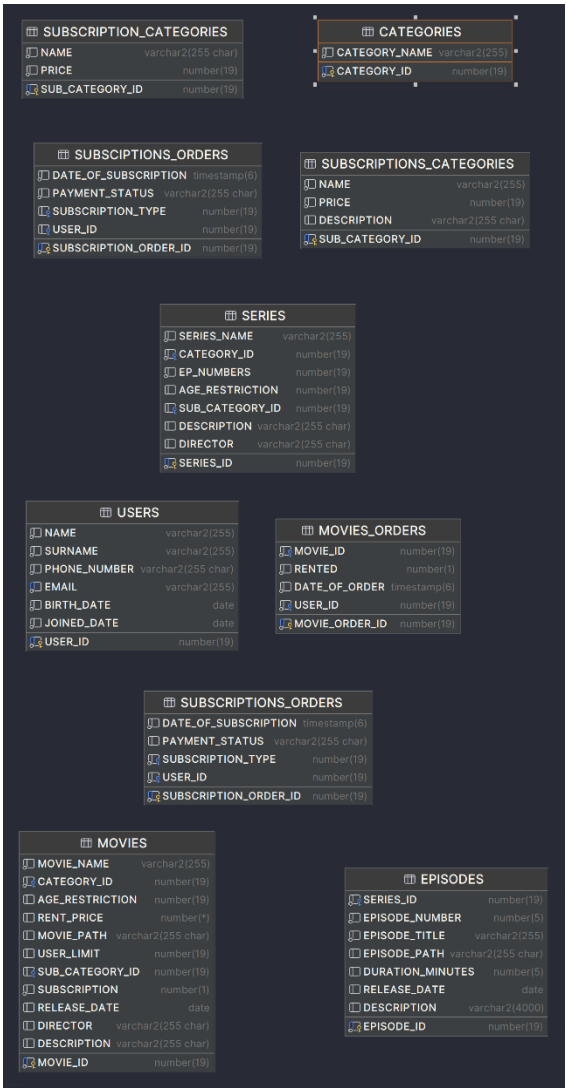
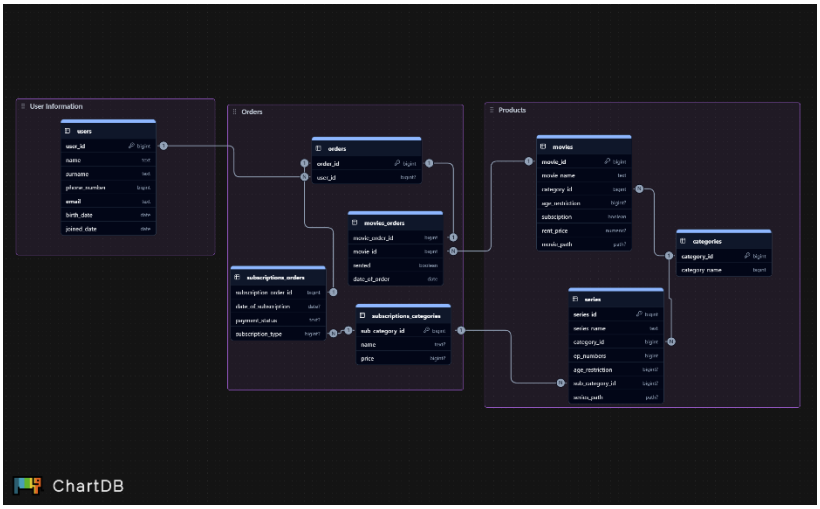
Bazy Danych Miniprojekt

Emilia Tokarz, Jakub Kowalewski

1. Schemat bazy danych:



Ten schemat jest pobrany z IntelliJ, nasz początkowy schemat wyglądał tak:



Ale ostatecznie musieliśmy dodać niektóre pola i zrezygnować z paru rzeczy, np. całej tabeli orders.

1. Stworzenie bazy danych Oracle:

```
CREATE TABLE users (
 user_id NUMBER(19) PRIMARY KEY,
 name VARCHAR2(255) NOT NULL,
 surname VARCHAR2(255) NOT NULL,
 phone_number NUMBER(19) NOT NULL,
 email VARCHAR2(255) NOT NULL UNIQUE,
 birth_date DATE NOT NULL,
 joined_date DATE NOT NULL
);

CREATE TABLE subscriptions_categories (
 sub_category_id NUMBER(19) PRIMARY KEY,
 name VARCHAR2(255),
 price NUMBER(19)
);
ALTER TABLE subscriptions_categories MODIFY (name VARCHAR2(255) NOT NULL);
ALTER TABLE subscriptions_categories MODIFY (price NUMBER(19) NOT NULL);

CREATE TABLE categories (
 category_id NUMBER(19) PRIMARY KEY,
 category_name VARCHAR2(255) NOT NULL
);

CREATE TABLE series (
 series_id NUMBER(19) PRIMARY KEY,
 series_name VARCHAR2(255) NOT NULL,
 category_id NUMBER(19) NOT NULL,
 ep_numbers NUMBER(19) NOT NULL,
 age_restriction NUMBER(19),
 sub_category_id NUMBER(19),
 series_path VARCHAR2(4000),
 CONSTRAINT fk_series_category FOREIGN KEY (category_id) REFERENCES categories (category_id),
 CONSTRAINT fk_series_subcat FOREIGN KEY (sub_category_id) REFERENCES subscriptions_categories (sub_category_id)
);

CREATE TABLE movies (
 movie_id NUMBER(19) PRIMARY KEY,
 movie_name VARCHAR2(255) NOT NULL,
 category_id NUMBER(19) NOT NULL,
 age_restriction NUMBER(19),
 subsciption NUMBER(1) NOT NULL, -- 1/0 czy dostepne w subskrypcji
 rent_price NUMBER,
 movie_path VARCHAR2(4000),
 user_limit NUMBER(19),
 CONSTRAINT fk_movies_category FOREIGN KEY (category_id) REFERENCES categories (category_id)
);
ALTER TABLE movies ADD (sub_category_id NUMBER(19));
ALTER TABLE movies
 ADD CONSTRAINT fk_movies_sub_category
 FOREIGN KEY (sub_category_id)
 REFERENCES subscriptions_categories(sub_category_id);



CREATE TABLE movies_orders (
 movie_order_id NUMBER(19) PRIMARY KEY,
 movie_id NUMBER(19) NOT NULL,
 rented NUMBER(1) NOT NULL,
 date_of_order DATE NOT NULL,
 CONSTRAINT fk_movieorder_movie FOREIGN KEY (movie_id) REFERENCES movies (movie_id)
);
```

```
CREATE TABLE subscriptions_orders (
 subscription_order_id NUMBER(19) PRIMARY KEY,
 date_of_subscription DATE,
 payment_status VARCHAR2(100),
 subscription_type NUMBER(19),
 CONSTRAINT fk_subsorder_subcat FOREIGN KEY (subscription_type) REFERENCES subscriptions_categories (sub_category_id),
 );

ALTER TABLE users MODIFY (phone_number VARCHAR2(12));

ALTER TABLE subscriptions_orders ADD (user_id NUMBER(19));
ALTER TABLE subscriptions_orders
 ADD CONSTRAINT fk_subsorder_user
 FOREIGN KEY (user_id)
 REFERENCES users(user_id);


ALTER TABLE movies_orders ADD (user_id NUMBER(19));
ALTER TABLE movies_orders
 ADD CONSTRAINT fk_movorder_user
 FOREIGN KEY (user_id)
 REFERENCES users(user_id);


ALTER TABLE movies_orders MODIFY (user_id NUMBER(19) NOT NULL);
ALTER TABLE subscriptions_orders MODIFY (user_id NUMBER(19) NOT NULL);

ALTER TABLE subscriptions_orders MODIFY (date_of_subscription DATE NOT NULL);
ALTER TABLE subscriptions_orders MODIFY (subscription_type NUMBER(19) NOT NULL);


ALTER TABLE MOVIES add (release_date Date);
ALTER TABLE MOVIES add (director VARCHAR2(100));
ALTER TABLE MOVIES add (description VARCHAR2(4000));

ALTER TABLE SUBSCRIPTIONS_CATEGORIES add (description VARCHAR2(4000))


ALTER TABLE SERIES add (description VARCHAR2(4000))

ALTER TABLE SERIES drop (SERIES_PATH);

CREATE TABLE episodes (
 episode_id NUMBER(19) PRIMARY KEY,
 series_id NUMBER(19) NOT NULL,
 episode_number NUMBER(5) NOT NULL,
 episode_title VARCHAR2(255) NOT NULL,
 episode_path VARCHAR2(4000),
 duration_minutes NUMBER(5),
 release_date DATE,
 description VARCHAR2(4000),

 CONSTRAINT fk_episode_series FOREIGN KEY (series_id) REFERENCES series (series_id)
);

ALTER TABLE series add (director VARCHAR2(100));

CREATE SEQUENCE movie_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE category_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE subscription_category_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE user_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE show_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE mov_ord_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE sub_ord_seq START WITH 1 INCREMENT BY 1;

CREATE sequence episode_seq START WITH 1 INCREMENT BY 1;
```

2. Setup projektu w Javie

Stworzyliśmy Maven’owy projekt I skonfigurowaliśmy go tak, aby działał z hibernate.

Fragment pliku pom.xml:

```xml
<dependencies>
    <!--hibernate-->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.2.7.Final</version>
    </dependency>
    <!-- oracle jdbc -->
    <dependency>
        <groupId>com.oracle.database.jdbc</groupId>
        <artifactId>ojdbc8</artifactId>
        <version>21.9.0.0</version>
    </dependency>
    <!-- jpa api-->
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>
```

Pl

Plik hibernate.cfg.xml

```xml
<!DOCTYPE hibernate-configuration SYSTEM
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>

        <property name="hibernate.connection.url">jdbc:oracle:thin:@192.168.0.115:1521/xepdb1</property>
        <!-- logowanie -->
        <property name="hibernate.connection.username">pom</property>
        <property name="hibernate.connection.password">pom</property>

        <property name="hibernate.dialect">org.hibernate.dialect.Oracle12cDialect</property>

        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>

        <property name="hibernate.hbm2ddl.auto">update</property>
    </session-factory>
</hibernate-configuration>
```

Stworzyliśmy klasę HibernateUtil która wczytuje konfigurację, rejestruje klasy encji i buduje fabrykę sesji.

```java
public class HibernateUtil {
    private static final SessionFactory sessionFactory;  2 usages

    static {
        try{
            sessionFactory = new Configuration()
                    .configure( resource: "hibernate.cfg.xml")
                    .addAnnotatedClass(User.class)
                    .addAnnotatedClass(objs.Category.class)
                    .addAnnotatedClass(objs.SubscriptionCategory.class)
                    .addAnnotatedClass(objs.Movie.class)
                    .addAnnotatedClass(objs.Show.class)
                    .addAnnotatedClass(objs.orders.MovieOrder.class)
                    .addAnnotatedClass(objs.orders.SubscriptionOrder.class)
                    .addAnnotatedClass(objs.ShowEpisode.class)
                    .buildSessionFactory();
        }
        catch(Throwable e){
            System.err.println("Initial SessionFactory creation failed." + e);
            throw new ExceptionInInitializerError(e);
        }
    }
    public static SessionFactory getSessionFactory() {  55 usages
        return sessionFactory;
    }
}
```

3. Stworzenie klas encji dla każdej tabeli

User:

```java
@Entity
@Table(name="users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_seq")
    @SequenceGenerator(name="user_seq", sequenceName = "user_seq", allocationSize = 1)
    @Column(name="user_id")
    private Long id;
    @Column(name="name",nullable=false)  2 usages
    private String name;

    @Column(name="surname", nullable=false)  2 usages
    private String surname;

    @Column(name="phone_number",nullable=false)  2 usages
    private String phoneNumber;

    @Column(name="email", nullable=false)  2 usages
    private String email;

    @Column(name="birth_date",nullable=false)  2 usages
    private LocalDate birthDate;

    @Column(name="joined_date",nullable=false)  2 usages
    private LocalDate joinedDate;
```

Movie:

```java
@Entity
@Table(name="movies")
public class Movie {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "movie_seq")
    @SequenceGenerator(name="movie_seq",sequenceName = "movie_seq", allocationSize = 1)
    @Column(name="movie_id")
    private Long movieID;

    @Column(name="movie_name", nullable=false)  2 usages
    private String movieName;

    @ManyToOne  2 usages
    @JoinColumn(name="category_id", nullable = false)
    private Category category;

    @Column(name="age_restriction")  2 usages
    private int ageRestriction;

    @Column(name="subscription", nullable = false)  2 usages
    private Boolean subscriptionAvailable;

    @Column(name="rent_price")  2 usages
    private Long rentPrice;
```

```java
    @Column(name="movie_path")  2 usages
    private String moviePath;

    @Column(name="user_limit")  2 usages
    private Long userLimit;

    @ManyToOne  2 usages
    @JoinColumn(name="sub_category_id")
    private SubscriptionCategory subscriptionCategory;

    @Column(name="release_date")  2 usages
    private LocalDate releaseDate;

    @Column(name="director")  2 usages
    private String director;

    @Column(name="description")  2 usages
    private String description;
```

Show (w db widnieje series – nazwa którą najpierw używaliśmy, ale w projekcie javowym zostało to zmienione w celu lepszej czytelności):

```java
@Entity
@Table(name="series")
public class Show {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "show_seq")
    @SequenceGenerator(name="show_seq",sequenceName = "show_seq", allocationSize = 1)
    @Column(name="series_id")
    private Long showId;

    @Column(name="series_name", nullable = false)  2 usages
    private String showName;

    @ManyToOne  2 usages
    @JoinColumn(name="category_id", nullable = false)
    private Category category;

    @Column(name="ep_numbers", nullable = false)  2 usages
    private Long epNumbers;

    @Column(name="age_restriction")  2 usages
    private Long ageRestriction;

    @ManyToOne  2 usages
    @JoinColumn(name="sub_category_id")
    private SubscriptionCategory subscriptionCategory;

    @Column(name="description")  2 usages
    private String description;

    @Column(name="director")  2 usages
    private String director;
```

ShowEpisode:

```java
@Entity  49 usages
@Table(name = "episodes")
public class ShowEpisode {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "episode_seq")
    @SequenceGenerator(name = "episode_seq", sequenceName = "episode_seq", allocationSize = 1)
    @Column(name = "episode_id")
    private Long episodeId;

    @ManyToOne(fetch = FetchType.LAZY)  2 usages
    @JoinColumn(name = "series_id", nullable = false)
    private Show show;

    @Column(name = "episode_number", nullable = false)  2 usages
    private int episodeNumber;

    @Column(name = "episode_title")  2 usages
    private String episodeTitle;

    @Column(name = "episode_path")  2 usages
    private String episodePath;

    @Column(name = "duration_minutes")  2 usages
    private Integer durationMinutes;

    @Column(name = "release_date")  2 usages
    private LocalDate releaseDate;

    @Column(name = "description", length = 4000)  2 usages
    private String description;
```

SubscriptionCategory:

```java
@Entity
@Table(name="subscriptions_categories")
public class SubscriptionCategory {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "subscription_category_seq")
    @SequenceGenerator(name="subscription_category_seq",sequenceName = "subscription_category_seq", allocationSize = 1)
    @Column(name="sub_category_id")
    private Long subCategoryId;

    @Column(name="name",nullable=false)  2 usages
    private String name;

    @Column(name="price",nullable=false)  2 usages
    private Long price;

    @Column(name="description")  2 usages
    private String description;
```

Category:

```java
@Entity
@Table(name="categories")
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,generator = "category_seq")
    @SequenceGenerator(name="category_seq", sequenceName = "category_seq", allocationSize = 1)
    @Column(name="category_id")
    private Long categoryId;

    @Column(name="category_name", nullable = false)  2 usages
    private String categoryName;
```

MovieOrder:

```java
@Entity  22 usages
@Table(name="movies_orders")
public class MovieOrder {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "mov_ord_seq")
    @SequenceGenerator(name="mov_ord_seq", sequenceName = "mov_ord_seq", allocationSize = 1)
    @Column(name="movie_order_id")
    private Long OrderId;

    @ManyToOne  2 usages
    @JoinColumn(name="movie_id", nullable = false)
    private Movie movie;

    @ManyToOne  2 usages
    @JoinColumn(name="user_id",nullable = false)
    private User user;

    @Column(name="rented",nullable = false)  3 usages
    private boolean rented;

    @Column(name="date_of_order",nullable = false)  3 usages
    private LocalDateTime dateOfOrder;
```

SubscriptionOrder:

```java
@Entity  46 usages
@Table(name="subscriptions_orders")
public class SubscriptionOrder {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sub_ord_seq")
    @SequenceGenerator(name = "sub_ord_seq",sequenceName = "sub_ord_seq", allocationSize = 1)
    @Column(name="subscription_order_id")
    private Long orderId;

    @Column(name="date_of_subscription",nullable=false)  2 usages
    private LocalDateTime dateOfSubscription;

    @Column(name="payment_status",nullable = false)  2 usages
    private String paymentStatus;

    @ManyToOne  2 usages
    @JoinColumn(name="subscription_type")
    private SubscriptionCategory subscriptionCategory;

    @ManyToOne  2 usages
    @JoinColumn(name="user_id")
    private User user;
```

Dodatkowo każda klasa ma zaimplementowane settery i gettery dla każdego atrybutu, np. w Category:

```java
    //SETTERS
    public void setCategoryId(Long categoryId) {  2 usages
        this.categoryId = categoryId;
    }
    public void setCategoryName(String categoryName) {
        this.categoryName = categoryName;
    }

    //GETTERS
    public Long getCategoryId() {  1 usage
        return categoryId;
    }
    public String getCategoryName() {
        return categoryName;
    }
}
```

4. Operacje CRUD oraz złożone operacje

Najpierw stworzyliśmy klasy DAO (Data Access Object) które odpowiadają za bezpośrednią komunikację z bazą danych.

```
public interface BasicDAO<T> {  8 usages  8 implementations
    void save(T entity);  8 implementations
    T getById(Long id);  8 implementations
    void update(T entity);  8 implementations
    void delete(T entity);  8 implementations
    List<T> getAll();  8 implementations
}
```

Każda klasa zawiera metody do podstawowych operacji CRUD. Dla łatwiejszego korzystania z aplikacji i bardziej skomplikowanych operacji stworzyliśmy również dodatkowe klasy *nazwa*Service.

User:

```
public class UserDAO implements BasicDAO<User> {  21 usages
    @Override
    public void save(User user) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.save(user);
        session.getTransaction().commit();
        session.close();
    }
    @Override
    public User getById(Long id){
        Session session = HibernateUtil.getSessionFactory().openSession();
        User user =session.get(User.class, id);
        session.close();
        return user;
    }
    @Override
    public void update(User user) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(user);
        session.getTransaction().commit();
        session.close();
```

```java
    @Override
    public void delete(User user) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.delete(user);
        session.getTransaction().commit();
        session.close();
    }
    @Override
    public List<User> getAll() {
        Session session = HibernateUtil.getSessionFactory().openSession();
        List<User> users = session.createQuery( s: "FROM User", User.class).list();
        session.close();
        return users;
    }
    public void deleteAll() {  no usages
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.createQuery( s: "DELETE FROM User").executeUpdate();
        session.getTransaction().commit();
        session.close();
    }
```

```java
public User getUserByEmail(String email) {  4 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    Query<User> query = session.createQuery( s: "from User where email = :email", User.class);
    query.setParameter( s: "email", email);
    User user=null;
    try{
        user=query.uniqueResult();
    } catch (Exception e){
        e.printStackTrace();
    } finally {
        session.close();
    }
    return user;
}
```

W UserService zaimplementowaliśmy metody do updatowania konkretnych informacji, używając UserDAO. Dodatkowo przeprowadzamy walidację danych. Walidacja email i numeru telefonu odbywa się przy użyciu regexów, a walidacja daty urodzenia na potrzeby testów aplikacji wymaga, aby użytkownik miał co najmniej 11 lat.

```java
public class UserService {  3 usages
    public UserDAO dao = new UserDAO();
    public void addUser(String name, String surname, String phoneNumber, String email, LocalDate birthDate) {
        if(dao.getUserByEmail(email) != null) {
            throw new IllegalArgumentException("User already exists");
        }
        validateData(name, surname, phoneNumber, email, birthDate);
        User user=new User();
        user.setName(name);
        user.setSurname(surname);
        user.setPhoneNumber(phoneNumber);
        user.setEmail(email);
        user.setBirthDate(birthDate);
        user.setJoinedDate(LocalDate.now());
        dao.save(user);
    }
    public User getUserById(Long id){  1 usage
        return dao.getById(id);
    }
```

```java
    public void updateUserEmail(Long id, String email) {  2 usages
        validateEmail(email);
        User user=dao.getById(id);
        if (user!=null){
            user.setEmail(email);
            dao.update(user);
        }
    }
    public void updateUserPhoneNumber(Long id, String phoneNumber) {  2 usages
        validatePhoneNumber(phoneNumber);
        User user=dao.getById(id);
        if (user!=null){
            user.setPhoneNumber(phoneNumber);
            dao.update(user);
        }
    }
    public void deleteUser(Long id){  2 usages
        User user=dao.getById(id);
        if (user!=null){
            dao.delete(user);
        }
    }
    public void deleteUser(String email){  2 usages
        User user=dao.getUserByEmail(email);
        if (user!=null){
            dao.delete(user);
        } else{
            System.out.println("User not found");
        }
    }
```

```java
public void updateUserNameAndSurname(Long id, String name, String surname) {  1 usage
    User user=dao.getById(id);
    if (user!=null){
        user.setName(name);
        user.setSurname(surname);
        dao.update(user);
    }
}

//walidacja danych
private void validateData(String name, String surname, String phoneNumber, String email, LocalDate birthDate) {
    if(name==null || name.isBlank()) throw new IllegalArgumentException("Name cannot be blank");
    if(surname == null || surname.isBlank()) throw new IllegalArgumentException("Surname cannot be blank");
    validatePhoneNumber(phoneNumber);
    validateEmail(email);
    Long minage=Long.parseLong( s: "11");
    LocalDate mini=LocalDate.now().minusYears(minage);
    if (birthDate == null) {
        throw new IllegalArgumentException("Birth date cannot be blank.");
    }
    if(birthDate.isAfter(mini)) throw new IllegalArgumentException("User must be at least 11 years old.");
}
```

```java
private void validatePhoneNumber(String phoneNumber) {  2 usages
    if(phoneNumber==null  || phoneNumber.isBlank()) throw new IllegalArgumentException("PhoneNumber cannot be blank");
    String e164Pattern = "^\\+[1-9]\\d{9,14}$";
    if(!phoneNumber.matches(e164Pattern)){
        throw new IllegalArgumentException("Invalid phone number. Expected format like +48123456789");
    }
}
private void validateEmail(String email) {  2 usages
    if(email==null  ||  email.isBlank()) throw new IllegalArgumentException("Email cannot be blank");
    String emailPattern = "^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,}$";
    if(!email.matches(emailPattern)){
        throw new IllegalArgumentException("Invalid email format.");
    }
}
```

Testy:

Testy robiliśmy z użyciem mocków, aby nie zapełniały nam bazy danych.

```java
public class UserServiceTest {

    private UserDAO dao;  20 usages
    private UserService service;  16 usages

    @BeforeEach
    void setUp() {
        dao = mock(UserDAO.class);
        service = new UserService();
        service.dao = dao;
    }
```

Aby udowodnić poprawność operacji, oto część testów:

```
@Test
void addUser_invalidEmail_throwsException() {
    Exception ex = assertThrows(IllegalArgumentException.class, () -> {
        service.addUser( name: "John",  surname: "Doe",  phoneNumber: "+48123456789",  email: "invalid-email",
                LocalDate.of( year: 1990, month: 1, dayOfMonth: 1));
    });
    assertTrue(ex.getMessage().contains("Invalid email format"));
}
```

```
@Test
void addUser_tooYoung_throwsException() {
    LocalDate tooYoung = LocalDate.now().minusYears( yearsToSubtract: 10);
    Exception ex = assertThrows(IllegalArgumentException.class, () -> {
        service.addUser( name: "John",  surname: "Doe",  phoneNumber: "+48123456789",  email: "john.doe@example.com", tooYoung);
    });
    assertTrue(ex.getMessage().contains("at least 11 years old"));
}
```

```
@Test
void getUserById_existingUser_returnsUser() {
    User user = new User();
    user.setName("Jane");
    when(dao.getById(1L)).thenReturn(user);
    User result = service.getUserById(1L);
    assertEquals( expected: "Jane", result.getName());
}
```

```
@Test
void testDeleteUser_whenUserDoesNotExist() {
    when(dao.getUserByEmail("missing@example.com")).thenReturn( t: null);
    assertDoesNotThrow(() -> service.deleteUser( email: "missing@example.com"));
    verify(dao, never()).delete(any());
}
```

CategoryDAO (bez podstawowych operacji implementujących interfejs basicDAO które wszędzie są takie same)

```
public Category getCategoryByName(String name) {  24 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    Category category=null;
    try{
        category = (Category) session.createQuery( s: "from Category where categoryName=:name",
                Category.class).setParameter( s: "name", name).uniqueResult();
    } finally {
        session.close();
    }
    return category;
}
```

CategoryService:

```java
public class CategoryService {  3 usages
    public CategoryDAO dao = new CategoryDAO();
    public void addCategory(String name){  2 usages
        if(dao.getCategoryByName(name) != null){
            throw new IllegalArgumentException("Category already exists");
        }
        Category category = new Category();
        category.setCategoryName(name);
        dao.save(category);
    }
    public Category getCategoryById(Long id){  1 usage
        return dao.getById(id);
    }
    public Category getCategoryByName(String name){  1 usage
        return dao.getCategoryByName(name);
    }
    public void deleteCategory(Long id){  1 usage
        Category category = dao.getById(id);
        dao.delete(category);
    }
    public void deleteCategory(String name){  1 usage
        Category category=dao.getCategoryByName(name);
        dao.delete(category);
    }
```

Poprawność:

```java
    @Test
    public void testAddCategory_success() {
        String name = "Test Category";

        when(mockDao.getCategoryByName(name)).thenReturn( t: null);

        categoryService.addCategory(name);

        verify(mockDao).getCategoryByName(name);
        verify(mockDao).save(any(Category.class));
    }

    @Test
    public void testAddCategory_alreadyExists_throws() {
        String name = "Existing Category";
        Category existing = new Category();
        existing.setCategoryName(name);

        when(mockDao.getCategoryByName(name)).thenReturn(existing);

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
            categoryService.addCategory(name);
        });
```

```java
@Test
public void testGetCategoryById() {
    Long id = 1L;
    Category mockCategory = new Category();
    mockCategory.setCategoryId(id);
    mockCategory.setCategoryName("Mocked");

    when(mockDao.getById(id)).thenReturn(mockCategory);

    Category result = categoryService.getCategoryById(id);

    assertNotNull(result);
    assertEquals(id, result.getCategoryId());
    assertEquals( expected: "Mocked", result.getCategoryName());
    verify(mockDao).getById(id);
}

@Test
public void testGetCategoryByName() {
    String name = "Some Category";
    Category mockCategory = new Category();
    mockCategory.setCategoryName(name);

    when(mockDao.getCategoryByName(name)).thenReturn(mockCategory);

    Category result = categoryService.getCategoryByName(name);

    assertNotNull(result);
    assertEquals(name, result.getCategoryName());
    verify(mockDao).getCategoryByName(name);
}
```

MovieDAO:

```java
public Movie getByNameDirector(String name, String director) {  17 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    Movie movie = null;
    try {
        movie = (Movie) session.createQuery( s: "from Movie where movieName = :name AND director= :director", Movie.class)
                .setParameter( s: "name", name)
                .setParameter( s: "director", director).uniqueResult();
    } finally {
        session.close();
    }
    return movie;
}
```

MovieService:

```java
public class MovieService {  3 usages
    public MovieDAO movieDao = new MovieDAO();  12 usages
    public CategoryDAO categoryDao = new CategoryDAO();  2 usages
    public SubscriptionCategoryDAO subscriptionCategoryDao= new SubscriptionCategoryDAO();  3

    public void addMovie(String name,  4 usages
                         String categoryName,
                         int ageRestriction,
                         Boolean subscriptionAvailable,
                         String subscriptionCategory,
                         Long rentPrice,String moviePath,
                         Long userLimit,LocalDate releaseDate,
                         String director, String description){
        if (movieDao.getByNameDirector(name,director)!=null){
            throw new IllegalArgumentException("Movie already exists");
        }
        Category category = categoryDao.getCategoryByName(categoryName);
        if (category==null){
            throw new IllegalArgumentException("Category not found");
        }
        SubscriptionCategory scategory = null;
        if (subscriptionCategory!=null) {
            scategory = subscriptionCategoryDao.getByName(subscriptionCategory);
            if (scategory == null) {
                throw new IllegalArgumentException("Subscription category not found");
            }
        }
```

```java
        }
        Movie movie = new Movie();
        movie.setMovieName(name);
        movie.setCategory(category);
        movie.setAgeRestriction(ageRestriction);
        movie.setSubscriptionAvailable(subscriptionAvailable);
        movie.setSubscriptionCategory(scategory);
        movie.setRentPrice(rentPrice);
        movie.setMoviePath(moviePath);
        movie.setUserLimit(userLimit);
        movie.setReleaseDate(releaseDate);
        movie.setDirector(director);
        movie.setDescription(description);
        movieDao.save(movie);
    }
```

```java
public Movie getMovie(String name, String director){  2 usages
    return movieDao.getByNameDirector(name,director);
}
public Movie getMovie(Long id){  2 usages
    return movieDao.getById(id);
}
public void deleteMovie(Long id){  2 usages
    Movie movie = movieDao.getById(id);
    if (movie==null){
        throw new IllegalArgumentException("Movie not found");
    }
    movieDao.delete(movie);
}
public void deleteMovie(String name,String director){  2 usages
    Movie movie = movieDao.getByNameDirector(name,director);
    if (movie==null){
        throw new IllegalArgumentException("Movie not found");
    }
    movieDao.delete(movie);
}
public List<Movie> getAllMovies(){  2 usages
    return movieDao.getAll();
}
public void updateMoviePrice(String name, String director, Long price){  1 usage
    Movie movie = getMovie(name, director);
    movie.setRentPrice(price);
    movieDao.update(movie);
}
```

```java
public void updateSubscription(String name, String director,Boolean subscriptionAvailable, String subscriptionCategory)
    Movie movie=getMovie(name,director);
    SubscriptionCategory scategory = null;
    if (subscriptionCategory!=null) {
        scategory = subscriptionCategoryDao.getByName(subscriptionCategory);
        if (scategory == null) {
            throw new IllegalArgumentException("Subscription category not found");
        }
    }
    movie.setSubscriptionAvailable(subscriptionAvailable);
    movie.setSubscriptionCategory(scategory);
    movieDao.update(movie);

}
```

Poprawność:

```java
@Test
public void testAddMovie_success() {
    // Arrange
    String name = "Movie A";
    String director = "Director A";
    String categoryName = "Action";
    String subscriptionName = "basic";

    Category category = new Category();
    category.setCategoryName(categoryName);

    SubscriptionCategory subscription = new SubscriptionCategory();
    subscription.setName(subscriptionName);
    when(mockMovieDao.getByNameDirector(name, director)).thenReturn( t: null);
    when(mockCategoryDao.getCategoryByName(categoryName)).thenReturn(category);
    when(mockSubscriptionCategoryDao.getByName(subscriptionName)).thenReturn(subscription);
    // Act
    movieService.addMovie(name, categoryName, ageRestriction: 16, subscriptionAvailable: true, subscriptionName, rentPrice: 20L,
            moviePath: "/movie/path", userLimit: 10L,
           LocalDate.of( year: 2020, month: 1, dayOfMonth: 1), director, description: "description");
    // Assert
    verify(mockMovieDao).getByNameDirector(name, director);
    verify(mockCategoryDao).getCategoryByName(categoryName);
    verify(mockSubscriptionCategoryDao).getByName(subscriptionName);
    verify(mockMovieDao).save(any(Movie.class));
}
```

```java
@Test
public void testAddMovie_alreadyExists_shouldThrow() {
    String name = "Movie B";
    String director = "Director B";

    when(mockMovieDao.getByNameDirector(name, director)).thenReturn(new Movie());

    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
        movieService.addMovie(name, categoryName: "Action", ageRestriction: 18, subscriptionAvailable: false,
                subscriptionCategory: null, rentPrice: 15L, moviePath: "/path", userLimit: 5L,
               LocalDate.now(), director, description: "desc");
    });

    assertEquals( expected: "Movie already exists", ex.getMessage());
    verify(mockMovieDao).getByNameDirector(name, director);
    verify(mockMovieDao, never()).save(any());
}
```

```java
@Test
public void testAddMovie_categoryNotFound_shouldThrow() {
    String name = "Movie C";
    String director = "Director C";
    String categoryName = "Fantasy";

    when(mockMovieDao.getByNameDirector(name, director)).thenReturn( t: null);
    when(mockCategoryDao.getCategoryByName(categoryName)).thenReturn( t: null);

    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
        movieService.addMovie(name, categoryName,  ageRestriction: 12,  subscriptionAvailable: false,  subscriptionCategory: null,
                rentPrice: 12L,  moviePath: "/path",  userLimit: 3L,
                LocalDate.now(), director,  description: "desc");
    });

    assertEquals( expected: "Category not found", ex.getMessage());
    verify(mockCategoryDao).getCategoryByName(categoryName);
    verify(mockMovieDao, never()).save(any());
}
```

```java
@Test
public void testDeleteMovie_byNameDirector_success() {
    String name = "Movie E";
    String director = "Director E";
    Movie mockMovie = new Movie();

    when(mockMovieDao.getByNameDirector(name, director)).thenReturn(mockMovie);

    movieService.deleteMovie(name, director);

    verify(mockMovieDao).getByNameDirector(name, director);
    verify(mockMovieDao).delete(mockMovie);
}
```

```java
    @Test
    public void testGetMovieById_found() {
        Movie movie = new Movie();
        movie.setMovieName("Test Movie");
        when(mockMovieDao.getById(1L)).thenReturn(movie);

        Movie result = movieService.getMovie( id: 1L);
        assertNotNull(result);
        assertEquals( expected: "Test Movie", result.getMovieName());
        verify(mockMovieDao).getById(1L);
    }

    @Test
    public void testGetMovieById_notFound() {
        when(mockMovieDao.getById(999L)).thenReturn( t: null);

        Movie result = movieService.getMovie( id: 999L);
        assertNull(result);
        verify(mockMovieDao).getById(999L);
    }
```

MovieOrderDAO

```java
public long countRentalsNow(Long movieId) {  5 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    LocalDateTime cutoff = LocalDateTime.now().minusHours(48);

    Long count = session.createQuery( s: """
        select count(mo) from MovieOrder mo
        where mo.movie.movieID = :movieId
        and mo.rented = true
        and mo.dateOfOrder >= :cutoff
    """, Long.class)
            .setParameter( s: "movieId", movieId)
            .setParameter( s: "cutoff", cutoff)
            .uniqueResult();

    session.close();
    return count != null ? count : 0;
}
```

MovieOrderService:

Ta klasa zajmuje się wypożyczeniem filmu, a metoda rentMovie() to jedna z operacji transakcjonalnych w której kontrolujemy ilość użytkowników, którzy aktualnie wypożyczają film, tak aby nie przekroczyli limitu.

```
public class MovieOrderService {  4 usages

    public MovieOrderDAO movieOrderDAO= new MovieOrderDAO();  3 usages
    public MovieDAO movieDAO= new MovieDAO();  2 usages
    public UserDAO userDAO= new UserDAO();  2 usages

    @Transactional  4 usages
    public void rentMovie(Long userId,Long movieId) {
        User user = userDAO.getById(userId);
        Movie movie = movieDAO.getById(movieId);

        if(user==null || movie==null) {
            throw new IllegalArgumentException("user or movie not found");
        }

        Long limit=movie.getUserLimit();
        if (limit != null) {
            long rentedNow=movieOrderDAO.countRentalsNow(movieId);
            if (rentedNow>=limit) {
                throw new IllegalArgumentException("Movie rental limit reached");
            }
        }
        MovieOrder order = new MovieOrder();
        order.setUser(user);
        order.setMovie(movie);
        order.setRented(true);
        LocalDateTime now = LocalDateTime.now();
        order.setDateOfOrder(now);

        movieOrderDAO.save(order);
    }
```

Poprawność:

```java
@BeforeEach
void setUp() {
    mockMovieOrderDAO = mock(MovieOrderDAO.class);
    mockMovieDAO = mock(MovieDAO.class);
    mockUserDAO = mock(UserDAO.class);

    movieOrderService = new MovieOrderService();

    movieOrderService.movieOrderDAO = mockMovieOrderDAO;
    movieOrderService.movieDAO = mockMovieDAO;
    movieOrderService.userDAO = mockUserDAO;

    user = new User();
    user.setID(1L);

    movie = new Movie();
    movie.setMovieID(1L);
    movie.setUserLimit(2L);

    when(mockUserDAO.getById(1L)).thenReturn(user);
    when(mockMovieDAO.getById(1L)).thenReturn(movie);
}
```

```java
    @Test
    void rentMovie_success() {
        // 1 wypozyczenie wiec limit 2 nie przekroczony
        when(mockMovieOrderDAO.countRentalsNow( movieId: 1L)).thenReturn( t: 1L);

        assertDoesNotThrow(() -> movieOrderService.rentMovie( userId: 1L,  movieId: 1L));

        verify(mockUserDAO).getById(1L);
        verify(mockMovieDAO).getById(1L);
        verify(mockMovieOrderDAO).countRentalsNow( movieId: 1L);
        verify(mockMovieOrderDAO).save(any(MovieOrder.class));
    }

    @Test
    void rentMovie_userNotFound() {
        when(mockUserDAO.getById(2L)).thenReturn( t: null);

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
            movieOrderService.rentMovie( userId: 2L,  movieId: 1L);
        });
        assertEquals( expected: "user or movie not found", ex.getMessage());

        verify(mockUserDAO).getById(2L);
        verify(mockMovieOrderDAO, never()).save(any());
    }
```

```java
    @Test
    void rentMovie_limitReached() {
        when(mockMovieOrderDAO.countRentalsNow( movieId: 1L)).thenReturn( t: 2L); // limit = 2, już osiągnięty

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
            movieOrderService.rentMovie( userId: 1L,  movieId: 1L);
        });
        assertEquals( expected: "Movie rental limit reached", ex.getMessage());

        verify(mockUserDAO).getById(1L);
        verify(mockMovieDAO).getById(1L);
        verify(mockMovieOrderDAO).countRentalsNow( movieId: 1L);
        verify(mockMovieOrderDAO, never()).save(any());
    }
}
```

ShowDAO

```java
public Show getByNameDirector(String name, String director) {  13 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    Show show = null;
    try {
        show = (Show) session.createQuery( s: "from Show where showName = :name AND director= :director", Show.class)
                .setParameter( s: "name", name)
                .setParameter( s: "director", director).uniqueResult();
    } finally {
        session.close();
    }
    return show;
}
```

ShowService

```java
public class ShowService {  3 usages
    public ShowDAO showDao = new ShowDAO();  13 usages
    public CategoryDAO categoryDao = new CategoryDAO();  2 usages
    public SubscriptionCategoryDAO subscriptionCategoryDao = new SubscriptionCategoryDAO();  2 usa

    public void addShow(String showName,  4 usages
                        String categoryName,
                        Long epNumbers,
                        Long ageRestriction,
                        String subscriptionCategoryName,
                        String description,
                        String director) {

        if (showDao.getByNameDirector(showName, director) != null){
            throw new IllegalArgumentException("Show already exists");
        }

        Category category = categoryDao.getCategoryByName(categoryName);
        if (category == null) {
            throw new IllegalArgumentException("Category not found");
        }

        SubscriptionCategory subscriptionCategory = null;
        if (subscriptionCategoryName != null) {
            subscriptionCategory = subscriptionCategoryDao.getByName(subscriptionCategoryName);
            if (subscriptionCategory == null) {
                throw new IllegalArgumentException("Subscription category not found");
            }
        }
```

```java
        Show show = new Show();
        show.setShowName(showName);
        show.setCategory(category);
        show.setEpNumbers(epNumbers);
        show.setAgeRestriction(ageRestriction);
        show.setSubscriptionCategory(subscriptionCategory);
        show.setDescription(description);
        show.setDirector(director);

        showDao.save(show);
    }
```

```java
public Show getShow(Long id) {  1 usage
    return showDao.getById(id);
}
public Show getShow(String showName, String director) {  2 usages
    return showDao.getByNameDirector(showName, director);
}

public List<Show> getAllShows() {  1 usage
    return showDao.getAll();
}

public void deleteShow(Long id) {  2 usages
    Show show = showDao.getById(id);
    if (show == null) {
        throw new IllegalArgumentException("Show not found");
    }
    showDao.delete(show);
}
public void deleteShow(String showName, String director) {  2 usages
    Show show = getShow(showName, director);
    if (show == null) {
        throw new IllegalArgumentException("Show not found");
    }
    showDao.delete(show);
}
```

```java
    public void updateEpNumbers(Long id, Long epNumbers) {  2 usages
        Show show = showDao.getById(id);
        if (show == null) {
            throw new IllegalArgumentException("Show not found");
        }
        show.setEpNumbers(epNumbers);
        showDao.update(show);
    }
    public void updateDirector(Long id, String director) {  2 usages
        Show show = showDao.getById(id);
        if (show == null) {
            throw new IllegalArgumentException("Show not found");
        }
        show.setDirector(director);
        showDao.update(show);
    }
}
```

Poprawność:

```java
    @Test
    public void addShow_success() {
        String showName = "Show A";
        String director = "Director A";
        String categoryName = "Comedy";
        String subscriptionCategoryName = "basic";

        when(mockShowDao.getByNameDirector(showName, director)).thenReturn( t: null);

        Category category = new Category();
        category.setCategoryName(categoryName);
        when(mockCategoryDao.getCategoryByName(categoryName)).thenReturn(category);

        SubscriptionCategory subscriptionCategory = new SubscriptionCategory();
        subscriptionCategory.setName(subscriptionCategoryName);
        when(mockSubscriptionCategoryDao.getByName(subscriptionCategoryName)).thenReturn(subscriptionCategory);

        showService.addShow(showName, categoryName,  epNumbers: 10L,  ageRestriction: 12L, subscriptionCategoryName,  description: "desc", director);

        verify(mockShowDao).getByNameDirector(showName, director);
        verify(mockCategoryDao).getCategoryByName(categoryName);
        verify(mockSubscriptionCategoryDao).getByName(subscriptionCategoryName);
        verify(mockShowDao).save(any(Show.class));
    }
```

```java
    @Test
    public void addShow_showAlreadyExists_shouldThrow() {
        when(mockShowDao.getByNameDirector( name: "Show B",  director: "Director B")).thenReturn(new Show());

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
            showService.addShow( showName: "Show B",  categoryName: "Drama",  epNumbers: 5L,  ageRestriction: 15L,
                    subscriptionCategoryName: null,  description: "desc",  director: "Director B");
        });

        assertEquals( expected: "Show already exists", ex.getMessage());
        verify(mockShowDao).getByNameDirector( name: "Show B",  director: "Director B");
        verify(mockShowDao, never()).save(any());
    }
```

```java
@Test
public void getShowById_success() {
    Show show = new Show();
    show.setShowName("Show E");

    when(mockShowDao.getById(1L)).thenReturn(show);

    Show result = showService.getShow( id: 1L);

    assertNotNull(result);
    assertEquals( expected: "Show E", result.getShowName());
}

@Test
public void getShowByNameDirector_success() {
    Show show = new Show();
    show.setShowName("Show F");

    when(mockShowDao.getByNameDirector( name: "Show F",  director: "Director F")).thenReturn(show);

    Show result = showService.getShow( showName: "Show F",  director: "Director F");

    assertNotNull(result);
    assertEquals( expected: "Show F", result.getShowName());
}
```

```java
@Test
public void deleteShowById_success() {
    Show show = new Show();
    when(mockShowDao.getById(1L)).thenReturn(show);

    showService.deleteShow( id: 1L);

    verify(mockShowDao).getById(1L);
    verify(mockShowDao).delete(show);
}

@Test
public void deleteShowById_notFound_shouldThrow() {
    when(mockShowDao.getById(2L)).thenReturn( t: null);

    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
        showService.deleteShow( id: 2L);
    });

    assertEquals( expected: "Show not found", ex.getMessage());
    verify(mockShowDao).getById(2L);
    verify(mockShowDao, never()).delete(any());
}
```

```java
    @Test
    public void updateEpNumbers_showNotFound_shouldThrow() {
        when(mockShowDao.getById(3L)).thenReturn( t: null);

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
            showService.updateEpNumbers( id: 3L,  epNumbers: 15L);
        });

        assertEquals( expected: "Show not found", ex.getMessage());
        verify(mockShowDao, never()).update(any());
    }

    @Test
    public void updateDirector_success() {
        Show show = new Show();
        when(mockShowDao.getById(1L)).thenReturn(show);

        showService.updateDirector( id: 1L,  director: "New Director");

        assertEquals( expected: "New Director", show.getDirector());
        verify(mockShowDao).update(show);
    }
```

SubscriptionCategoryDAO:

```java
public SubscriptionCategory getByName(String name) {  26 usages
    Session session = HibernateUtil.getSessionFactory().openSession();
    SubscriptionCategory category = null;
    try {
        Query<SubscriptionCategory> query = session.createQuery(
                s: "from SubscriptionCategory where name = :name", SubscriptionCategory.class);
        query.setParameter( s: "name", name);
        category = query.uniqueResult();
    } finally {
        session.close();
    }
    return category;
}
```

SubscriptionCategoryService:

```java
public class SubscriptionCategoryService {  2 usages
    public SubscriptionCategoryDAO dao=new SubscriptionCategoryDAO();
    public void addSubscriptionCategory(String name, Long price, String description) {  2 usages
        if (dao.getByName(name) != null) {
            throw new IllegalArgumentException("Subscription category with name '" + name + "' already exists.");
        }
        SubscriptionCategory subscriptionCategory=new SubscriptionCategory();
        subscriptionCategory.setName(name);
        subscriptionCategory.setPrice(price);
        subscriptionCategory.setDescription(description);
        dao.save(subscriptionCategory);
    }
    public SubscriptionCategory getSubscriptionCategory(Long id) {  1 usage
        SubscriptionCategory subscriptionCategory=dao.getById(id);
        return subscriptionCategory;
    }
    public SubscriptionCategory getSubscriptionCategory(String name) {  4 usages
        SubscriptionCategory subscriptionCategory=dao.getByName(name);
        return subscriptionCategory;
    }
    public void deleteSubscriptionCategory(Long id) {  2 usages
        SubscriptionCategory subscriptionCategory=dao.getById(id);
        if (subscriptionCategory == null) {
            throw new IllegalArgumentException("Subscription not found");
        }
        dao.delete(subscriptionCategory);
```

```java
public void deleteSubscriptionCategory(String name) {  no usages
    if (subscriptionCategory == null) {
        throw new IllegalArgumentException("Subscription not found");
    }
    dao.delete(subscriptionCategory);
}
public void updateSubscriptionCategory(String name, Long price, String description) {
    SubscriptionCategory subscriptionCategory=getSubscriptionCategory(name);
    if (subscriptionCategory == null) {
        throw new IllegalArgumentException("Subscription not found");
    }
    subscriptionCategory.setPrice(price);
    subscriptionCategory.setDescription(description);
    dao.update(subscriptionCategory);

}
public void updateSubscriptionCategoryPrice(String name, Long price) {  2 usages
    SubscriptionCategory subscriptionCategory=getSubscriptionCategory(name);
    if (subscriptionCategory == null) {
        throw new IllegalArgumentException("Subscription not found");
    }
    subscriptionCategory.setPrice(price);
    dao.update(subscriptionCategory);
}
public void updateSubscriptionCategoryDescription(String name, String description) {
    SubscriptionCategory subscriptionCategory=getSubscriptionCategory(name);
    if (subscriptionCategory == null) {
        throw new IllegalArgumentException("Subscription not found");
    }
    subscriptionCategory.setDescription(description);
    dao.update(subscriptionCategory);
```

Poprawność:

```java
@Test
void addSubscriptionCategory_success() {
    String name = "Premium";
    Long price = 100L;
    String description = "Premium subscription";

    when(dao.getByName(name)).thenReturn( t: null); // no existing category

    service.addSubscriptionCategory(name, price, description);

    ArgumentCaptor<SubscriptionCategory> captor = ArgumentCaptor.forClass(SubscriptionCategory.class);
    verify(dao).save(captor.capture());

    SubscriptionCategory saved = captor.getValue();
    assertEquals(name, saved.getName());
    assertEquals(price, saved.getPrice());
    assertEquals(description, saved.getDescription());
}
```

```java
@Test
void getSubscriptionCategoryByName_returnsCategory() {
    SubscriptionCategory cat = new SubscriptionCategory();
    cat.setName("TestName");

    when(dao.getByName("TestName")).thenReturn(cat);

    SubscriptionCategory result = service.getSubscriptionCategory( name: "TestName");
    assertNotNull(result);
    assertEquals( expected: "TestName", result.getName());
}

@Test
void deleteSubscriptionCategoryById_success() {
    SubscriptionCategory cat = new SubscriptionCategory();

    when(dao.getById(1L)).thenReturn(cat);

    service.deleteSubscriptionCategory( id: 1L);

    verify(dao).delete(cat);
}
```

```java
@Test
void updateSubscriptionCategoryPrice_success() {
    SubscriptionCategory cat = new SubscriptionCategory();
    cat.setName("Silver");

    when(dao.getByName("Silver")).thenReturn(cat);

    service.updateSubscriptionCategoryPrice( name: "Silver",  price: 300L);

    assertEquals( expected: 300L, cat.getPrice());
    verify(dao).update(cat);
}

@Test
void updateSubscriptionCategoryPrice_notFound_throws() {
    when(dao.getByName("Bronze")).thenReturn( t: null);

    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () ->
            service.updateSubscriptionCategoryPrice( name: "Bronze",  price: 100L)
    );
    assertTrue(ex.getMessage().contains("not found"));
    verify(dao, never()).update(any());
}
```

SubscriptionOrderDAO:

```
    public List<SubscriptionOrder> findByUserIdAndPaymentStatus(Long userId, String paymentStatus) {  7 usages
        Session session = HibernateUtil.getSessionFactory().openSession();
        List<SubscriptionOrder> results = session.createQuery( s: """
        FROM SubscriptionOrder so
        WHERE so.user.id = :userId
        AND so.paymentStatus = :paymentStatus
    """, SubscriptionOrder.class)
                .setParameter( s: "userId", userId)
                .setParameter( s: "paymentStatus", paymentStatus)
                .getResultList();
        session.close();
        return results;
    }
```

SubscriptionOrderService:

Jest to klasa pozwalająca na zakup subskrypcji. buySubscription używa transakcji aby wykupić subskrypcję, jeśli jeszcze jej nie ma.

```
public class SubscriptionOrderService {  12 usages

    public SubscriptionOrderDAO subscriptionOrderDAO = new SubscriptionOrderDAO();  3 usages
    public SubscriptionCategoryDAO subscriptionCategoryDAO = new SubscriptionCategoryDAO();  2 usages
    public UserDAO userDAO = new UserDAO();  2 usages

    @Transactional  6 usages
    public void buySubscription(Long userId, Long subscriptionCategoryId) {
        User user = userDAO.getById(userId);
        SubscriptionCategory category = subscriptionCategoryDAO.getById(subscriptionCategoryId);

        if (user == null || category == null) {
            throw new IllegalArgumentException("User or subscription category not found");
        }
        SubscriptionOrder activeSubscription = getActiveSubscription(userId);
        if (activeSubscription != null) {
            throw new IllegalStateException("User already has an active subscription");
        }

        SubscriptionOrder order = new SubscriptionOrder();
        order.setUser(user);
        order.setSubscriptionCategory(category);
        LocalDateTime now = LocalDateTime.now();
        order.setDateOfSubscription(now);
        order.setPaymentStatus("PAID");

        subscriptionOrderDAO.save(order);
    }
    private static final int SUBSCRIPTION_DAYS=30;  1 usage
```

```
    private static final int SUBSCRIPTION_DAYS=30;  1 usage

    public SubscriptionOrder getActiveSubscription(Long userId) {  9 usages
        List<SubscriptionOrder> orders = subscriptionOrderDAO.findByUserIdAndPaymentStatus(userId,  paymentStatus: "PAID");
        LocalDateTime now = LocalDateTime.now();
        return orders.stream()  Stream<SubscriptionOrder>
                .filter(order -> order.getDateOfSubscription().plusDays(SUBSCRIPTION_DAYS).isAfter(now))
                .max(Comparator.comparing(SubscriptionOrder::getDateOfSubscription))  Optional<SubscriptionOrder>
                .orElse( other: null);

    }
}
```

Poprawność:

```java
@Test
public void buySubscription_success() {
    Long userId = 1L;
    Long categoryId = 2L;

    User user = new User();
    user.setID(userId);

    SubscriptionCategory category = new SubscriptionCategory();
    category.setSubCategoryId(categoryId);

    when(mockUserDAO.getById(userId)).thenReturn(user);
    when(mockSubscriptionCategoryDAO.getById(categoryId)).thenReturn(category);

    subscriptionOrderService.buySubscription(userId, categoryId);

    verify(mockUserDAO).getById(userId);
    verify(mockSubscriptionCategoryDAO).getById(categoryId);
    verify(mockSubscriptionOrderDAO).save(any(SubscriptionOrder.class));
}
```

```java
@Test
public void buySubscription_subscriptionCategoryNotFound_throws() {
    Long userId = 1L;
    Long categoryId = 2L;

    when(mockUserDAO.getById(userId)).thenReturn(new User());
    when(mockSubscriptionCategoryDAO.getById(categoryId)).thenReturn( t: null);

    IllegalArgumentException ex = assertThrows(IllegalArgumentException.class, () -> {
        subscriptionOrderService.buySubscription(userId, categoryId);
    });

    assertEquals( expected: "User or subscription category not found", ex.getMessage());
    verify(mockUserDAO).getById(userId);
    verify(mockSubscriptionCategoryDAO).getById(categoryId);
    verify(mockSubscriptionOrderDAO, never()).save(any());
```

```java
private static final int SUBSCRIPTION_DAYS = 30;  no usages

@Test
public void getActiveSubscription_shouldReturnActiveSubscription() {
    Long userId = 1L;

    LocalDateTime now = LocalDateTime.now();
    SubscriptionOrder activeOrder = new SubscriptionOrder();
    activeOrder.setDateOfSubscription(now.minusDays(10));  // wciąż aktywna
    activeOrder.setPaymentStatus("PAID");

    SubscriptionOrder oldOrder = new SubscriptionOrder();
    oldOrder.setDateOfSubscription(now.minusDays(40));  // wygasła
    oldOrder.setPaymentStatus("PAID");

    when(mockSubscriptionOrderDAO.findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID"))
            .thenReturn(Arrays.asList(oldOrder, activeOrder));

    SubscriptionOrder result = subscriptionOrderService.getActiveSubscription(userId);

    assertNotNull(result);
    assertEquals(activeOrder.getDateOfSubscription(), result.getDateOfSubscription());
    verify(mockSubscriptionOrderDAO).findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID");
}
```

```java
@Test
public void getActiveSubscription_shouldReturnNullWhenNoSubscriptions() {
    Long userId = 1L;

    when(mockSubscriptionOrderDAO.findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID"))
            .thenReturn(Collections.emptyList());

    SubscriptionOrder result = subscriptionOrderService.getActiveSubscription(userId);

    assertNull(result);
    verify(mockSubscriptionOrderDAO).findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID");
}
```

```java
@Test
public void getActiveSubscription_shouldReturnNullWhenAllSubscriptionsExpired() {
    Long userId = 1L;

    LocalDateTime now = LocalDateTime.now();
    SubscriptionOrder expiredOrder1 = new SubscriptionOrder();
    expiredOrder1.setDateOfSubscription(now.minusDays(40));
    expiredOrder1.setPaymentStatus("PAID");

    SubscriptionOrder expiredOrder2 = new SubscriptionOrder();
    expiredOrder2.setDateOfSubscription(now.minusDays(31));
    expiredOrder2.setPaymentStatus("PAID");

    when(mockSubscriptionOrderDAO.findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID"))
            .thenReturn(Arrays.asList(expiredOrder1, expiredOrder2));

    SubscriptionOrder result = subscriptionOrderService.getActiveSubscription(userId);

    assertNull(result);
    verify(mockSubscriptionOrderDAO).findByUserIdAndPaymentStatus(userId, paymentStatus: "PAID");
}
```

5. Przykładowe dane

Korzystając z zaimplementowanych metod dodaliśmy do bazy pokazowe dane.

```java
// ADDING SUBSCRIPTION CATEGORIES
SubscriptionCategoryService subscriptionCategoryService = new SubscriptionCategoryService();
subscriptionCategoryService.addSubscriptionCategory( name: "basic", price: 10L, description: "basic subscription, hd, only movies available");
subscriptionCategoryService.addSubscriptionCategory( name: "premium", price: 15L, description: "allows you to watch shows and movies in full
subscriptionCategoryService.addSubscriptionCategory( name: "VIP", price: 50L, description: "allows you to watch shows and movies in full hd 4k,
// ADDING CATEGORIES
CategoryService categoryService = new CategoryService();
categoryService.addCategory( name: "Romance");
categoryService.addCategory( name: "Fantasy");
categoryService.addCategory( name: "Horror");
categoryService.addCategory( name: "Action");
categoryService.addCategory( name: "Comedy");
// ADDING USERS
UserService userService = new UserService();
userService.addUser( name: "Robert", surname: "Lewandowski", phoneNumber: "+48999999999", email: "probierz@hate.club",
        LocalDate.of( year: 1980, month: 1, dayOfMonth: 1));
userService.addUser( name: "Krzysztof", surname: "Gonciarz", phoneNumber: "+48122112212", email: "naruciak@jestglupi.pl",
        LocalDate.of( year: 1990, month: 11, dayOfMonth: 4));
userService.addUser( name: "Finn", surname: "Człowiek", phoneNumber: "+48500855008", email: "pora@naprzygo.de",
        LocalDate.of( year: 2005, month: 5, dayOfMonth: 14));
userService.addUser( name: "Pies", surname: "Jake", phoneNumber: "+48123456789", email: "panna@jednorozek.love",
        LocalDate.of( year: 2010, month: 7, dayOfMonth: 19));
userService.addUser( name: "Lodowy", surname: "Król", phoneNumber: "+48666666666", email: "szymonkrol@lodziarnia.ice",
        LocalDate.of( year: 1000, month: 1, dayOfMonth: 5));
```

```
//      ADDING EPISODES
ShowEpisodeService showEpisodeService = new ShowEpisodeService();
showEpisodeService.addEpisode( showId: 1L, episodeNumber: 1, title: "Tom and Jerry: Barnyard Bunk",LocalDate.of( year: 1930, month: 1, dayOfMonth: 1
        duration: 5, episodePath: "shows/Tom_and_Jerry_Barnyard_Bunk_1932_512kb.mp4", description: "Ep1 adventure");
showEpisodeService.addEpisode( showId: 1L, episodeNumber: 2, title: "Tom and Jerry: Jolly Fish",LocalDate.of( year: 1930, month: 1, dayOfMonth: 1),
        duration: 5, episodePath: "shows/Tom_and_Jerry_Jolly_Fish_1932_512kb.mp4", description: "Ep2 adventure");

//ADDING SHOWS
ShowService showService = new ShowService();
showService.addShow( showName: "Tom and Jerry", categoryName: "Comedy", epNumbers: 2L, ageRestriction: 7L, subscriptionCategoryName: "basic",
        description: "Our beloved Tom's and Jerry's take part in various adventures!!", director: "John Doe");
//ADDING MOVIES
MovieService movieService = new MovieService();
movieService.addMovie( name: "The Phantom Of The Opera",  categoryName: "Horror", ageRestriction: 16,
        subscriptionAvailable: true, subscriptionCategory: "premium", rentPrice: 5L, moviePath: "movies/ThePhantomOfTheOperaPresentedByMoviePowder_512kb.
, userLimit: null,LocalDate.of( year: 1925, month: 1, dayOfMonth: 1), director: "Julian Rupert", description: "At the Opera of Paris, a mysterious phan
                "threatens a famous lyric singer, Carlotta and thus forces her to give up her role (Marguerite in Faust) for unknown Chr
                "this phantom (a masked man) in the catacombs, where he lives.");
movieService.addMovie( name: "Peter Pan", categoryName: "Fantasy", ageRestriction: 7, subscriptionAvailable: false, subscriptionCategory: null, rentPrice: 5L,
        moviePath: "movies/Peter Pan.mp4", userLimit: 2L,LocalDate.of( year: 1924, month: 1, dayOfMonth: 1),
        director: "Herbert Brenon", description: "Peter Pan is a 1924 American silent fantasy adventure film released by Paramount Pictures T
movieService.addMovie( name: "Number Please?", categoryName: "Comedy", ageRestriction: 12,
        subscriptionAvailable: true, subscriptionCategory: "basic", rentPrice: 5L, moviePath: "movies/Number_Please_512kb.mp4",
        userLimit: null,LocalDate.of( year: 1920, month: 1, dayOfMonth: 1), director: "Hal Roach",  description: "Number, Please? is a 1920 American s
```

Co zaskutkowało pojawieniem się danych w bazie, np. tabela Users:

	USER_ID	NAME	SURNAME	PHONE_NUMBER	EMAIL	BIRTH_DATE	JOINED_DATE
1	1	Jan	Kowalski	+11123456789	janek@gmail.com	2000-01-01	2025-06-15
2	2	Robert	Lewandowski	+48999999999	probierz@hate.club	1980-01-01	2025-06-15
3	3	Krzysztof	Gonciarz	+48122112212	naruciak@jestglupi.pl	1990-11-04	2025-06-15
4	4	Finn	Człowiek	+48500855008	pora@naprzygo.de	2005-05-14	2025-06-15
5	5	Pies	Jake	+48123456789	panna@jednorozek.love	2010-07-19	2025-06-15
6	6	Lodowy	Król	+48666666666	szymonkrol@lodziarnia.ice	1000-01-05	2025-06-15

Episodes:

	EPISODE_ID	SERIES_ID	EPISODE_NUMBER	EPISODE_TITLE	EPISODE_PATH
1	1	1	1	Tom and Jerry: Barnyard Bunk	shows/Tom_and_Jerry_Barnyard_Bunk_1932_51
2	2	1	2	Tom and Jerry: Jolly Fish	shows/Tom_and_Jerry_Jolly_Fish_1932_512kb

DURATION_MINUTES	RELEASE_DATE	DESCRIPTION
5	1930-01-01	Ep1 adventure
5	1930-01-01	Ep2 adventure

Gdzie series_id =1 odpowiada w series:

	SERIES_ID	SERIES_NAME	CATEGORY_ID	EP_NUMBERS	AGE_RESTRICTION	SUB_CATEGORY_ID
1	1	Tom and Jerry	25	2	7	1

DESCRIPTION	DIRECTOR
Our beloved Tom's and Jerry's take part in various…	John Doe

Dodaliśmy również zamównienia:

```java
public static void main(String[] args) {
    SubscriptionOrderService subscriptionOrderService = new SubscriptionOrderService();
    try {
        subscriptionOrderService.buySubscription( userId: 3L,  subscriptionCategoryId: 1L);
        System.out.println("Subskrypcja zakupiona pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy zakupie subskrypcji: " + e.getMessage());
    }
    try {
        subscriptionOrderService.buySubscription( userId: 5L,  subscriptionCategoryId: 3L);
        System.out.println("Subskrypcja zakupiona pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy zakupie subskrypcji: " + e.getMessage());
    }
    try {
        subscriptionOrderService.buySubscription( userId: 1L,  subscriptionCategoryId: 2L);
        System.out.println("Subskrypcja zakupiona pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy zakupie subskrypcji: " + e.getMessage());
    }
```

```java
    MovieOrderService movieOrderService = new MovieOrderService();
    try {
        movieOrderService.rentMovie( userId: 1L,  movieId: 3L);
        System.out.println("Film wypożyczony pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy wypożyczeniu filmu: " + e.getMessage());
    }
    try {
        movieOrderService.rentMovie( userId: 2L,  movieId: 3L);
        System.out.println("Film wypożyczony pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy wypożyczeniu filmu: " + e.getMessage());
    }
    try {
        movieOrderService.rentMovie( userId: 4L,  movieId: 5L);
        System.out.println("Film wypożyczony pomyślnie");
    } catch (IllegalArgumentException e) {
        System.err.println("Błąd przy wypożyczeniu filmu: " + e.getMessage());
    }
```

Po czym nasza tabela subscriptions_orders wygląda tak:

	SUBSCRIPTION_ORDER_ID	DATE_OF_SUBSCRIPTION	PAYMENT_STATUS	SUBSCRIPTION_TYPE	USER_ID
1	1	2025-06-16 00:52:48.070918	PAID	1	3
2	2	2025-06-16 00:52:48.107204	PAID	3	5
3	3	2025-06-16 00:52:48.114284	PAID	2	1

A movies_orders:

	MOVIE_ORDER_ID	MOVIE_ID	RENTED	DATE_OF_ORDER	USER_ID
1	1	3	1	2025-06-16 00:50:50.270173	1
2	2	3	1	2025-06-16 00:50:50.325742	2
3	3	5	1	2025-06-16 00:50:50.792809	4

6. Raporty

Stworzyliśmy klasę RaportService z zaimplementowanymi funkcjami raportów. Dodatkowo dla niektórych raportów zaimplementowaliśmy klasy pomocnicze DTO (Data Transfer Object) dzięki którym nie zwracamy obiektów tylko czytelnie sformatowane dane.

Klasy pomocnicze:

```
public class ActiveSubscriptionByCategoryDTO {  10 usages
    private String categoryName;  3 usages
    private Long activeSubscriptionsCount;  3 usages

    public ActiveSubscriptionByCategoryDTO(String categoryName, Long activeSubscriptionsCount) {  3 usages
        this.categoryName = categoryName;
        this.activeSubscriptionsCount = activeSubscriptionsCount;
    }

    public String getCategoryName() {
        return categoryName;
    }

    public Long getActiveSubscriptionsCount() {  1 usage
        return activeSubscriptionsCount;
    }

    @Override
    public String toString() {
        return "Category: " + categoryName + ", Active subscriptions: " + activeSubscriptionsCount;
    }
}
```

```java
public class MovieRentalRevenueByCategoryDTO {  10 usages
    private String categoryName;  3 usages
    private Long totalRevenue;  3 usages

    public MovieRentalRevenueByCategoryDTO(String categoryName, Long totalRevenue) {  3 usage
        this.categoryName = categoryName;
        this.totalRevenue = totalRevenue;
    }

    public String getCategoryName() { return categoryName; }

    public Long getTotalRevenue() { return totalRevenue; }

    @Override
    public String toString() {
        return "MovieRentalRevenueByCategoryDTO{" +
                "categoryName='" + categoryName + '\'' +
                ", totalRevenue=" + totalRevenue +
                '}';
    }
}
```

```java
public class MoviesCountByCategoryDTO {  10 usages
    private String categoryName;  3 usages
    private Long moviesCount;  3 usages

    public MoviesCountByCategoryDTO(String categoryName, Long moviesCount) {  3 usage
        this.categoryName = categoryName;
        this.moviesCount = moviesCount;
    }

    public String getCategoryName() {
        return categoryName;
    }

    public Long getMoviesCount() {  1 usage
        return moviesCount;
    }

    @Override
    public String toString() {
        return "Category: " + categoryName + ", Movies count: " + moviesCount;
    }
}
```

RaportService:

```java
public class RaportService {  3 usages
    private final SessionFactory sessionFactory;  7 usages
    public RaportService(SessionFactory sessionFactory) {  1 usage
        this.sessionFactory = sessionFactory;
    }
    private SubscriptionOrderDAO subscriptionOrderDAO=new SubscriptionOrderDAO();  no usages
    private MovieDAO movieDAO=new MovieDAO();  no usages
    private UserDAO userDAO=new UserDAO();  no usages

    // aktywne subskrypcje wg kategorii
    public List<ActiveSubscriptionByCategoryDTO> activeSubscriptionsByCategory() {  1 usage
        Session session = sessionFactory.openSession();
        List<ActiveSubscriptionByCategoryDTO> results = session.createQuery( s: """
        select new service.raportsUtil.ActiveSubscriptionByCategoryDTO(so.subscriptionCategory.name, count(so))
        from SubscriptionOrder so
        where so.paymentStatus = 'PAID'
        group by so.subscriptionCategory.name
        """, ActiveSubscriptionByCategoryDTO.class).getResultList();
        session.close();
        return results;
    }
```

```java
    public List<Movie> moviesBySubscriptionCategory(String subscriptionCategoryName) {  1 usage
        Session session = sessionFactory.openSession();
        List<Movie> results = session.createQuery( s: """
        select m
        from Movie m
        where m.subscriptionCategory.name = :subCatName
        """, Movie.class)
                .setParameter( s: "subCatName", subscriptionCategoryName)
                .getResultList();
        session.close();
        return results;
    }
```

```java
public List<Show> showsBySubscriptionCategory(String subscriptionCategoryName) {  1 usage
    Session session = sessionFactory.openSession();
    List<Show> results = session.createQuery( s: """
    select s
    from Show s
    where s.subscriptionCategory.name = :subCatName
    """, Show.class)
            .setParameter( s: "subCatName", subscriptionCategoryName)
            .getResultList();
    session.close();
    return results;
}

//liczba filmow w kategoriach
public List<MoviesCountByCategoryDTO> moviesCountByCategory() {  1 usage
    Session session = sessionFactory.openSession();
    List<MoviesCountByCategoryDTO> results = session.createQuery( s: """
    select new service.raportsUtil.MoviesCountByCategoryDTO(m.category.categoryName, count(m))
    from Movie m
    group by m.category.categoryName
    """, MoviesCountByCategoryDTO.class).getResultList();
    session.close();
    return results;
}
```

```java
//uzytkownicy bez subskrypcji
public List<User> usersWithoutActiveSubscription() {  1 usage
    Session session = sessionFactory.openSession();
    List<User> results = session.createQuery( s: """
    select u
    from User u
    where not exists (
        select 1 from SubscriptionOrder so
        where so.user.id = u.id and so.paymentStatus = 'PAID'
    )
    """, User.class).getResultList();
    session.close();
    return results;
}
public List<MovieRentalRevenueByCategoryDTO> rentalRevenueByCategory() {  1 usage
    Session session = sessionFactory.openSession();
    // suma rentPrice grupujac po kategorii
    List<MovieRentalRevenueByCategoryDTO> results = session.createQuery( s: """
    select new service.raportsUtil.MovieRentalRevenueByCategoryDTO(
        m.category.categoryName,
        sum(m.rentPrice)
    )
    from MovieOrder mo
    join mo.movie m
    where mo.rented = true
    group by m.category.categoryName
    """, MovieRentalRevenueByCategoryDTO.class)
            .getResultList();

    session.close();
    return results;
```

Poprawność (testy):

```java
@Test
void activeSubscriptionsByCategory_returnsCorrectList() {
    Query<ActiveSubscriptionByCategoryDTO> mockQuery = mock(Query.class);
    List<ActiveSubscriptionByCategoryDTO> mockResult = List.of(
            new ActiveSubscriptionByCategoryDTO( categoryName: "Basic",  activeSubscriptionsCount: 10L),
            new ActiveSubscriptionByCategoryDTO( categoryName: "Premium",  activeSubscriptionsCount: 5L)
    );

    when(mockSession.createQuery(anyString(), eq(ActiveSubscriptionByCategoryDTO.class))).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockResult);

    List<ActiveSubscriptionByCategoryDTO> result = raportService.activeSubscriptionsByCategory();

    assertEquals( expected: 2, result.size());
    assertEquals( expected: "Basic", result.get(0).getCategoryName());
    assertEquals( expected: 10L, result.get(0).getActiveSubscriptionsCount());
    verify(mockSession).close();
}
```

```java
@Test
void moviesBySubscriptionCategory_returnsCorrectMovies() {
    Query<Movie> mockQuery = mock(Query.class);
    List<Movie> mockMovies = List.of(new Movie(), new Movie());

    when(mockSession.createQuery(anyString(), eq(Movie.class))).thenReturn(mockQuery);
    when(mockQuery.setParameter(anyString(), any())).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockMovies);

    List<Movie> result = raportService.moviesBySubscriptionCategory( subscriptionCategoryName: "Premium");

    assertEquals( expected: 2, result.size());
    verify(mockQuery).setParameter( s: "subCatName",  o: "Premium");
    verify(mockSession).close();
}
```

```java
@Test
void showsBySubscriptionCategory_returnsCorrectShows() {
    Query<Show> mockQuery = mock(Query.class);
    List<Show> mockShows = List.of(new Show());

    when(mockSession.createQuery(anyString(), eq(Show.class))).thenReturn(mockQuery);
    when(mockQuery.setParameter(anyString(), any())).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockShows);

    List<Show> result = raportService.showsBySubscriptionCategory( subscriptionCategoryName: "Premium");

    assertEquals( expected: 1, result.size());
    verify(mockQuery).setParameter( s: "subCatName",  o: "Premium");
    verify(mockSession).close();
}
```

```java
@Test
void moviesCountByCategory_returnsCorrectDTOs() {
    Query<MoviesCountByCategoryDTO> mockQuery = mock(Query.class);
    List<MoviesCountByCategoryDTO> mockDTOs = List.of(
            new MoviesCountByCategoryDTO( categoryName: "Action",  moviesCount: 12L),
            new MoviesCountByCategoryDTO( categoryName: "Comedy",  moviesCount: 7L)
    );

    when(mockSession.createQuery(anyString(), eq(MoviesCountByCategoryDTO.class))).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockDTOs);

    List<MoviesCountByCategoryDTO> result = raportService.moviesCountByCategory();

    assertEquals( expected: 2, result.size());
    assertEquals( expected: "Action", result.get(0).getCategoryName());
    assertEquals( expected: 12L, result.get(0).getMoviesCount());
    verify(mockSession).close();
}
```

```java
@Test
void usersWithoutActiveSubscription_returnsUsers() {
    Query<User> mockQuery = mock(Query.class);
    List<User> mockUsers = List.of(new User(), new User());

    when(mockSession.createQuery(anyString(), eq(User.class))).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockUsers);

    List<User> result = raportService.usersWithoutActiveSubscription();

    assertEquals( expected: 2, result.size());
    verify(mockSession).close();
}
```

```java
@Test
void rentalRevenueByCategory_returnsCorrectDTOs() {
    Query<MovieRentalRevenueByCategoryDTO> mockQuery = mock(Query.class);
    List<MovieRentalRevenueByCategoryDTO> mockDTOs = List.of(
            new MovieRentalRevenueByCategoryDTO( categoryName: "Drama",  totalRevenue: 15000L),
            new MovieRentalRevenueByCategoryDTO( categoryName: "Thriller",  totalRevenue: 9000L)
    );

    when(mockSession.createQuery(anyString(), eq(MovieRentalRevenueByCategoryDTO.class))).thenReturn(mockQuery);
    when(mockQuery.getResultList()).thenReturn(mockDTOs);

    List<MovieRentalRevenueByCategoryDTO> result = raportService.rentalRevenueByCategory();

    assertEquals( expected: 2, result.size());
    assertEquals( expected: "Drama", result.get(0).getCategoryName());
    assertEquals( expected: 15000L, result.get(0).getTotalRevenue());
    verify(mockSession).close();
}
}
```

Wywołanie raportów zrobiliśmy w mainie, po zastanowieniu doszliśmy do wniosku, że należałoby zrobić do tego osobne klasy (nie zdążyliśmy tego zrobić ale wnioski zostały wyciągnięte).

```java
public class Main {
    public static void main(String[] args) {

        SessionFactory sessionFactory= HibernateUtil.getSessionFactory();
        RaportService  raportService= new RaportService(sessionFactory);
        try {
            List<ActiveSubscriptionByCategoryDTO> activeSubs = raportService.activeSubscriptionsByCategory();
            System.out.println("=== Aktywne subskrypcje wg kategorii ===");
            activeSubs.forEach(dto ->
                    System.out.println(dto.getCategoryName() + ": " + dto.getActiveSubscriptionsCount())
            );
```

Output:

```
=== Aktywne subskrypcje wg kategorii ===
basic: 1
premium: 1
VIP: 1
```

```java
List<Movie> movies = raportService.moviesBySubscriptionCategory( subscriptionCategoryName: "basic");
System.out.println("\n=== Filmy dla kategorii 'basic' ===");
movies.forEach(m -> System.out.println(m.getMovieName()));
```

```
=== Filmy dla kategorii 'basic' ===
Number Please?
```

```java
List<MoviesCountByCategoryDTO> movieCounts = raportService.moviesCountByCategory();
System.out.println("\n=== Liczba filmów wg kategorii ===");
movieCounts.forEach(dto ->
        System.out.println(dto.getCategoryName() + ": " + dto.getMoviesCount())
);
```

```
=== Liczba filmów wg kategorii ===
Fantasy: 1
Horror: 1
Comedy: 1
```

```java
  List<User> usersNoSub = raportService.usersWithoutActiveSubscription();
  System.out.println("\n=== Użytkownicy bez aktywnej subskrypcji ===");
  usersNoSub.forEach(u -> System.out.println(u.getEmail()));
```

```
=== Użytkownicy bez aktywnej subskrypcji ===
probierz@hate.club
pora@naprzygo.de
szymonkrol@lodziarnia.ice
```

```java
    List<MovieRentalRevenueByCategoryDTO> revenues = raportService.rentalRevenueByCategory();
    System.out.println("\n=== Przychody z wypożyczeń filmów wg kategorii ===");
    revenues.forEach(dto ->
            System.out.println(dto.getCategoryName() + ": " + dto.getTotalRevenue())
    );
} finally {
    sessionFactory.close();
}
```

```
=== Przychody z wypożyczeń filmów wg kategorii ===
Horror: 10
Fantasy: 5
```