МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики  
Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА № 1

**Проектирование приложения**

по курсу   
Технологии сетевого программирования

Группа 6302-010302D

Студент О.В. Удалова

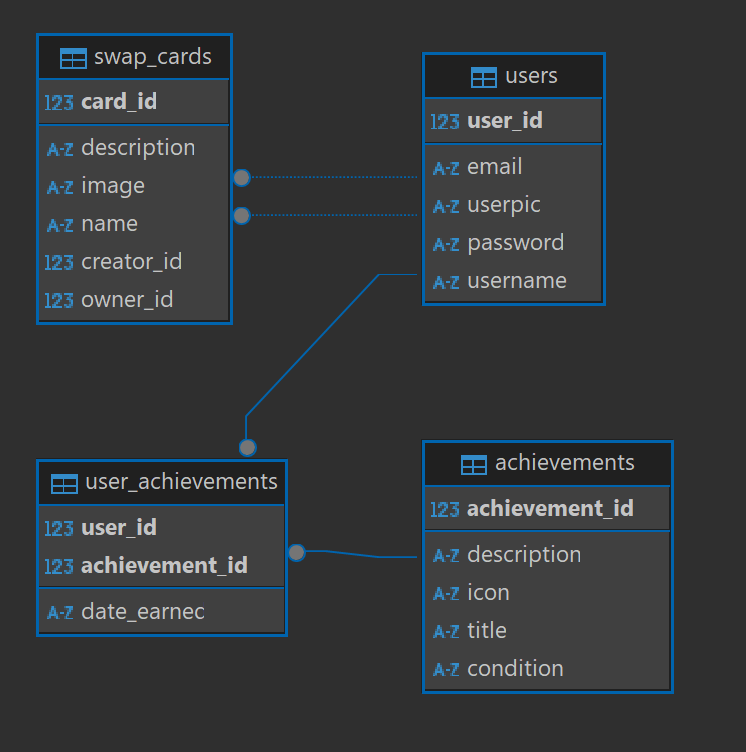
(*подпись*)

Самара 2025

## Описание структуры базы данных

Структура базы данных проекта построена с использованием PostgreSQL и включает в себя следующие основные таблицы:

* **users** — хранит информацию о пользователях.
  + user\_id (INTEGER, PRIMARY KEY, AUTO\_INCREMENT) — уникальный идентификатор пользователя.
  + username (VARCHAR, NOT NULL) — имя пользователя.
  + userpic (VARCHAR) — ссылка на изображение профиля.
  + email (VARCHAR, UNIQUE, NOT NULL) — адрес электронной почты.
  + password (VARCHAR, NOT NULL) — хешированный пароль.
* **swap\_cards** — содержит информацию о картах, созданных пользователями.
  + card\_id (INTEGER, PRIMARY KEY, AUTO\_INCREMENT) — уникальный идентификатор карты.
  + name (VARCHAR, NOT NULL) — название карты.
  + image (VARCHAR, NOT NULL) — ссылка на изображение карты.
  + description (TEXT) — описание карты.
  + creator\_id (INTEGER, FOREIGN KEY -> users.user\_id, NOT NULL) — идентификатор создателя карты.
  + owner\_id (INTEGER, FOREIGN KEY -> users.user\_id, NOT NULL) — идентификатор текущего владельца карты.
* **achievements** — содержит информацию о достижениях.
  + achievement\_id (INTEGER, PRIMARY KEY, AUTO\_INCREMENT) — уникальный идентификатор достижения.
  + title (VARCHAR, NOT NULL) — название достижения.
  + icon (VARCHAR, NOT NULL) — ссылка на изображение.
  + description (TEXT, NOT NULL) — описание достижения.
  + condition (VARCHAR, NOT NULL) — условие получения достижения.
* **user\_achievements** — таблица для связывания пользователей и их достижений.
  + user\_id (INTEGER, FOREIGN KEY -> users.user\_id, PRIMARY KEY) — идентификатор пользователя.
  + achievement\_id (INTEGER, FOREIGN KEY -> achievements.achievement\_id, PRIMARY KEY) — идентификатор достижения.
  + date\_earned (DATE, NOT NULL) — дата получения достижения.

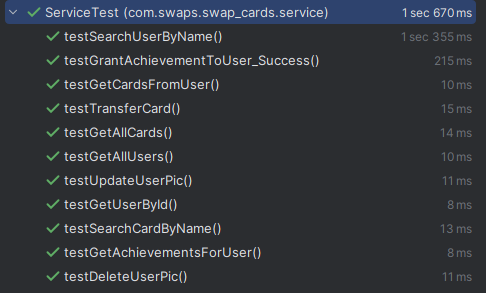


## Проделанная работа

1. Был установлен Docker, в котором была развёрнута база данных. За image был взят postgres 13.3.
2. Был установлен графический интерфейс DBeaver для работы с базой данных вручную и просмотра изменений. Подключение к базе данных было успешно установлено.
3. Через spring boot была создана основа приложения, в которой дальше и велась разработка.
4. Были настроены pom.xml, application.properties. Подключение к базе данных:

spring.application.name=Swap Cards  
spring.datasource.url=jdbc:postgresql://localhost:5432/swapcards  
spring.datasource.username=jaba  
spring.datasource.password=12345  
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true

1. Был создан package entity и написаны классы для каждой таблицы из базы данных (все скрипты приведены в следующей части работы). Для UserAchievement все же был использован составной ключ.
2. Для работы с данными был создан package service и в нем: UserService.java, SwapCardService.java, AchievementService.java. Для UserAchievement сервиса не создавалось, так как эта таблица служит связью между user и achievement, поэтому она изменялась/дополнялась в их методах.
3. Для проверки работоспособности было написано два теста: DatabaseFillTest (Для заполнения тестовыми данными и проверки корректности), ServiceTest (для проверки методов работы с данными). Также все достижения были заполнены вручную через DBeaver, так как для них не было реализовано отдельного функционала создания через service (и в будущем это и не требуется, достижения являются статичной таблицей).
4. После нескольких корректировок, все тесты были пройдены успешно.



## Скрипты

**Package entity.**

Achievement.java

package com.swaps.swap\_cards.entity;  
  
import jakarta.persistence.\*;  
import java.io.Serializable;  
  
@Entity  
@Table(name = "achievements")  
public class Achievement implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "achievement\_id")  
 private Integer id;  
  
 @Column(name = "title", nullable = false)  
 private String title;  
  
 @Column(name = "icon", nullable = false)  
 private String linkToIcon;  
  
 @Column(name = "description", nullable = false)  
 private String description;  
  
 @Column(name = "condition", nullable = false)  
 private String condition;  
  
 public Achievement() { }  
  
 public Integer getId() { return id; }  
 public String getTitle() { return title; }  
 public String getLinkToIcon() { return linkToIcon; }  
 public String getDescription() { return description; }  
 public String getCondition() { return condition; }  
  
 public void setTitle(String title) { this.title = title; }  
 public void setLinkToIcon(String linkToIcon) { this.linkToIcon = linkToIcon; }  
 public void setDescription(String description) { this.description = description; }  
 public void setCondition(String condition) { this.condition = condition; }  
}

SwapCard.java

package com.swaps.swap\_cards.entity;  
  
import jakarta.persistence.\*;  
import java.io.Serializable;  
  
@Entity  
@Table(name = "swap\_cards")  
public class SwapCard implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "card\_id")  
 private Integer id;  
  
 @Column(name = "name", nullable = false)  
 private String name;  
  
 @Column(name = "image", nullable = false)  
 private String linkToImage;  
  
 @Column(name = "description")  
 private String description;  
  
 @ManyToOne  
 @JoinColumn(name = "creator\_id", nullable = false)  
 private User creator;  
  
 @ManyToOne  
 @JoinColumn(name = "owner\_id", nullable = false)  
 private User owner;  
  
 public SwapCard() { }  
  
 public Integer getId() { return id; }  
 public String getName() { return name; }  
 public String getLinkToImage() { return linkToImage; }  
 public String getDescription() { return description; }  
 public User getCreator() { return creator; }  
 public User getOwner() { return owner; }  
  
 public void setName(String name) { this.name = name; }  
 public void setLinkToImage(String linkToImage) { this.linkToImage = linkToImage; }  
 public void setDescription(String description) { this.description = description; }  
 public void setOwner(User owner) { this.owner = owner; }  
 public void setCreator(User creator) { this.creator = creator; }  
}

User.java

package com.swaps.swap\_cards.entity;  
  
import com.swaps.swap\_cards.util.PasswordUtil;  
import jakarta.persistence.\*;  
import java.io.Serializable;  
  
@Entity  
@Table(name = "users")  
public class User implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "user\_id")  
 private Integer id;  
  
 @Column(name = "username", nullable = false)  
 private String userName;  
  
 @Column(name = "userpic")  
 private String linkToUserPic;  
  
 @Column(name = "email", nullable = false, unique = true)  
 private String email;  
  
 @Column(name = "password", nullable = false)  
 private String password;  
  
 public User() { }  
  
 public Integer getId() { return id; }  
 public String getUserName() { return userName; }  
 public String getLinkToUserPic() { return linkToUserPic; }  
 public String getEmail() { return email; }  
 public String getPassword() { return password; }  
  
 public void setUserName(String userName) { this.userName = userName; }  
 public void setLinkToUserPic(String linkToUserPic) { this.linkToUserPic = linkToUserPic; }  
 public void setEmail(String email) { this.email = email; }  
 public void setPassword(String rawPassword) {  
 this.password = PasswordUtil.*hashPassword*(rawPassword);  
 }  
}

UserAchievement.java

package com.swaps.swap\_cards.entity;  
  
import jakarta.persistence.\*;  
import java.io.Serializable;  
import java.time.LocalDate;  
  
@Entity  
@Table(name = "user\_achievements")  
public class UserAchievement implements Serializable {  
 @EmbeddedId  
 private UserAchievementId id;  
  
 @Column(name = "date\_earned", nullable = false)  
 private LocalDate dateEarned;  
  
 public UserAchievement() {  
 this.id = new UserAchievementId();  
 }  
  
 public UserAchievementId getId() { return id; }  
 public LocalDate getDateEarned() { return dateEarned; }  
  
 public void setUser(User user) {  
 if (this.id == null) this.id = new UserAchievementId();  
 this.id.user = user;  
 }  
 public void setAchievement(Achievement achievement) {  
 if (this.id == null) this.id = new UserAchievementId();  
 this.id.achievement = achievement;  
 }  
 public void setDateEarned(LocalDate dateEarned) { this.dateEarned = dateEarned; }  
  
 @Embeddable  
 public static class UserAchievementId implements Serializable {  
 @ManyToOne  
 @JoinColumn(name = "user\_id")  
 private User user;  
  
 @ManyToOne  
 @JoinColumn(name = "achievement\_id")  
 private Achievement achievement;  
  
 public UserAchievementId() { }  
  
 }  
}

**Package service.**

AchievementService.java

package com.swaps.swap\_cards.service;  
  
import com.swaps.swap\_cards.entity.Achievement;  
import com.swaps.swap\_cards.entity.UserAchievement;  
import com.swaps.swap\_cards.entity.User;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.PersistenceContext;  
import jakarta.transaction.Transactional;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.List;  
  
@Service  
public class AchievementService {  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 public Achievement getAchievementById(Integer id) {  
 return entityManager.find(Achievement.class, id);  
 }  
  
 public List<Achievement> getAchievementsForUser(Integer userId) {  
 String query = "SELECT ua.id.achievement FROM UserAchievement ua WHERE ua.id.user.id = :userId";  
 return entityManager.createQuery(query, Achievement.class)  
 .setParameter("userId", userId)  
 .getResultList();  
 }  
  
 public boolean checkAchievementCondition(User user, Achievement achievement) {  
 Long giftCount = entityManager.createQuery(  
 "SELECT COUNT(c) FROM SwapCard c WHERE c.creator.id = :userId AND c.owner.id != :userId", Long.class)  
 .setParameter("userId", user.getId())  
 .getSingleResult();  
  
 Long createCount = entityManager.createQuery(  
 "SELECT COUNT(c) FROM SwapCard c WHERE c.creator.id = :userId", Long.class)  
 .setParameter("userId", user.getId())  
 .getSingleResult();  
  
 Long haveCount = entityManager.createQuery(  
 "SELECT COUNT(c) FROM SwapCard c WHERE c.owner.id = :userId", Long.class)  
 .setParameter("userId", user.getId())  
 .getSingleResult();  
  
 Long achievementCount = entityManager.createQuery(  
 "SELECT COUNT(ua) FROM UserAchievement ua WHERE ua.id.user.id = :userId", Long.class)  
 .setParameter("userId", user.getId())  
 .getSingleResult();  
  
 Long haveForeignCount = entityManager.createQuery(  
 "SELECT COUNT(c) FROM SwapCard c WHERE c.owner.id = :userId AND c.creator.id != :userId", Long.class)  
 .setParameter("userId", user.getId())  
 .getSingleResult();  
  
 return switch (achievement.getCondition()) {  
 case "create 5 cards" -> createCount >= 5;  
 case "create first card" -> createCount >= 1;  
 case "gift your card" -> giftCount >= 1;  
 case "get card from another user" -> haveForeignCount >= 1;  
 case "have 10 cards" -> haveCount >= 10;  
 case "get 5 achievements" -> achievementCount >= 5;  
 case "get 10 cards from another users" -> haveForeignCount >= 10;  
 case "have 50 cards" -> haveCount >= 50;  
 case "gift your cards to 5 users" -> giftCount >= 5;  
 case "create account" -> true;  
  
 default -> false;  
 };  
 }  
  
 @Transactional  
 public void grantAchievementToUser(User user, Achievement achievement) {  
 if (checkAchievementCondition(user, achievement)) {  
 UserAchievement userAchievement = entityManager.createQuery(  
 "SELECT ua FROM UserAchievement ua WHERE ua.id.user.id = :userId AND ua.id.achievement.id = :achievementId", UserAchievement.class)  
 .setParameter("userId", user.getId())  
 .setParameter("achievementId", achievement.getId())  
 .getResultList()  
 .stream()  
 .findFirst()  
 .orElse(null);  
  
 if (userAchievement == null) {  
 userAchievement = new UserAchievement();  
 userAchievement.setUser(user);  
 userAchievement.setAchievement(achievement);  
 userAchievement.setDateEarned(LocalDate.*now*());  
  
 entityManager.persist(userAchievement);  
 }  
 } else {  
 throw new IllegalArgumentException("You have not met the conditions for the achievement");  
 }  
 }  
}

SwapCardService.java

package com.swaps.swap\_cards.service;  
  
import com.swaps.swap\_cards.entity.SwapCard;  
import com.swaps.swap\_cards.entity.User;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.PersistenceContext;  
import jakarta.transaction.Transactional;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class SwapCardService {  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 @Transactional  
 public SwapCard createSwapCard(String name, String linkToImage, String description, User creator) {  
 SwapCard swapCard = new SwapCard();  
 swapCard.setName(name);  
 swapCard.setDescription(description);  
 swapCard.setLinkToImage(linkToImage);  
 swapCard.setCreator(creator);  
 swapCard.setOwner(creator);  
 entityManager.persist(swapCard);  
 return swapCard;  
 }  
  
 @Transactional  
 public void transferCard(Integer cardId, User currentOwner, User newOwner) {  
 SwapCard card = entityManager.find(SwapCard.class, cardId);  
  
 if (card != null) {  
 if (card.getOwner().equals(currentOwner)) {  
 card.setOwner(newOwner);  
 entityManager.merge(card);  
 } else {  
 throw new IllegalArgumentException("You are not the owner of this card");  
 }  
  
 } else {  
 throw new IllegalArgumentException("Card not found");  
 }  
 }  
  
 public List<SwapCard> getAllCards() {  
 String query = "SELECT c FROM SwapCard c";  
 return entityManager.createQuery(query, SwapCard.class).getResultList();  
 }  
  
 public List<SwapCard> searchCardByName(String nameQuery) {  
 String query = "SELECT c FROM SwapCard c WHERE c.name LIKE :nameQuery";  
 return entityManager.createQuery(query, SwapCard.class)  
 .setParameter("nameQuery", "%" + nameQuery + "%")  
 .getResultList();  
 }  
}

UserService.java

package com.swaps.swap\_cards.service;  
  
import com.swaps.swap\_cards.entity.Achievement;  
import com.swaps.swap\_cards.entity.SwapCard;  
import com.swaps.swap\_cards.entity.User;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.PersistenceContext;  
import jakarta.transaction.Transactional;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class UserService {  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 @Transactional  
 public User createUser(String userName, String email, String password) {  
 User user = new User();  
 user.setUserName(userName);  
 user.setEmail(email);  
 user.setPassword(password);  
 entityManager.persist(user);  
 return user;  
 }  
  
 public List<User> getAllUsers() {  
 String query = "SELECT u FROM User u";  
 return entityManager.createQuery(query, User.class).getResultList();  
 }  
  
 public User getUserById(Integer id) {  
 return entityManager.find(User.class, id);  
 }  
  
 public List<User> searchUserByName(String nameQuery) {  
 String query = "SELECT u FROM User u WHERE u.userName LIKE :nameQuery";  
 return entityManager.createQuery(query, User.class)  
 .setParameter("nameQuery", "%" + nameQuery + "%")  
 .getResultList();  
 }  
  
 public List<SwapCard> getCardsFromUser(Integer userId) {  
 String query = "SELECT c FROM SwapCard c WHERE c.owner.id = :userId";  
 return entityManager.createQuery(query, SwapCard.class)  
 .setParameter("userId", userId)  
 .getResultList();  
 }  
  
 @Transactional  
 public void updateUserPic(Integer userId, String newPicUrl) {  
 User user = entityManager.find(User.class, userId);  
 if (user != null) {  
 user.setLinkToUserPic(newPicUrl);  
 entityManager.merge(user);  
 }  
 }  
 @Transactional  
 public void deleteUserPic(Integer userId) {  
 User user = entityManager.find(User.class, userId);  
 if (user != null) {  
 user.setLinkToUserPic(null);  
 entityManager.merge(user);  
 }  
 }  
  
 public List<Achievement> getAchievements(Integer userId) {  
 String query = "SELECT ua.id.achievement FROM UserAchievement ua WHERE ua.id.user.id = :userId";  
 return entityManager.createQuery(query, Achievement.class)  
 .setParameter("userId", userId)  
 .getResultList();  
 }  
  
}

**Tests.**

DatabaseFillTest.java

package com.swaps.swap\_cards.DatabaseFill;  
  
import com.swaps.swap\_cards.entity.SwapCard;  
import com.swaps.swap\_cards.entity.User;  
import com.swaps.swap\_cards.service.AchievementService;  
import com.swaps.swap\_cards.service.SwapCardService;  
import com.swaps.swap\_cards.service.UserService;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
  
import static org.junit.jupiter.api.Assertions.*assertNotNull*;  
  
@SpringBootTest  
public class DatabaseFillTest {  
 @Autowired  
 private UserService userService;  
  
 @Autowired  
 private AchievementService achievementService;  
  
 @Autowired  
 private SwapCardService swapCardService;  
  
 @BeforeEach  
 public void setup() {  
 }  
  
 @Test  
 public void testFillDatabase() {  
 // Создание пользователей  
 User user1 = userService.createUser("Anya", "anya@example.com", "123");  
 User user2 = userService.createUser("Vova", "vova@example.com", "123");  
 User user3 = userService.createUser("Ilya", "ilya@example.com", "123");  
  
 // Создание карточек  
 SwapCard card1 = swapCardService.createSwapCard("anya", "0", "annoying little girl", user1);  
 SwapCard card2 = swapCardService.createSwapCard("vova", "0", "annoying boy", user2);  
 SwapCard card3 = swapCardService.createSwapCard("ilya", "0", "annoying boy 2", user3);  
 SwapCard card4 = swapCardService.createSwapCard("dog", "0", "very loud dog", user2);  
  
 *assertNotNull*(user1.getId());  
 *assertNotNull*(user2.getId());  
 *assertNotNull*(user3.getId());  
  
 *assertNotNull*(card1.getId());  
 *assertNotNull*(card2.getId());  
 *assertNotNull*(card3.getId());  
 *assertNotNull*(card4.getId());  
 }  
}

ServiceTest.java

package com.swaps.swap\_cards.service;  
  
import com.swaps.swap\_cards.entity.SwapCard;  
import com.swaps.swap\_cards.entity.User;  
import com.swaps.swap\_cards.entity.Achievement;  
import jakarta.transaction.Transactional;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
@SpringBootTest  
public class ServiceTest {  
 @Autowired  
 private UserService userService;  
  
 @Autowired  
 private AchievementService achievementService;  
  
 @Autowired  
 private SwapCardService swapCardService;  
  
 @BeforeEach  
 public void setUp() {  
 }  
  
 //AchievementService  
 @Test  
 @Transactional  
 public void testGrantAchievementToUser\_Success() {  
 *assertDoesNotThrow*(() -> achievementService.grantAchievementToUser(userService.getUserById(2), achievementService.getAchievementById(1)));  
 List<Achievement> achievements = achievementService.getAchievementsForUser(2);  
 *assertTrue*(achievements.contains(achievementService.getAchievementById(1)));  
 }  
  
 @Test  
 public void testGetAchievementsForUser() {  
 List<Achievement> achievements = achievementService.getAchievementsForUser(1);  
 *assertNotNull*(achievements);  
 *assertEquals*(1, achievements.size());  
 }  
  
 // UserService  
 @Test  
 public void testGetAllUsers() {  
 List<User> users = userService.getAllUsers();  
  
 *assertNotNull*(users);  
 *assertEquals*(3, users.size());  
 }  
  
 @Test  
 public void testGetUserById() {  
 User user = userService.getUserById(1);  
  
 *assertNotNull*(user);  
 *assertEquals*(1, user.getId());  
 }  
  
 @Test  
 public void testSearchUserByName() {  
 String nameQuery = "Anya";  
 List<User> users = userService.searchUserByName(nameQuery);  
  
 *assertNotNull*(users);  
 *assertEquals*(1, users.size());  
  
 users.forEach(user -> *assertTrue*(user.getUserName().contains(nameQuery)));  
 }  
  
 @Test  
 @Transactional  
 public void testGetCardsFromUser() {  
 List<SwapCard> cards = userService.getCardsFromUser(1);  
  
 *assertNotNull*(cards);  
 *assertEquals*(1, cards.size());  
 }  
  
 @Test  
 @Transactional  
 public void testUpdateUserPic() {  
 Integer userId = 1;  
 String newPicUrl = "0";  
  
 userService.updateUserPic(userId, newPicUrl);  
  
 User user = userService.getUserById(userId);  
 *assertNotNull*(user);  
 *assertEquals*(newPicUrl, user.getLinkToUserPic());  
 }  
  
 @Test  
 @Transactional  
 public void testDeleteUserPic() {  
 Integer userId = 1;  
  
 userService.deleteUserPic(userId);  
  
 User user = userService.getUserById(userId);  
 *assertNotNull*(user);  
 *assertNull*(user.getLinkToUserPic());  
 }  
  
  
  
 // SwapCardService  
 @Test  
 public void testGetAllCards() {  
 List<SwapCard> cards = swapCardService.getAllCards();  
  
 *assertNotNull*(cards);  
 *assertEquals*(4, cards.size());  
 }  
  
 @Test  
 public void testSearchCardByName() {  
 String nameQuery = "anya";  
 List<SwapCard> cards = swapCardService.searchCardByName(nameQuery);  
  
 *assertNotNull*(cards);  
 *assertEquals*(1, cards.size());  
 }  
  
 @Test  
 @Transactional  
 public void testTransferCard() {  
 User currentOwner = userService.getUserById(1);  
 User newOwner = userService.getUserById(2);  
 SwapCard card = userService.getCardsFromUser(1).get(0);  
  
 *assertEquals*(currentOwner, card.getOwner());  
  
 // Выполняем передачу карточки  
 swapCardService.transferCard(card.getId(), currentOwner, newOwner);  
  
  
 *assertNotNull*(card);  
 *assertEquals*(newOwner, card.getOwner());  
 }  
  
}