
Reservation.java
8  @Entity
9  @Table(name = "RESERVATION")
10 public class Reservation {
11     2 usages
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Column(name = "ID_RESERVATION", nullable = false)
15     private Integer id;
16
17     2 usages
18     @ManyToOne(fetch = FetchType.LAZY)
19     @Cascade(org.hibernate.annotations.CascadeType.ALL)
20     @JoinColumn(name = "ID_SCREENING")
21     private Screening idScreening;
22
23     2 usages
24     @Column(name = "NAME")
25     private String name;
26
27     2 usages
28     @Column(name = "LAST_NAME")
29     private String lastName;
30
31     2 usages
32     @ManyToOne(fetch = FetchType.LAZY)
33     @JoinColumn(name = "USER_ID")
34     private User userId;
35
36     2 usages
37     @Column(name = "SEAT_NUMBER")
38     private Integer seatNumber;
39
40     1 no.body
41     public Integer getId() { return id; }
42
43     1 no.body
44     public void setId(Integer id) { this.id = id; }
45
46     1 usage 1 no.body
47     public Screening getIdScreening() { return idScreening; }
48
49     1 usage 1 no.body
50     public void setIdScreening(Screening idScreening) {
51         this.idScreening = idScreening;
52     }
53
54     1 no.body
55     public String getName() { return name; }
56
57     1 no.body
58     public void setName(String name) { this.name = name; }
59 }
```

Hierarchia projektu:



Projekt posiada do ważnych encji repozytoria, Data Transfer Object (klasy utworzone na wzór oryginalnej tabeli, które służą do udostępniania/modyfikowania/dodawania danych z zewnątrz) oraz Data Access Object (dostęp do danych np. wyszukuje po userze itp.)

Projekt posiada dwa główne kontrolery:

AuthController – odpowiadający za autoryzację

FilmController – odpowiada za wszystko inne

Zwracanie wiadomości po weryfikacji tokenu JWT ze strony backendu:

```
AuthEntryPointJwt.java
1 package com.kino.springjwt.security.jwt;
2
3 import ...
4
19
20 3 usages no.body
21 @Component
22 public class AuthEntryPointJwt implements AuthenticationEntryPoint {
23
24     1 usage
25     private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);
26
27     no usages no.body
28     @Override
29     public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException)
30         throws IOException, ServletException {
31         logger.error("Unauthorized error: {}", authException.getMessage());
32
33         response.setContentType(MediaType.APPLICATION_JSON_VALUE);
34         response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
35
36         final Map<String, Object> body = new HashMap<>();
37         body.put("status", HttpServletResponse.SC_UNAUTHORIZED);
38         body.put("error", "Unauthorized");
39         body.put("message", authException.getMessage());
40         body.put("path", request.getServletPath());
41
42         final ObjectMapper mapper = new ObjectMapper();
43         mapper.writeValue(response.getOutputStream(), body);
44     }
45 }
```

Tworzenie tokenu JWT:

```
JwtUtils.java
1 package com.kino.springjwt.security.jwt;
2
3 import ...
13
14 4 usages 1 no.body *
15 @Component
16 public class JwtUtils {
17     5 usages
18     private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);
19
20     3 usages
21     @Value("MySecret")
22     private String jwtSecret;
23
24     1 usage
25     @Value("${app.jwtExpirationMs}")
26     private int jwtExpirationMs;
27
28     1 usage 1 no.body
29     @
30     public String generateJwtToken(Authentication authentication) {
31
32         UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();
33
34         return Jwts.builder()
35             .setSubject((userPrincipal.getUsername()))
36             .setIssuedAt(new Date())
37             .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
38             .signWith(SignatureAlgorithm.HS512, jwtSecret)
39             .compact();
40     }
41
42     1 usage 1 no.body
43     public String getUsernameFromJwtToken(String token) {
44         return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
45     }
46
47     1 usage 1 no.body
48     public boolean validateJwtToken(String authToken) {
49         try {
50             Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
51             return true;
52         } catch (SignatureException e) {
53             logger.error("Invalid JWT signature: {}", e.getMessage());
54         } catch (MalformedJwtException e) {
55             logger.error("Invalid JWT token: {}", e.getMessage());
56         } catch (ExpiredJwtException e) {
57             logger.error("JWT token is expired: {}", e.getMessage());
58         } catch (UnsupportedJwtException e) {
59             logger.error("JWT token is unsupported: {}", e.getMessage());
60         } catch (IllegalArgumentException e) {
61             logger.error("JWT claims string is empty: {}", e.getMessage());
62         }
63
64         return false;
65     }
66 }
```

Jak można zobaczyć, token JWT w moim przypadku zawiera zakodowany login użytkownika oraz czas, przez jaki token jest ważny.

Sprawdzanie poprawności tokenu:

```
AuthTokenFilter.java

21 public class AuthTokenFilter extends OncePerRequestFilter {
22     @Autowired
23     private JwtUtils jwtUtils;
24
25     @Autowired
26     private UserDetailsServiceImpl userDetailsService;
27
28     private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);
29
30     @Override
31     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
32         throws ServletException, IOException {
33         try {
34             String jwt = parseJwt(request);
35             if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
36                 String username = jwtUtils.getUserNameFromJwtToken(jwt);
37
38                 UserDetails userDetails = userDetailsService.loadUserByUsername(username);
39                 UsernamePasswordAuthenticationToken authentication =
40                     new UsernamePasswordAuthenticationToken(
41                         userDetails,
42                         null,
43                         userDetails.getAuthorities());
44                 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
45
46                 SecurityContextHolder.getContext().setAuthentication(authentication);
47             }
48         } catch (Exception e) {
49             logger.error("Cannot set user authentication: {}", e);
50         }
51
52         filterChain.doFilter(request, response);
53     }
54
55     private String parseJwt(HttpServletRequest request) {
56         String headerAuth = request.getHeader("Authorization");
57
58         if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
59             return headerAuth.substring(7, headerAuth.length());
60         }
61
62         return null;
63     }
64 }
65
```

Zarządzanie sesją oraz dekodowanie hasła:

```
WebSecurityConfig.java
24  @Configuration
25  @EnableGlobalMethodSecurity(
26  prePostEnabled = true)
27  public class WebSecurityConfig {
28      1 usage
29      @Autowired
30      UserDetailsServiceImpl userDetailsService;
31
32      1 usage
33      @Autowired
34      private AuthEntryPointJwt unauthorizedHandler;
35
36      1 usage  no.body
37      @Bean
38      public AuthTokenFilter authenticationJwtTokenFilter() { return new AuthTokenFilter(); }
39
40      1 usage  no.body
41      @Bean
42      public DaoAuthenticationProvider authenticationProvider() {
43          DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
44
45          authProvider.setUserDetailsService(userDetailsService);
46          authProvider.setPasswordEncoder(passwordEncoder());
47
48          return authProvider;
49      }
50
51      no usages  no.body
52      @Bean
53      public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
54          return authConfig.getAuthenticationManager();
55      }
56
57      1 usage  no.body
58      @Bean
59      public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
60
61      no usages  no.body
62      @Bean
63      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
64          http.cors().and().csrf().disable().authorizeRequests().antMatchers("/api/auth/**").permitAll()
65              .antMatchers("/api/test/**").permitAll().anyRequest().authenticated();
66
67          http.authenticationProvider(authenticationProvider());
68
69          http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
70
71          return http.build();
72      }
73  }
74  }
```

Przykładowe repozytorium Filmów:

```
MovieRepository.java
1 package com.kino.springjwt.service;
2
3 import ...
4
5
6 6 usages no.body
7
8
9 @Repository
10 public interface MovieRepository extends JpaRepository<Movie, Integer> {
11
12 6 usages no.body
13     Optional<Movie> findById(Integer id);
14
15 1 usage no.body
16     Optional<Movie> findAllByNameContainingIgnoreCase(String word);
17 }
18
```

W tym przypadku, stworzyłem metodę szukającą filmy po id oraz po słowach kluczowych, znajdujących się w tytule filmu.

Z racji prostoty i objętości innych repozytoriów, nie będę ich pokazywał ani opisywał, udostępnię tylko kod źródłowy.

Przykładowy serwis, szukający detali użytkownika:

```
13 4 usages no.body
14 @Service
15 public class UserDetailsServiceImpl implements UserDetailsService {
16 1 usage
17     @Autowired
18     UserRepository userRepository;
19
20 1 usage no.body
21     @Override
22     @Transactional
23     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
24         User user = userRepository.findByUsername(username).orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));
25         return UserDetailsImpl.build(user);
26     }
27 }
```


Serwis zarządzający dostępem. Sprawdza on, do czego użytkownik ma dostęp.

```
UserDetailsImpl.java
15 public class UserDetailsImpl implements UserDetails {
    no usages
16     private static final long serialVersionUID = 1L;
17
    4 usages
18     private Long id;
19
    2 usages
20     private String username;
21
    2 usages
22     private String email;
23
    2 usages
24     @JsonIgnore
25     private String password;
26
    2 usages
27     private Collection<? extends GrantedAuthority> authorities;
28
    1 usage no.body
29     public UserDetailsImpl(Long id, String username, String email, String password,
30         Collection<? extends GrantedAuthority> authorities) {
31         this.id = id;
32         this.username = username;
33         this.email = email;
34         this.password = password;
35         this.authorities = authorities;
36     }
37
    no.body
38     @ static UserDetailsImpl build(User user) {
39         List<GrantedAuthority> authorities = user.getRoles().stream() Stream<Role>
40             .map(role -> new SimpleGrantedAuthority(role.getName().name())) Stream<SimpleGrantedAuthority>
41             .collect(Collectors.toList());
42
43         return new UserDetailsImpl(
44             user.getId(),
45             user.getUsername(),
46             user.getEmail(),
47             user.getPassword(),
48             authorities);
49     }
50
    2 usages no.body
51     @Override
52     public Collection<? extends GrantedAuthority> getAuthorities() { return authorities; }
53
    no.body
54
55     no.body
56     public Long getId() { return id; }
57
58     no.body
59
60     no.body
61     public String getEmail() { return email; }
62
63     no.body
64     @Override
```

Przykładowe obiekty DTO Seansów:

```
ScreeningDTO.java
7  public class ScreeningDTO {
8
9      private Integer id;
10     private MovieDTO idMovie;
11     private Double price;
12     private Timestamp date;
13     private Integer seats;
14
15     public ScreeningDTO() {
16
17     }
18
19     public ScreeningDTO(Integer id, MovieDTO idMovie, Double price, Timestamp date, Integer seats) {
20         this.id = id;
21         this.idMovie = idMovie;
22         this.price = price;
23         this.date = date;
24         this.seats = seats;
25     }
26
27     @ public ScreeningDTO(Screening screening) {
28         this.id = screening.getId();
29         this.idMovie = new MovieDTO(screening.getIdMovie());
30         this.price = screening.getPrice();
31         this.date = screening.getDate();
32         this.seats = screening.getSeats();
33     }
34
35     public Integer getId() { return id; }
36
37     public void setId(Integer id) { this.id = id; }
38
39     public MovieDTO getIdMovie() { return idMovie; }
40
41     public void setIdMovie(MovieDTO idMovie) { this.idMovie = idMovie; }
42
43     public Double getPrice() { return price; }
44
45     public void setPrice(Double price) { this.price = price; }
```

Ta klasa służy do wypisywania danych związanych z seansami

Klasa wykorzystywana do dodawania seansów:

```
NewScreening.java
5 public class NewScreening {
6
7     3 usages
    private Integer id;
8     3 usages
    private Integer idMovie;
9     3 usages
    private Timestamp date;
10    3 usages
    private Double price;
11    3 usages
    private Integer seats;
12
13
14    no usages new *
    public NewScreening(){
15
16    }
17
18    no usages new *
    public NewScreening(Integer id, Integer idMovie, Timestamp date, Double price, Integer seats) {
19        this.id = id;
20        this.idMovie = idMovie;
21        this.date = date;
22        this.price = price;
23        this.seats = seats;
24    }
25
26    new *
    public Integer getId() { return id; }
29
30    new *
    public void setId(Integer id) {
31        this.id = id;
32    }
33
34    3 usages new *
    public Integer getIdMovie() { return idMovie; }
37
38    no usages new *
    public void setIdMovie(Integer idMovie) { this.idMovie = idMovie; }
41
42    new *
    public Timestamp getDate() { return date; }
45
46    new *
    public void setDate(Timestamp date) { this.date = date; }
49
50    new *
    public Double getPrice() { return price; }
53
54    new *
    public void setPrice(Double price) { this.price = price; }
57
```

Jak można zauważyć lekko się od siebie klasy różnią. Wynika to z tego, że przy np. dodawaniu do bazy, to nie dodajemy obiektu Film tylko id Filmu a przy wypisywaniu danych zależy nam na jak największej ilości informacji, więc wypisanie obiektu jest ok.

Pozostałe DTO adekwatnie do powyższych przykładów.

Data Access Object:

Z racji tego, że klasy DAO posiadają proste funkcje, tłumaczące się z samych siebie lub też z ich nazewnictwa oraz są obszerne, postanowiłem ich nie opisywać

MovieDAO:

```
MovieDAO.java
8   import java.util.List;
9   import java.util.Optional;
10  import java.util.stream.Collectors;
11
12  @Service
13  public class MovieDAO {
14
15      private final MovieRepository movieRepository;
16
17      public MovieDAO(MovieRepository movieRepository) { this.movieRepository = movieRepository; }
18
19      public List<MovieDTO> getAllMovies() {
20          return movieRepository.findAll().stream().map(MovieDTO::new).collect(Collectors.toList());
21      }
22
23      public MovieDTO getMovieById(Integer id) {
24          return new MovieDTO(movieRepository.findById(id).get());
25      }
26
27      public Movie save(Movie movie) { return movieRepository.save(movie); }
28
29      public void delete(Integer id) { movieRepository.deleteById(id); }
30
31      public void addMovie(MovieDTO movieDTO) {
32          Movie movie = new Movie();
33          movie.setDescription(movieDTO.getDescription());
34          movie.setName(movieDTO.getName());
35          movie.setImage(movieDTO.getImage());
36          movie.setPegi(movieDTO.getPegi());
37          movie.setRating(movieDTO.getRating());
38          movieRepository.save(movie);
39      }
40
41      public List<MovieDTO> getMoviesByWord(String word) {
42          return movieRepository.findAllByNameContainingIgnoreCase(word).map(MovieDTO::new).stream().collect(Collectors.toList());
43      }
44
45      public Movie getMovieByIdMovie(Integer id) { return movieRepository.findById(id).get(); }
46  }
```

ScreeningDAO (seanse):

```
ScreeningDAO.java
8  import org.springframework.stereotype.Service;
9
10 import java.util.List;
11
12 5 usages 1 no.body *
12  @Service
13  public class ScreeningDAO {
14
15      8 usages
15      private final ScreeningRepository screeningRepository;
16
17      2 usages
16      private final MovieRepository movieRepository;
17
18      no usages 1 no.body *
18      public ScreeningDAO(ScreeningRepository screeningRepository, MovieRepository movieRepository) {
19          this.screeningRepository = screeningRepository;
20          this.movieRepository = movieRepository;
21      }
22
23      1 usage 1 no.body
23      public List<Screening> getAllScreenings() { return screeningRepository.findAll(); }
24
25      3 usages 1 no.body
27      public Screening getScreeningById(Integer id) { return screeningRepository.findById(id).get(); }
28
29      2 usages 1 no.body
31      public Screening save(Screening screening) { return screeningRepository.save(screening); }
32
33      no usages 1 no.body
35      public void delete(Integer id) { screeningRepository.deleteById(id); }
36
37      no usages 1 no.body
39      public long countScreenings() { return screeningRepository.count(); }
40
41      1 usage 1 no.body
43      public List<ScreeningDTO> getScreeningByFilmyId(Integer id) {
44          return screeningRepository.findAllByIdMovieId(id).stream().map(ScreeningDTO::new).collect(java.util.stream.Collectors.toList());
45      }
46
47      no usages new *
47      @Transactional
47      public void addScreening(NewScreening newScreening) {
48          Screening screening = new Screening();
49          screening.setIdMovie(movieRepository.findById(newScreening.getIdMovie()).get());
50          screening.setDate(newScreening.getDate());
51          screening.setPrice(newScreening.getPrice());
52          screening.setSeats(newScreening.getSeats());
53          screeningRepository.save(screening);
54      }
55  }
56
57  |
```

UserDAO:

```
UserDAO.java
1  package com.kino.springjwt.DAO;
2
3  import com.kino.springjwt.models.User;
4  import com.kino.springjwt.repository.UserRepository;
5  import org.springframework.stereotype.Service;
6
7  @Service
8  public class UserDAO {
9      private final UserRepository userRepository;
10
11     public UserDAO(UserRepository userRepository) { this.userRepository = userRepository; }
12
13     public UserRepository getUserRepository() { return userRepository; }
14
15     public User findById(Integer id) {
16         return userRepository.findById(Long.valueOf(id)).get();
17     }
18
19 }
20
21
22
23
24
```

RezerwacjeDAO:

```
ReservationDAO.java
8
9 import java.util.List;
10
11 @Service
12 public class ReservationDAO {
13
14     6 usages
15     private final ReservationRepository reservationRepository;
16     2 usages
17     private final ScreeningDAO screeningDAO;
18     1 usage
19     private UserDao userDao;
20
21     ± no.body
22     public ReservationDAO(ReservationRepository reservationRepository, ScreeningDAO screeningDAO) {
23         this.reservationRepository = reservationRepository;
24         this.screeningDAO = screeningDAO;
25     }
26
27     ± no.body
28     public void add(NewReservation newReservation) {
29         Reservation reservation = new Reservation();
30         reservation.setName(newReservation.getName());
31         reservation.setLastName(newReservation.getLastName());
32         reservation.setUserId(userDAO.findById(Math.toIntExact(newReservation.getUserId())));
33         reservation.setSeatNumber(newReservation.getSeatNumber());
34         reservation.setScreeningId(screeningDAO.getScreeningById(newReservation.getScreeningId()));
35         reservationRepository.save(reservation);
36     }
37
38     no usages ± no.body
39     public void delete(Integer id) {
40         reservationRepository.deleteById(id);
41     }
42
43     1 usage ± no.body
44     public void save(Reservation reservation) {
45         reservationRepository.save(reservation);
46     }
47
48     1 usage ± no.body
49     public List<ReservationDTO> getReservationsByScreening(int id) {
50         return reservationRepository.findAllByIdScreeningId(id).stream().map(ReservationDTO::new).collect(java.util.stream.Collectors.toList());
51     }
52
53     1 usage ± no.body
54     public List<ReservationDTO> getReservationsByUser(int id) {
55         return reservationRepository.findAllByIdUserId((long) id).stream().map(ReservationDTO::new).collect(java.util.stream.Collectors.toList());
56     }
57 }
```

Kontrolery:

Kontroler autoryzacji:

```
AuthController.java
36 @CrossOrigin(origins = "*", maxAge = 3600)
37 @RestController
38 @RequestMapping("/api/auth")
39 public class AuthController {
40     1 usage
41     @Autowired
42     AuthenticationManager authenticationManager;
43
44     3 usages
45     @Autowired
46     UserRepository userRepository;
47
48     4 usages
49     @Autowired
50     RoleRepository roleRepository;
51
52     1 usage
53     @Autowired
54     PasswordEncoder encoder;
55
56     1 usage
57     @Autowired
58     JwtUtils jwtUtils;
59
60     no usages  no.body *
61     @PostMapping("/signin")
62     public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {
63
64         Authentication authentication = authenticationManager.authenticate(
65             new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));
66
67         SecurityContextHolder.getContext().setAuthentication(authentication);
68         String jwt = jwtUtils.generateJwtToken(authentication);
69
70         UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();
71         List<String> roles = userDetails.getAuthorities().stream()
72             .map(GrantedAuthority::getAuthority)
73             .collect(Collectors.toList());
74
75         return ResponseEntity.ok(new JwtResponse(jwt,
76             userDetails.getId(),
77             userDetails.getUsername(),
78             userDetails.getEmail(),
79             roles));
80     }
81 }
```

Kontroler ten odpowiada za rejestrację użytkowników oraz ich logowanie. Sprawdza by nie było powtarzających się emaili czy ten nicków. Jeżeli chodzi o logowanie, sprawdza czy dane są zgodne i generuje odpowiedź zwrotną (w przypadku prawidłowego zalogowania zwraca token JWT a w przypadku błędnych danych, status 401, unauthorized)


```
76 @PostMapping("/signup")
77 public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {
78     if (userRepository.existsByUsername(signUpRequest.getUsername())) {
79         return ResponseEntity
80             .badRequest()
81             .body(new MessageResponse("Error: Username is already taken!"));
82     }
83
84     if (userRepository.existsByEmail(signUpRequest.getEmail())) {
85         return ResponseEntity
86             .badRequest()
87             .body(new MessageResponse("Error: Email is already in use!"));
88     }
89
90     User user = new User(signUpRequest.getUsername(),
91         signUpRequest.getEmail(),
92         encoder.encode(signUpRequest.getPassword()));
93
94     Set<String> strRoles = signUpRequest.getRole();
95     Set<Role> roles = new HashSet<>();
96
97     if (strRoles == null) {
98         Role userRole = roleRepository.findByName(ERole.ROLE_USER)
99             .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
100         roles.add(userRole);
101     } else {
102         strRoles.forEach(role -> {
103             switch (role) {
104                 case "admin" -> {
105                     Role adminRole = roleRepository.findByName(ERole.ROLE_ADMIN)
106                         .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
107                     roles.add(adminRole);
108                 }
109                 case "mod" -> {
110                     Role modRole = roleRepository.findByName(ERole.ROLE_MODERATOR)
111                         .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
112                     roles.add(modRole);
113                 }
114                 default -> {
115                     Role userRole = roleRepository.findByName(ERole.ROLE_USER)
116                         .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
117                     roles.add(userRole);
118                 }
119             }
120         });
121     }
122
123     user.setRoles(roles);
124     userRepository.save(user);
125
126     return ResponseEntity.ok(new MessageResponse("User registered successfully!"));
127 }
128 }
```

Kontroler zarządzający filmami, seansami, rejestracją

```
FilmController.java
18 @CrossOrigin(origins = "*", maxAge = 3600)
19 @RestController
20 public class FilmController {
21
22     7 usages
23     private final MovieDAO movieDAO;
24     8 usages
25     private final ScreeningDAO screeningDAO;
26     4 usages
27     private final ReservationDAO reservationDAO;
28     2 usages
29     private final UserDAO userDAO;
30
31     no body
32     @Autowired
33     public FilmController(MovieDAO movieDAO, ScreeningDAO screeningDAO, ReservationDAO reservationDAO, UserDAO userDAO) {
34         this.movieDAO = movieDAO;
35         this.screeningDAO = screeningDAO;
36         this.reservationDAO = reservationDAO;
37         this.userDAO = userDAO;
38     }
39
40     no body
41     @GetMapping("/filmy")
42     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
43     public List<MovieDTO> getMovies() { return movieDAO.getAllMovies(); }
44
45     no usages no body
46     @GetMapping("/filmy/{id}")
47     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
48     public MovieDTO getMovie(@PathVariable("id") int id) { return movieDAO.getMovieById(id); }
49
50     no usages no body
51     @GetMapping("/filmy/{id}/seanse")
52     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
53     public List<ScreeningDTO> getSeancebyFilm(@PathVariable("id") int id) {
54         return screeningDAO.getScreeningByFilmId(id);
55     }
56
57     no usages no body
58     @PostMapping("/filmy/add")
59     @PreAuthorize("hasRole('MODERATOR') or hasRole('ADMIN')")
60     public void addMovie(@RequestBody MovieDTO movieDTO) { movieDAO.addMovie(movieDTO); }
61
62     no usages no body
63     @GetMapping("/seanse/{id}")
64     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
65     public Screening getScreening(@PathVariable("id") int id) { return screeningDAO.getScreeningById(id); }
66
67     no usages no body
68     @GetMapping("/seanse/{id}/rezerwacje")
69     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
70     public List<ReservationDTO> getReservationsByScreening(@PathVariable("id") int id) {
71         return reservationDAO.getReservationsByScreening(id);
72     }
73 }
```

FilmController.java

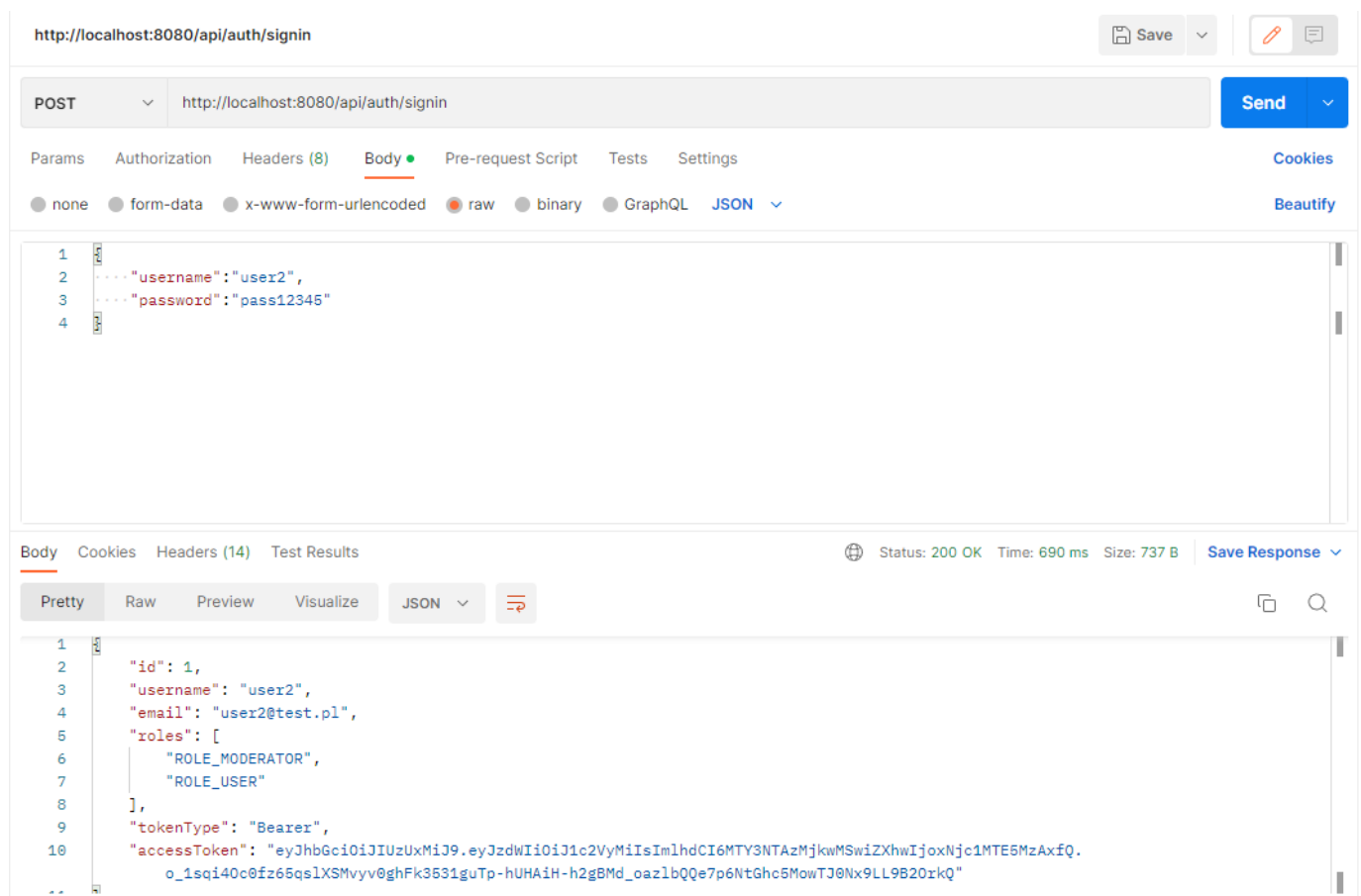
```
71     @PostMapping("/{rezerwacja/add}")
72     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
73     @RequestBody
74     public void addReservation(NewReservation newReservation) {
75         Reservation reservation = new Reservation();
76         reservation.setName(newReservation.getName());
77         reservation.setLastName(newReservation.getLastName());
78         reservation.setSeatNumber(newReservation.getSeatNumber());
79         reservation.setIdScreening(screeningDAO.getScreeningById(newReservation.getIdScreening()));
80         reservation.setUserId(userDAO.findById(newReservation.getUserId()));
81         reservationDAO.save(reservation);
82     }
83
84     no usages  no.body
85     @GetMapping("/{rezerwacje/user/{id}")
86     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
87     public List<ReservationDTO> getReservationsByUser(@PathVariable("id") int id) {
88         return reservationDAO.getReservationsByUser(id);
89     }
90
91     no usages  no.body
92     @GetMapping("/{filmy/search/{word}")
93     @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
94     public List<MovieDTO> getMoviesByWord(@PathVariable("word") String word) {
95         return movieDAO.getMoviesByWord(word);
96     }
97
98     no usages  new *
99     @PostMapping("/{seanse/add}")
100     @PreAuthorize("hasRole('MODERATOR') or hasRole('ADMIN')")
101     @RequestBody
102     public void addScreening(NewScreening newScreening) {
103         Screening screening = new Screening();
104         screening.setDate(newScreening.getDate());
105         screening.setPrice(newScreening.getPrice());
106         screening.setPrice(newScreening.getPrice());
107         screening.setIdMovie(movieDAO.getMovieByIdMovie(newScreening.getIdMovie()));
108         screeningDAO.save(screening);
109     }
110
111     no usages  new *
112     @GetMapping("/{seanse}")
113     @PreAuthorize("hasRole('MODERATOR') or hasRole('ADMIN')")
114     public List<Screening> getScreenings() {
115         return screeningDAO.getAllScreenings();
116     }
117
118     no usages  new *
119     @PutMapping("/{seanse/modify/{id}")
120     @PreAuthorize("hasRole('MODERATOR') or hasRole('ADMIN')")
121     @RequestBody
122     public void updateScreening(@PathVariable("id") int id, NewScreening newScreening) {
123         Screening screening = screeningDAO.getScreeningById(id);
124         screening.setDate(newScreening.getDate());
125         screening.setPrice(newScreening.getPrice());
126         screening.setPrice(newScreening.getPrice());
127         screening.setIdMovie(movieDAO.getMovieByIdMovie(newScreening.getIdMovie()));
128         screening.setSeats(newScreening.getSeats());
129         screeningDAO.save(screening);
130     }
```


Jak można zauważyć, kontroler posiada masę metod zarządzającymi danymi, od ich zwracania, po modyfikacje.

Wszystkie metody aby zostały wykonane potrzebują autoryzacji według ról użytkownika. Oznacza to, że zwykły użytkownik nie będzie mógł dodawać filmów czy też seansów ani ich modyfikować ale może je przeglądać.

Poniżej zamieszczam testowanie API w programie Postman:

Logowanie:



Wylistowanie seansów:

http://localhost:8080/seanse

Save

GET

http://localhost:8080/seanse

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (14)

Test Results

Status: 200 OK

Time: 37 ms

Size: 4.91 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "id": 1,
3    "idMovie": {
4      "id": 1,
5      "name": "Fight
6        Club
7
8      "description": "bicie
9        sie
10
11      "rating": 9.8,
12      "pgi": 18,
13      "image": "https://akns-images.eonline.com/eol_images/Entire_Site/2019914/rs_685x1024-191014113838-634-fight-club-poster.cl.101419.jpg?
14        fit=around%7C685:1024&output-quality=98&crop=685:1024;center,top
15    },
16    "price": 12.0,
17    "date": "2023-01-07T13:33:53.000+00:00",
18    "seats": 44
19  },
20  {
21    "id": 2,
22    "idMovie": {
23      "id": 2,
24      "name": "American
25        Psycho
26
27      "description": "
```

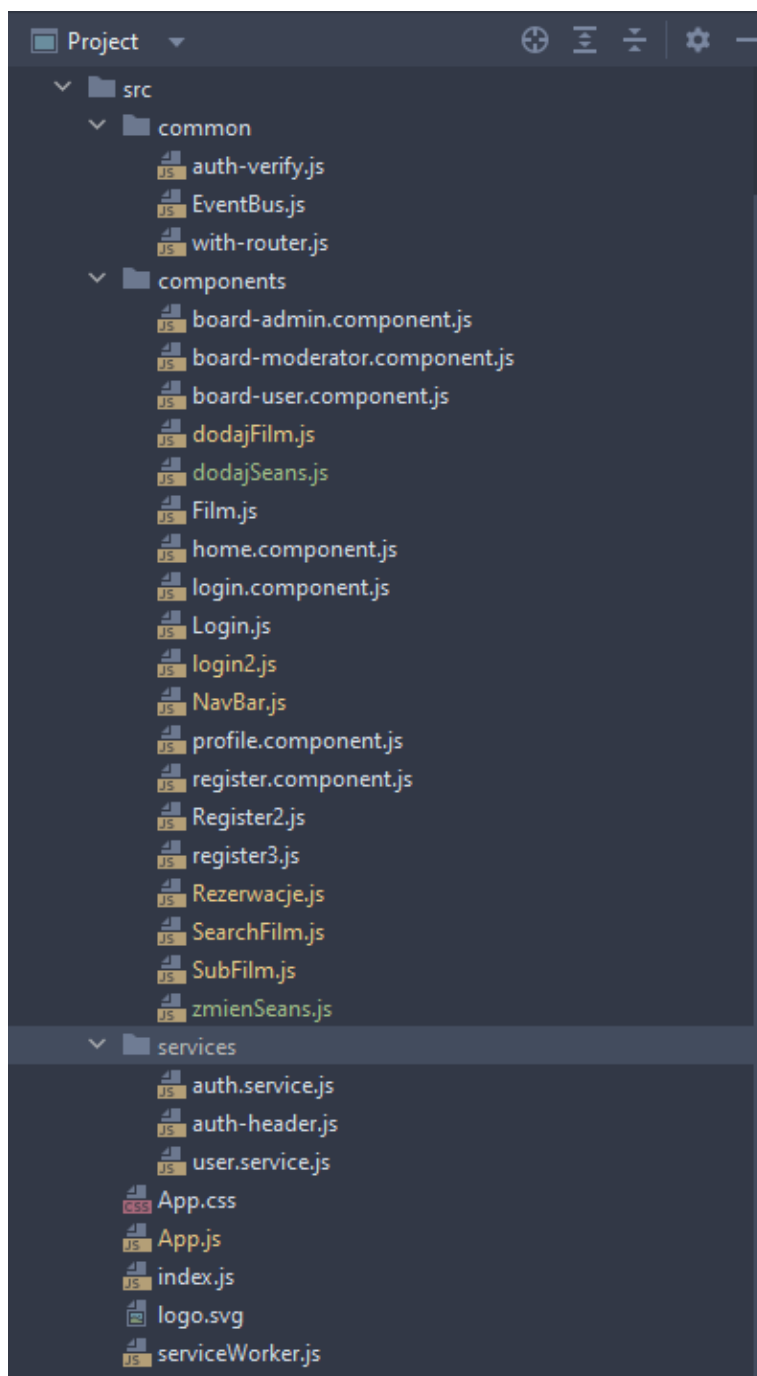
Frontend

W tej części skupię się wyłącznie na opisanu wyglądu aplikacji. Kod źródłowy jest udostępniony w postaci linku do gita lub dysku.

Dla praktycznie każdego pola input istnieje walidacja.

W frontendzie korzystałem z popularnej biblioteki komponentów ChakraUI.

Hierarchia aplikacji:



Lista Routów (możliwości)

```
82 <div className="container mt-3">
83   <Routes>
84     <Route path="/" element={<Home />} />
85     <Route path="/home" element={<Home />} />
86     <Route path="/login2" element={<Login2 />} />
87     <Route path="/register3" element={<Register3 />} />
88     <Route path="/profile" element={<Profile />} />
89     <Route path="/filmy" element={<Filmy url={'http://localhost:8080/filmy'} />} />
90     <Route path="/filmy/add" element={<DodajFilm />} />
91     <Route path="/filmy/search" element={<SearchFilm />} />
92     <Route path="/seanse/add" element={<DodajSeans />} />
93     <Route path="/seanse/modify" element={<ZmienSeans />} />
94     <Route path="/rezerwacje" element={<Rezerwacje />} />
95     <Route path="/user" element={<BoardUser />} />
96     <Route path="/mod" element={<BoardModerator />} />
97     <Route path="/admin" element={<BoardAdmin />} />
98   </Routes>
99 </div>
100
101   {<AuthVerify logOut={this.logOut}/>}
102 </div>
103 );
```

Wygląd logowania:

Logo

Sign InSign Up

Sign in to your account

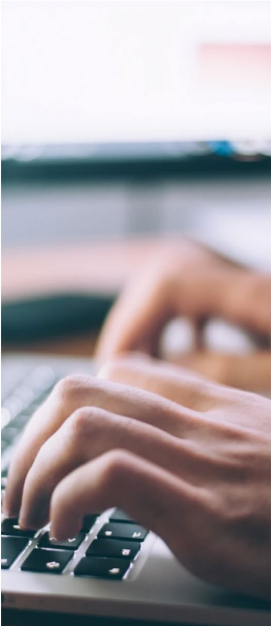
Username

test5

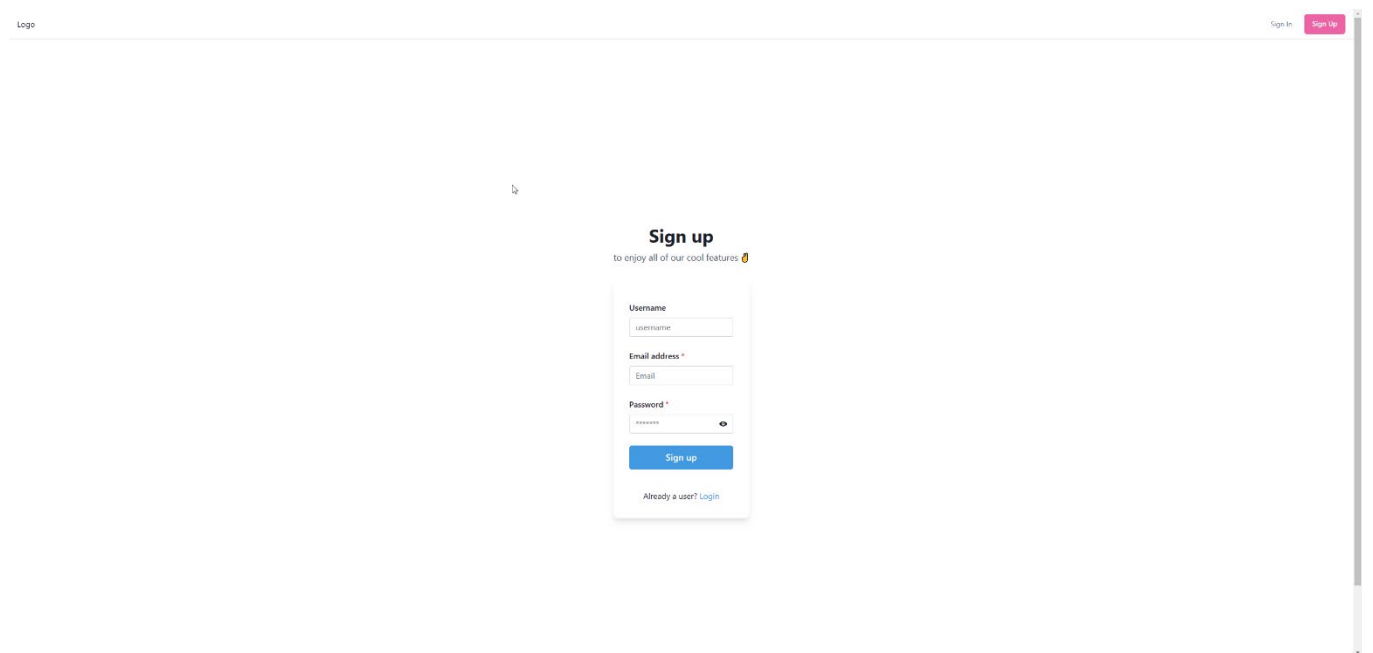
Password

This field is required!

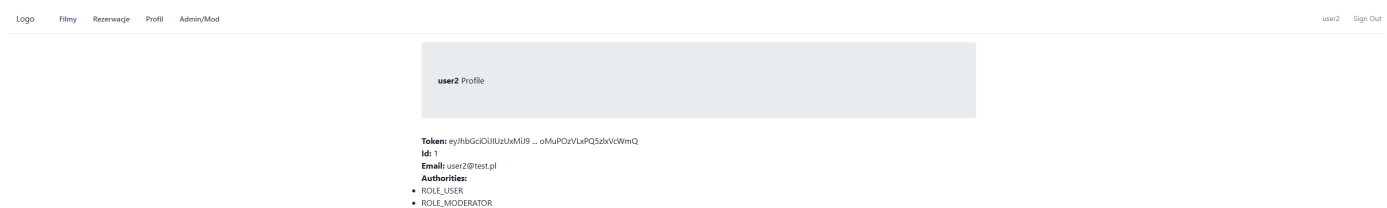
Sign in



Wygląd rejestracji:

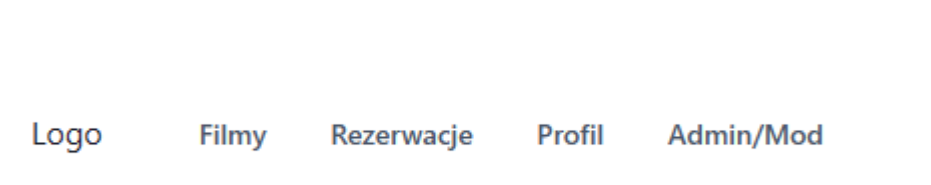


Wygląd po zalogowaniu:



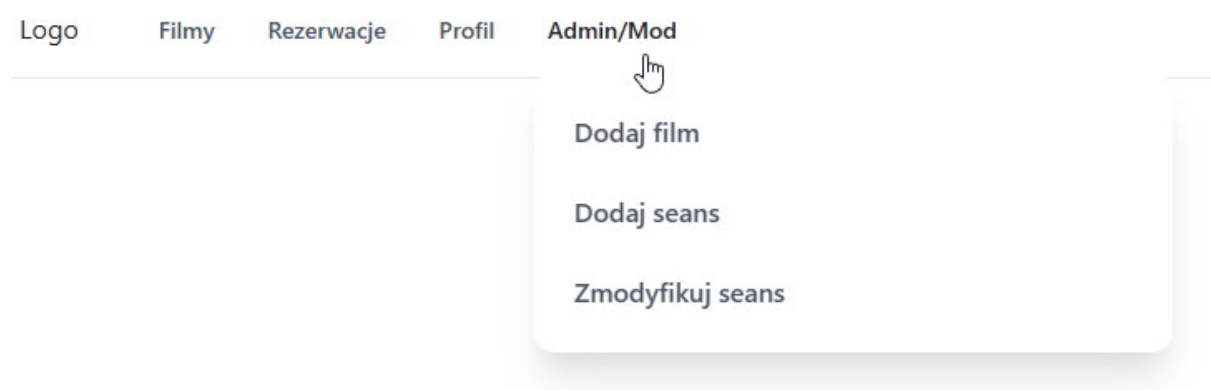
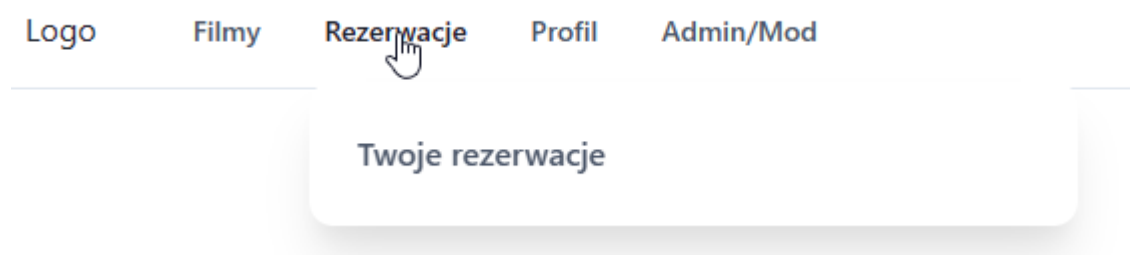
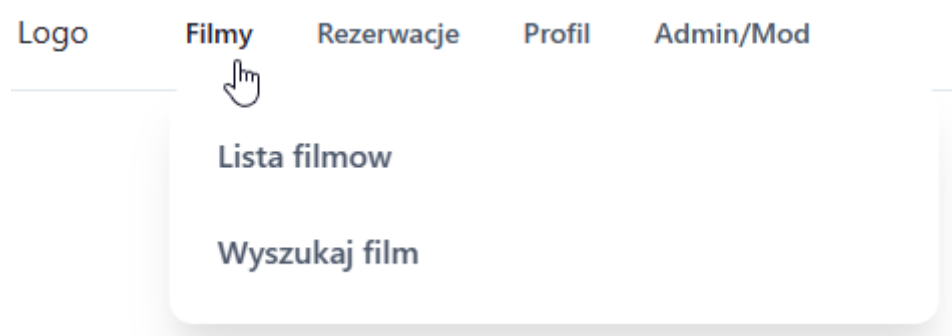
Po zalogowaniu przekierowywane nas na stronę naszego profilu, gdzie widnieją informacje o naszych rolach, emailu a nawet tokenie JWT, którego aktualnie używamy

NavBar:

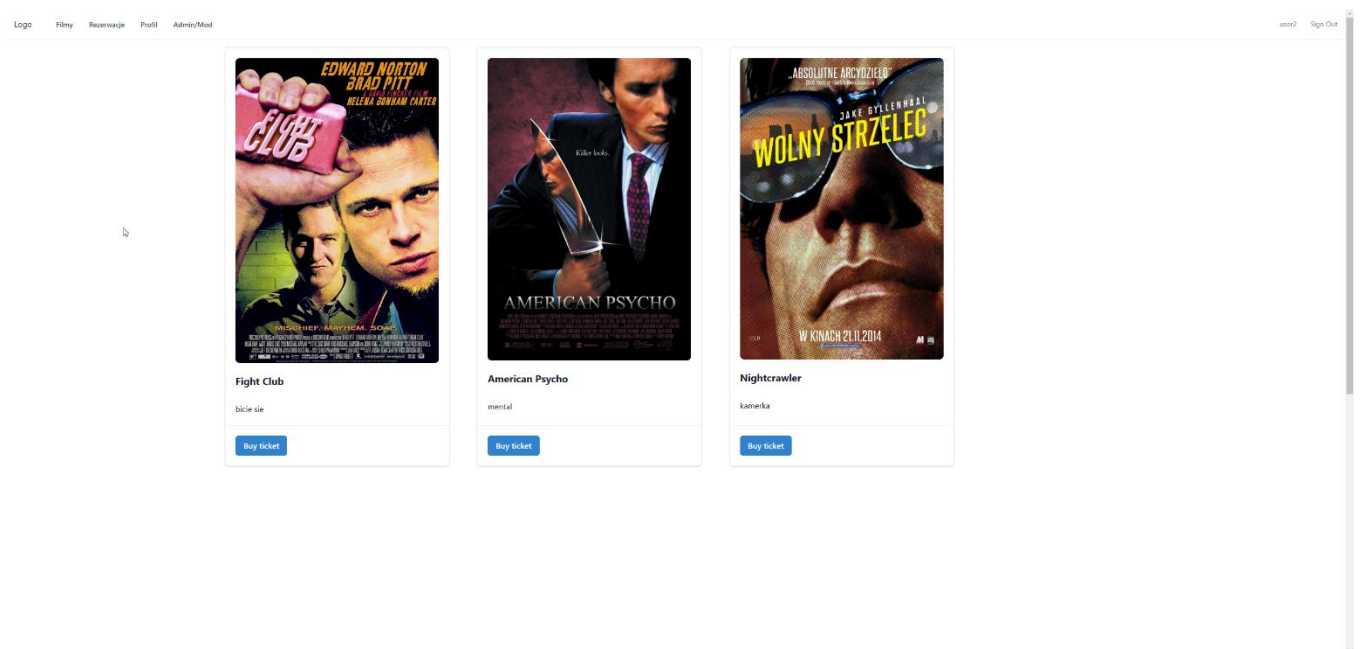


Pozycja Admin/Mod jest widoczna tylko dla zalogowanych adminów/modów

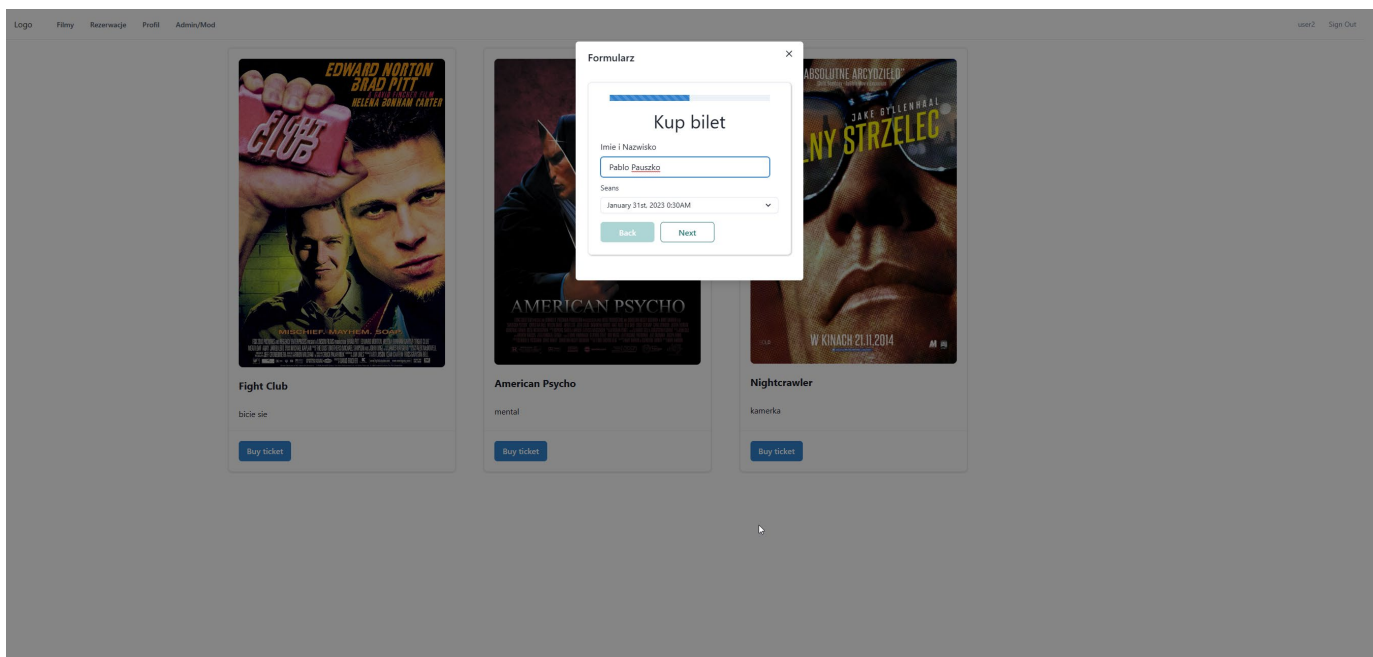
Rozwijalna lista NavBaru:



Lista filmów:



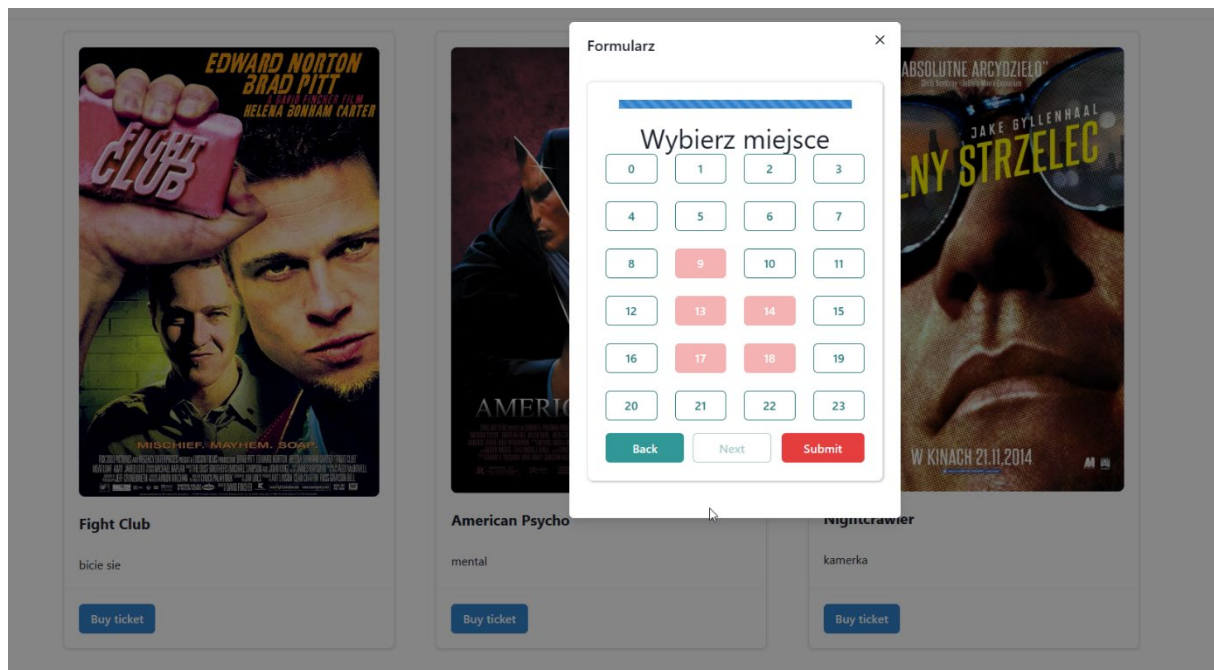
Po kliknięciu przycisku Buy Ticket, otwiera się okno umożliwiające kupno biletu



Musimy wybrać seans (date), na który chcemy iść dotycząca klikniętego filmu oraz wpisać Imię i Nazwisko.

Przestarzałe seanse, czyli takie, których data minęła, nie są wyświetlane do rezerwacji.

Następnie po kliknięciu Next, pojawia się lista miejsc do wyboru dla danego seansu. Miejsca na czerwono są zajęte i nie można ich kliknąć.



Istnieje możliwość zakupu wielu miejsc np.:



Po kliknięciu submit, przekierowujemy nas na stronę z rezerwacjami dla naszego konta:

LogoFilmyRezerwacjeProfilAdmin/Mod

user2Sign Out

Twoje rezerwacje



Nightcrawler
Data: 30.01.2023 15:33
Wybrane miejsca: 26, 25, 35, 34,
Łączna cena: 132 zł

Zapłać



Fight Club
Data: 31.01.2023 00:30
Wybrane miejsca: 18, 17, 13, 14,
21, 20,
Łączna cena: 180 zł

Zapłać



Donnie Darko
Data: 07.02.2023 23:30
Wybrane miejsca: 41, 42,
Łączna cena: 50 zł

Zapłać



Nightcrawler
Data: 15.02.2023 01:16
Wybrane miejsca: 34, 33,
Łączna cena: 40 zł

Zapłać

W moich rezerwacjach, wyświetlane są rezerwacje pogrupowane według seansu. Wyświetlona jest data seansu oraz wszystkie wykupione przez Ciebie miejsca. Widnieje także łączna cena (cena 1 biletu mnożona razy ilość miejsc).

Dodanie filmu:



Dodaj Film

do bazy danych

Nazwa filmu

Opis filmu *

Rating *

Pegi

Obraz

Dodaj film

Dodanie seansu:

Dodaj Seans

do bazy danych

Wybierz film *

American Psycho

Wybierz date *

14.01.2023 00:20



Cena *

20.00

Ilość miejsc *

40

Dodaj seans

Modyfikacja seansu:

Aktualizuj seans

do bazy danych

Wybierz seans *

Seans nr: 25 Film: Nightcrawler

Wybierz date *

15.02.2023 01:16



Cena *

20

Ilość miejsc *

45

Aktualizuj seans