Минобрнауки России | РГРТУ | Кафедра ВПМ

Дисциплина «УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ»

**Отчет о практических работах**

Выполнил:

Лапкин Михаил Ильич

Лягушева Мария Алексеевна

студент группы 0413

электронная почта [adraydante@gmail.com](mailto:adraydante@gmail.com)

green.day.forever@mail.ru

Проверил:

Пруцков Александр Викторович

д-р техн. наук, профессор кафедры ВПМ

Рязань 2022

# Описание сетевой информационной системы

## Цель работы

Цель работы – улучшение работы в сфере обмена вещами, изучение новых технологий по созданию сетевой информационной системы "обмен вещами".

## Концептуальная и логическая модели базы данных

На схеме концептуальной модели базы данных (БД) (рисунок 1) отображены основные сущности: пользователь, вещь, заявка, а также вспомогательные сущности: типы пользователей, статус пользователя, статус вещи, статус заявки.



Рисунок 1 - Концептуальная модель базы данных

На рисунке 2 показана логическая модель базы данных. Для всех сущностей добавлены идентификаторы типа int. Для строковых полей атрибутов сущностей был выбран тип varchar с различной размерностью.

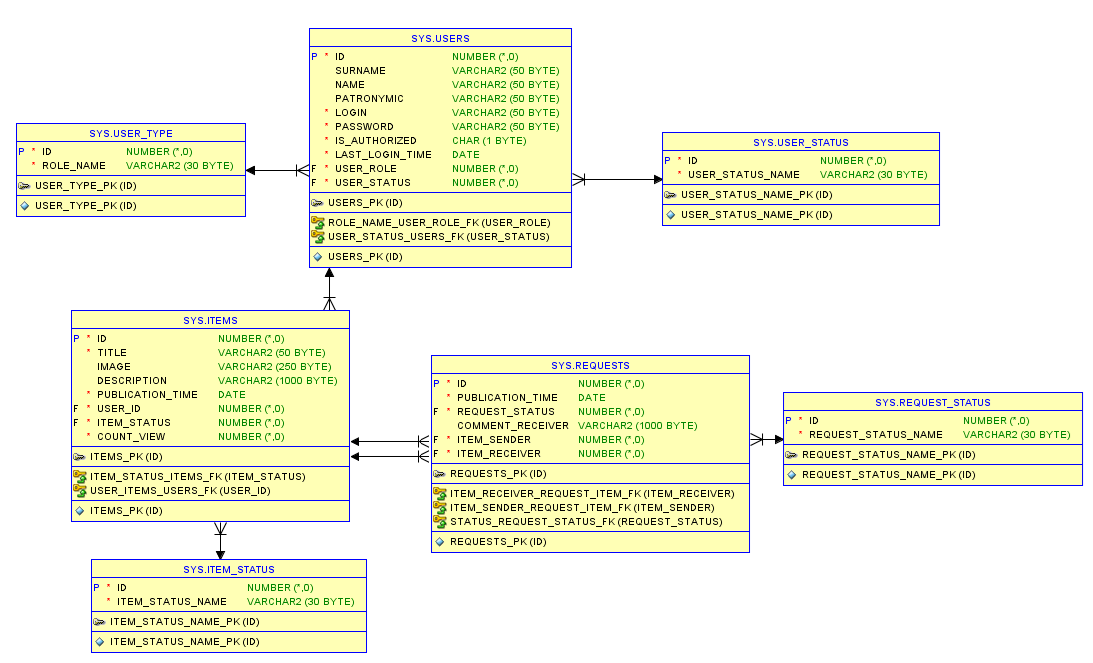


Рисунок 2 - Логическая модель базы данных

## Логика работы сетевой информационной системы, её связь с моделью базы данных

Действия групп пользователей изменяют данные в базе данных (таблица 1).

Таблица 1 – Действия групп пользователей и соответствующие им изменения в базе данных

|  |  |  |
| --- | --- | --- |
| **Действие** | **Группа пользователей** | **Изменения данных в базе данных** |
| Добавление администратора | Администратор | Добавление нового администратора в БД |
| Удаление администратора | Администратор | Удаление администратора из БД |
| Добавление модератора | Администратор | Добавление нового модератора в БД |
| Удаление модератора | Администратор | Удаление модератора из БД |
| Добавление пользователя | Администратор | Добавление нового пользователя в БД |
| Удаление пользователя | Администратор | Удаление пользователя из БД |
| Редактирование данных пользователей | Администратор | Изменение данных о пользователе в БД (логин и пароль) |
| Авторизация | Администратор | Изменение последнего времени авторизации и статуса авторизации (автоматически) |
| Блокирование пользователя | Модератор | Изменение статуса пользователя на "заблокирован" |
| Разблокирование пользователя | Модератор | Изменение статуса пользователя на "разблокирован" |
| Удаление вещи | Модератор | Удаление вещи из БД |
| Удаление заявки | Модератор | Удаление заявки из БД |
| Отмена заявки | Модератор | Изменение статуса заявки на "отменена" |
| Завершение заявки | Модератор | Изменение статуса заявки на "завершена" |
| Авторизация | Модератор | Изменение последнего времени авторизации и статуса авторизации (автоматически) |
| Добавление вещи | Пользователь | Добавление новой вещи в БД |
| Изменение вещи | Пользователь | Изменение информации о вещи в БД |
| Изменение статуса вещи | Пользователь | Изменение статуса вещи на "скрыта" или "открыта" |
| Удаление вещи | Пользователь | Удаление вещи из БД |
| Создание заявки на обмен | Пользователь | Добавление новой заявки в БД |
| Прием заявки на обмен | Пользователь | Изменение статуса заявки на "завершена" |
| Отклонение заявки на обмен | Пользователь | Изменение статуса заявки на "отменена" |
| Авторизация | Пользователь | Изменение последнего времени авторизации и статуса авторизации (автоматически) |

## Отчеты, предоставляемые администратору, модератору и другим группам пользователей

Группам пользователей выдаются различные отчеты (таблица 2).

Таблица 2 – Отчеты для групп пользователей

|  |  |
| --- | --- |
| **Группа пользователей** | **Наименование отчета** |
| Администраторы | Просмотр авторизованных пользователей в данный момент  Просмотр неавторизованных пользователей в данный момент |
| Модераторы | Просмотр заблокированных пользователей  Просмотр всех отмененных заявок |
| Пользователи | Просмотр вещей, выставленных на обмен  Просмотр своих скрытых вещей  Просмотр заявок на его вещи |

# Требуемое программное обеспечение

Для функционирования разработанной сетевой информационной системы требуется программное обеспечение (Таблица 3).

Таблица 3 – Требуемое программное обеспечение

|  |  |  |
| --- | --- | --- |
| **Название** | **Версия** | **Назначение** |
| JDK | 1.8.0 | Разработка на языке Java. Компиляция и запуск написанного кода. |
| Oracle XE Express Edition | [11.2](http://127.0.0.1:8080/apex/f?p=4950:1:535723075287915) | БД для хранения данных |
| SQL Developer | 18.4.0.376 | Создание SQL скрипта для БД |
| Apache Tomcat | 8.5 | Запуск веб-приложения, написанного на Java. Реализация Java-сервлетов и JavaServer Pages (JSP) для создания эффективной среды Java-сервера. |
| Библиотека JDBC | 21.7.0.0 | Установка соединения с базой данных согласно специально описанному URL. |
| Библиотека JSTL | 1.2 | Отделение вёрстки от  программирования на JSP-страницах. |

# Список используемых запросов на языке SQL

## Запросы создания таблиц базы данных

--создание таблицы Типы пользователей

create table user\_type(

id int not null,

role\_name varchar(30) not null,

constraint user\_type\_pk PRIMARY KEY (id)

);

--создание таблицы Статусы пользователей

create table user\_status(

id int not null,

user\_status\_name varchar(30) not null,

constraint user\_status\_name\_pk PRIMARY KEY (id)

);

--создание таблицы Пользователи

create table users(

id int not null,

surname varchar(50),

name varchar(50),

patronymic varchar(50),

login varchar(50) not null,

password varchar(50) not null,

is\_authorized char(1) not null,

last\_login\_time date not null,

user\_role int not null,

user\_status int not null,

constraint users\_pk PRIMARY KEY (id),

constraint role\_name\_user\_role\_fk foreign key (user\_role) references user\_type(id),

constraint user\_status\_users\_fk foreign key (user\_status) references user\_status(id)

);

--создание таблицы Статусы вещей

create table item\_status(

id int not null,

item\_status\_name varchar(30) not null,

constraint item\_status\_name\_pk PRIMARY KEY (id)

);

--создание таблицы Вещи

create table items(

id int not null,

title varchar(50) not null,

image varchar(250),

description varchar(1000) null,

publication\_time date not null,

user\_id int not null,

item\_status int not null,

count\_view int not null,

constraint items\_pk PRIMARY KEY (id),

constraint user\_items\_users\_fk foreign key (user\_id) references users(id),

constraint item\_status\_items\_fk foreign key (item\_status) references item\_status(id)

);

--создание таблицы Статусы заявок

create table request\_status(

id int not null,

request\_status\_name varchar(30) not null,

constraint request\_status\_name\_pk PRIMARY KEY (id)

);

--создание таблицы Заявки

create table requests(

id int not null,

publication\_time date not null,

request\_status int not null,

comment\_receiver varchar(1000) null,

item\_sender int not null,

item\_receiver int not null,

constraint requests\_pk PRIMARY KEY (id),

constraint status\_request\_status\_fk foreign key (request\_status) references request\_status(id),

constraint item\_sender\_request\_item\_fk foreign key (item\_sender) references items(id),

constraint item\_receiver\_request\_item\_fk foreign key (item\_receiver) references items(id)

);

## Запросы выборки данных из таблиц базы данных

--ОТЧЕТЫ

--Администраторы:

--Просмотр авторизованных пользователей в данный момент:

SELECT \* FROM users

WHERE is\_authorized = '1';

--Просмотр неавторизованных пользователей в данный момент

SELECT \* FROM users

WHERE is\_authorized = '0';

--Просмотр пользователей за определенный период

SELECT \* FROM users

WHERE last\_login\_time = '24-04-1997';

--Модераторы:

--Просмотр заявок на обмен, произведенных за определенный период

SELECT \* FROM requests

WHERE publication\_time >= '24-04-1997'

AND publication\_time <= '07-11-2022';

--Просмотр заблокированных пользователей

SELECT \* FROM users

WHERE user\_status = '2';

--Просмотр всех отмененных заявок

SELECT \* FROM requests

WHERE request\_status = '3';

--Пользователи

--Просмотр вещей, выставленных на обмен

SELECT \* FROM items

WHERE item\_status = '1';

--Просмотр своих скрытых вещей

SELECT \* FROM items

WHERE item\_status = '2' AND user\_id = '1';

--Просмотр заявок на его вещи

SELECT items.\*

FROM requests JOIN items on requests.item\_sender = items.id

WHERE user\_id = '1' and requests.request\_status = '1';

## Запросы изменения данных в таблицах базы данных

--создание последовательности id пользователей

CREATE SEQUENCE users\_id

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 20;

--DROP SEQUENCE users\_id;

--создание последовательности id вещей

CREATE SEQUENCE items\_id

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 20;

--DROP SEQUENCE items\_id;

--создание последовательности id запросов

CREATE SEQUENCE requests\_id

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 20;

--DROP SEQUENCE requests\_id;

--ДЕЙСТВИЯ

--Добавления user в БД

INSERT INTO users (id, surname, name, patronymic, login, password, is\_authorized, last\_login\_time, user\_role, user\_status)

VALUES (users\_id.NEXTVAL, 'Ivanоv', 'Ivan', 'Ivanovich', 'ivanov', '1', '1', '24-04-1997', 3, 1);

--Добавление вещи в БД

INSERT INTO items (id, title, image, description, publication\_time, user\_id, item\_status, count\_view)

VALUES (items\_id.NEXTVAL, 'Большая Сумка', 'bag.png', 'Новая большая сумка', '29-09-1999', 1, 1, 2);

--Добавление заявки в БД

INSERT INTO requests (id, publication\_time, request\_status, comment\_receiver, item\_sender, item\_receiver)

VALUES (requests\_id.NEXTVAL, '07-11-2022', 1, 'Обменяю вещь', 1, 4);

--Изменение статуса пользователя

UPDATE users

SET user\_status = '2'

WHERE id = 1;

--Изменение статуса заявки

UPDATE requests

SET request\_status = '2'

WHERE id = 1;

--Изменение статуса вещи

UPDATE items

SET item\_status = '3'

WHERE id = 1;

--Изменение владельца вещи

UPDATE items

SET user\_id = '3'

WHERE id = 1;

--Удаление пользователя из БД

DELETE FROM users

WHERE id = 1;

--Удаление вещи из БД

DELETE FROM items

WHERE id = 1;

--Удаление заявки из БД

DELETE FROM requests

WHERE id = 1;

# Инструкции по работе с сетевой информационной системой

## Инструкция администратора

1. Вход в систему: авторизация - ввести логин и пароль.

2. Просмотр информации о себе: нажать на кнопку "Моя информация".

3. Добавление пользователя: нажать на кнопку "Добавить пользователя", ввести данные и нажать на кнопку "Добавить".

4. Редактирование пользователя: нажать на кнопку "Редактирование и удаление пользователей", выбрать нужного пользователя и нажать на кнопку "Редактировать". Внести необходимые изменения и нажать на кнопку "Сохранить изменения".

5. Удаление пользователя: нажать на кнопку "Редактирование и удаление пользователей", выбрать нужного пользователя и нажать на кнопку "Удалить".

6. Просмотр авторизованных пользователей в данный момент: нажать на кнопку "Просмотр авторизованных пользователей".

## Инструкция модератора

1. Вход в систему: авторизация - ввести логин и пароль.

2. Просмотр информации о себе: нажать на кнопку "Моя информация".

3. Просмотр всех вещей в системе: нажать на кнопку "Вещи".

4. Заблокировать (разблокировать) вещь: нажать на кнопку "Вещи", выбрать необходимую вещь и нажать на кнопку "Заблокировать" ("Разблокировать").

5. Просмотр всех обменов: нажать на кнопку "Обмены".

6. Завершить обмен: нажать на кнопку "Обмены", затем выбрать конкретный обмен и нажать на кнопку "Завершить".

7. Отменить обмен: нажать кнопку "Обмены", затем выбрать конкретный обмен и нажать на кнопку "Отменить".

8. Удалить обмен: нажать кнопку "Обмены", затем выбрать конкретный обмен и нажать на кнопку "Удалить".

9. Просмотр списка пользователей: нажать на кнопку "Список пользователей".

10. Заблокировать пользователя: нажать на кнопку "Список пользователей" и затем нажать на кнопку "Заблокировать".

11. Разблокировать пользователя: нажать на кнопку "Список пользователей" и затем нажать на кнопку "Разблокировать".

## Инструкция пользователя

1. Вход в систему: авторизация - ввести логин и пароль.

2. Просмотр информации о себе: нажать на кнопку "Моя информация".

3. Просмотр своих вещей: нажать на кнопку "Мои вещи".

4. Добавление вещи: нажать на кнопку "Мои вещи", далее нажать на кнопку "Добавить вещь", далее ввести информацию о вещи, нажать на кнопку "Добавить вещь" или "Отменить добавление".

5. Скрыть вещь: нажать на кнопку "Мои вещи" и у конкретной вещи нажать на кнопку "Скрыть".

6. Посмотреть информацию о своей вещи: нажать на кнопку "Мои вещи", далее нажать на кнопку "Подробно" рядом с конкретной вещью.

7. Просмотр своих обменов и заявок на обмен: нажать на кнопку "Обмены и заявки".

8. Произвести обмен: нажать на кнопку "Обмены и заявки", выбрать нужный обмен и затем нажать на кнопку "Обменять".

9. Отказаться от обмена: нажать на кнопку "Обмены и заявки", выбрать нужный обмен и затем нажать на кнопку "Отменить".

10. Выбрать вещь для обмена: нажать на кнопку "Список вещей" и посмотреть предлагаемые вещи.

11. Создать заявку на обмен: нажать на кнопку "Список вещей", выбрать необходимую вещь и нажать на кнопку "Начать обмен", после чего выбрать свою вещь для обмена с помощью кнопки "Обменять эту вещь", либо нажать на кнопку "Отменить обмен".

12. Посмотреть информацию о вещи другого пользователя: нажать на кнопку "Список вещей", далее нажать на кнопку "Подробно" рядом с конкретной вещью.

# Листинг основных классов программы с комментариями Javadoc

## Класс FrontController

package ru.rsreu.exchangethings.controller;  
  
import ru.rsreu.exchangethings.configuration.PathToDirectoryImages;  
import ru.rsreu.exchangethings.controller.controls.Control;  
import ru.rsreu.exchangethings.controller.controls.commands.CommandControlFactory;  
import ru.rsreu.exchangethings.model.service.UserService;  
import ru.rsreu.exchangethings.view.parameters.ActionBean;  
  
import javax.servlet.ServletContext;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.UnsupportedEncodingException;  
/\*\*  
 \* A class that implements design pattern "FrontController".  
 \* It is Application Controller.  
 \*/  
public class FrontController extends HttpServlet {  
 /\*\* Variable that instantiates an object factory  
 \*/  
 private final CommandControlFactory client = new CommandControlFactory();  
 /\*\*  
 \* Variable that instantiates an object userService  
 \*/  
 private UserService userService = UserService.instance;  
 /\*\*  
 \* Method that defines the command  
 \*/  
 private void processRequest(HttpServletRequest request, HttpServletResponse response) {  
 Control command = client.defineControl(request);  
 command.control(request, response);  
 }  
  
 /\*\*  
 \* Method that initializes data and prepares for processing requests  
 \*/  
 public void init() {  
 ServletContext context = this.getServletContext();  
 context.setAttribute("actions", new ActionBean());  
 PathToDirectoryImages.setPath(context.getRealPath("/"));  
 userService.updateAllUserAuth();  
 }  
 /\*\* Method that handles GET requests  
 \* @param request - Lower bound of the range of generated random numbers  
 \* @param response - Upper bound of the range of generated random numbers  
 \* @throws UnsupportedEncodingException  
 \*/  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws UnsupportedEncodingException {  
 processRequest(request, response);  
 }  
 /\*\* Method that handles POST requests  
 \* @param request - Lower bound of the range of generated random numbers  
 \* @param response - Upper bound of the range of generated random numbers  
 \* @throws UnsupportedEncodingException  
 \*/  
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws UnsupportedEncodingException {  
 processRequest(request, response);  
 }  
  
 /\*\*  
 \* Method that releases the resources occupied by the servlet.  
 \*/  
 public void destroy() {  
 }  
}

## Класс UserEntity

package ru.rsreu.exchangethings.model.datalayer.entity;  
  
import java.sql.Date;  
  
/\*\*  
 \* Data-class describing the User Entity of the subject area  
 \*/  
public class UserEntity {  
 /\*\*  
 \* field-properties  
 \*/  
 private int userID;  
 private String surname;  
 private String name;  
 private String patronymic;  
 private String login;  
 private String password;  
 private String isAuthorized;  
 private Date lastLoginTime;  
 private int userRole;  
 private int userStatus;  
 /\*\*  
 \* Constructor - determines a new entity  
 \*/  
 public UserEntity(int id, String surname, String name, String patronymic, String login, String password,  
 String isAuthorized, Date lastLoginTime, int userRole, int userStatus) {  
 this.userID = id;  
 this.surname = surname;  
 this.name = name;  
 this.patronymic = patronymic;  
 this.login = login;  
 this.password = password;  
 this.isAuthorized = isAuthorized;  
 this.lastLoginTime = lastLoginTime;  
 this.userRole = userRole;  
 this.userStatus = userStatus;  
 }  
  
 /\*\*  
 \*Getters and Setters  
 \*/  
 public int getUserID() {  
 return userID;  
 }  
  
 public void setUserID(int userID) {  
 this.userID = userID;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getIsAuthorized() {  
 return isAuthorized;  
 }  
  
 public void setIsAuthorized(String isAuthorized) {  
 this.isAuthorized = isAuthorized;  
 }  
  
 public Date getLastLoginTime() {  
 return lastLoginTime;  
 }  
  
 public void setLastLoginTime(Date lastLoginTime) {  
 this.lastLoginTime = lastLoginTime;  
 }  
  
 public int getUserRole() {  
 return userRole;  
 }  
  
 public void setUserRole(int userRole) {  
 this.userRole = userRole;  
 }  
  
 public int getUserStatus() {  
 return userStatus;  
 }  
  
 public void setUserStatus(int userStatus) {  
 this.userStatus = userStatus;  
 }  
  
 @Override  
 public String toString() {  
 return "UserEntity{" +  
 "userID=" + userID +  
 ", surname='" + surname + '\'' +  
 ", name='" + name + '\'' +  
 ", patronymic='" + patronymic + '\'' +  
 ", login='" + login + '\'' +  
 ", password='" + password + '\'' +  
 ", isAuthorized='" + isAuthorized + '\'' +  
 ", lastLoginTime=" + lastLoginTime +  
 ", userRole=" + userRole +  
 ", userStatus=" + userStatus +  
 '}';  
 }  
}

## Класс ItemEntity

package ru.rsreu.exchangethings.model.datalayer.entity;  
  
import java.sql.Date;  
/\*\*  
 \* Data-class describing the Item Entity of the subject area  
 \*/  
  
public class ItemEntity {  
 /\*\*  
 \* field-properties  
 \*/  
 private int itemsID;  
 private String title;  
 private String image;  
 private String description;  
 private Date publicationTime;  
 private int userID;  
 private int itemStatus;  
 private int countView;  
 /\*\*  
 \* Constructor - determines a new entity  
 \*/  
 public ItemEntity(int itemsID, String title, String image, String description, Date publicationTime,  
 int userID, int itemStatus, int countView) {  
 this.itemsID = itemsID;  
 this.title = title;  
 this.image = image;  
 this.description = description;  
 this.publicationTime = publicationTime;  
 this.userID = userID;  
 this.itemStatus = itemStatus;  
 this.countView = countView;  
 }  
 /\*\*  
 \*Getters and Setters  
 \*/  
 public int getItemsID() {  
 return itemsID;  
 }  
  
 public void setItemsID(int itemsID) {  
 this.itemsID = itemsID;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getImage() {  
 return image;  
 }  
  
 public void setImage(String image) {  
 this.image = image;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public Date getPublicationTime() {  
 return publicationTime;  
 }  
  
 public void setPublicationTime(Date publicationTime) {  
 this.publicationTime = publicationTime;  
 }  
  
 public int getUserID() {  
 return userID;  
 }  
  
 public void setUserID(int userID) {  
 this.userID = userID;  
 }  
  
 public int getItemStatus() {  
 return itemStatus;  
 }  
  
 public void setItemStatus(int itemStatus) {  
 this.itemStatus = itemStatus;  
 }  
  
 public int getCountView() {  
 return countView;  
 }  
  
 public void setCountView(int countView) {  
 this.countView = countView;  
 }  
  
 @Override  
 public String toString() {  
 return "ItemEntity{" +  
 "itemsID=" + itemsID +  
 ", title='" + title + '\'' +  
 ", image='" + image + '\'' +  
 ", description='" + description + '\'' +  
 ", publicationTime=" + publicationTime +  
 ", userID=" + userID +  
 ", itemStatus=" + itemStatus +  
 ", countView=" + countView +  
 '}';  
 }  
}

## Класс RequestEntity

package ru.rsreu.exchangethings.model.datalayer.entity;  
  
import java.sql.Date;  
/\*\*  
 \* Data-class describing the Request Entity of the subject area  
 \*/  
public class RequestEntity {  
 /\*\*  
 \* field-properties  
 \*/  
 private int RequestID;  
 private Date publicationTime;  
 private int requestStatus;  
 private String commentReceiver;  
 private int itemSenderID;  
 private int itemReceiverID;  
 /\*\*  
 \* Constructor - determines a new entity  
 \*/  
 public RequestEntity(int requestID, Date publicationTime, int requestStatus, String commentReceiver, int itemSenderID, int itemReceiverID) {  
 RequestID = requestID;  
 this.publicationTime = publicationTime;  
 this.requestStatus = requestStatus;  
 this.commentReceiver = commentReceiver;  
 this.itemSenderID = itemSenderID;  
 this.itemReceiverID = itemReceiverID;  
 }  
 /\*\*  
 \*Getters and Setters  
 \*/  
 public int getRequestID() {  
 return RequestID;  
 }  
  
 public void setRequestID(int requestID) {  
 RequestID = requestID;  
 }  
  
 public Date getPublicationTime() {  
 return publicationTime;  
 }  
  
 public void setPublicationTime(Date publicationTime) {  
 this.publicationTime = publicationTime;  
 }  
  
 public int getRequestStatus() {  
 return requestStatus;  
 }  
  
 public void setRequestStatus(int requestStatus) {  
 this.requestStatus = requestStatus;  
 }  
  
 public String getCommentReceiver() {  
 return commentReceiver;  
 }  
  
 public void setCommentReceiver(String commentReceiver) {  
 this.commentReceiver = commentReceiver;  
 }  
  
 public int getItemSenderID() {  
 return itemSenderID;  
 }  
  
 public void setItemSenderID(int itemSenderID) {  
 this.itemSenderID = itemSenderID;  
 }  
  
 public int getItemReceiverID() {  
 return itemReceiverID;  
 }  
  
 public void setItemReceiverID(int itemReceiverID) {  
 this.itemReceiverID = itemReceiverID;  
 }  
  
 @Override  
 public String toString() {  
 return "RequestEntity{" +  
 "RequestID=" + RequestID +  
 ", publicationTime=" + publicationTime +  
 ", requestStatus=" + requestStatus +  
 ", commentReceiver='" + commentReceiver + '\'' +  
 ", itemSenderID=" + itemSenderID +  
 ", itemReceiverID=" + itemReceiverID +  
 '}';  
 }  
}

## Интерфейс UserDAO

package ru.rsreu.exchangethings.model.datalayer;  
  
import ru.rsreu.exchangethings.model.UserRoleEnum;  
import ru.rsreu.exchangethings.model.datalayer.entity.UserEntity;  
  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.util.List;  
  
/\*\*  
 \* DAO user interface  
 \*/  
public interface UserDAO {  
 /\*\*  
 \* Method that allows view currently authorized/unauthorized users:  
 \* @param authorizationStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 List<UserEntity> getUsersByAuthorizationStatus(String authorizationStatus) throws SQLException;

package ru.rsreu.exchangethings.model.datalayer;  
  
import ru.rsreu.exchangethings.model.UserRoleEnum;  
import ru.rsreu.exchangethings.model.datalayer.entity.UserEntity;  
  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.util.List;  
  
/\*\*  
 \* DAO user interface  
 \*/  
public interface UserDAO {  
 /\*\*  
 \* Method that allows view currently authorized/unauthorized users:  
 \* @param authorizationStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 List<UserEntity> getUsersByAuthorizationStatus(String authorizationStatus) throws SQLException;  
  
 /\*\*  
 \* Method that allows view users for a specific period  
 \* @param lastLoginTime  
 \* @return  
 \* @throws SQLException  
 \* @throws ParseException  
 \*/  
 List<UserEntity> getUsersByLastLoginTime(String lastLoginTime) throws SQLException, ParseException;  
 //Просмотр заблокированных пользователей  
 List<UserEntity> getBlockedUsers(int blockedStatus) throws SQLException;  
  
 /\*\*  
 \* View users by role  
 \* @param role  
 \* @return  
 \* @throws SQLException  
 \*/  
 List<UserEntity> getAllByRole(int role) throws SQLException;  
  
 /\*\*  
 \* Adding a user  
 \* @throws SQLException  
 \* @throws ParseException  
 \*/  
 void insertUser(String surname, String name, String patronymic,  
 String login, String password, String isAuthorized,  
 String lastLoginTime, int userRole, int userStatus) throws SQLException, ParseException;  
  
 /\*\*  
 \*Update user by status  
 \* @param userStatus  
 \* @param userId  
 \* @throws SQLException  
 \*/  
 void updateUserStatus(int userStatus, int userId) throws SQLException;  
  
 /\*\*  
 \* Deleting a user  
 \* @param userId  
 \* @throws SQLException  
 \*/  
 void deleteUser(int userId) throws SQLException;  
  
 /\*\*  
 \* User login  
 \* @param login  
 \* @param password  
 \* @return  
 \* @throws SQLException  
 \*/  
 UserEntity loginUser(String login, String password) throws SQLException;  
  
 /\*\*  
 \* Get User By Id  
 \* @param id  
 \* @return  
 \* @throws SQLException  
 \*/  
 UserEntity getUserById(int id) throws SQLException;  
  
 /\*\*  
 \* Update All User Auth  
 \* @throws SQLException  
 \*/  
 void updateAllUserAuth() throws SQLException;  
  
 /\*\*  
 \* Update User Auth  
 \* @param idUser  
 \* @param auth  
 \* @throws SQLException  
 \*/  
 void updateUserAuth(int idUser, Boolean auth) throws SQLException;  
  
 /\*\*  
 \* Update User Info  
 \* @param userEntity  
 \* @throws SQLException  
 \*/  
 void updateUserInfo(UserEntity userEntity) throws SQLException;  
}

## Класс ItemDAO

package ru.rsreu.exchangethings.model.datalayer;  
  
import ru.rsreu.exchangethings.model.datalayer.entity.ItemEntity;  
  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.util.List;  
/\*\*  
 \* DAO Item interface  
 \*/  
public interface ItemDAO {  
 /\*\*  
 \* View items for exchange  
 \* @param userId  
 \* @param itemStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 List<ItemEntity> getItemsForExchange(int userId, int itemStatus) throws SQLException;

/\*\*  
 \* View user items  
 \* @param userId  
 \* @return  
 \* @throws SQLException  
 \*/  
List<ItemEntity> getItemsByUser(int userId) throws SQLException;  
  
/\*\*  
 \* Get All  
 \* @return  
 \* @throws SQLException  
 \*/  
List<ItemEntity> getAll() throws SQLException;  
  
/\*\*  
 \* Get Item By Id  
 \* @param id  
 \* @return  
 \* @throws SQLException  
 \*/  
ItemEntity getItemById(int id) throws SQLException;  
  
/\*\*  
 \* Increment Count View  
 \* @param id  
 \* @throws SQLException  
 \*/  
void incrementCountView(int id) throws SQLException;

/\*\*  
 \* InsertItem  
 \* @throws SQLException  
 \* @throws ParseException  
 \*/  
 void insertItem(String title, String image, String description, String publicationTime, int userId,  
 int itemStatus) throws SQLException, ParseException;  
  
 /\*\*  
 \* Update Item Status  
 \* @param itemStatus  
 \* @param itemId  
 \* @throws SQLException  
 \*/  
 void updateItemStatus(int itemStatus, int itemId) throws SQLException;  
  
 /\*\*  
 \* Delete Item  
 \* @param userId  
 \* @throws SQLException  
 \*/  
 void deleteItem(int userId) throws SQLException;  
  
 /\*\*  
 \* Update Item User Owner  
 \* @param itemId  
 \* @param userId  
 \* @throws SQLException  
 \*/  
 void updateItemUserOwner(int itemId, int userId) throws SQLException;  
}

## Класс RequestDAO

package ru.rsreu.exchangethings.model.datalayer;  
  
import ru.rsreu.exchangethings.model.datalayer.entity.RequestEntity;  
  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.util.List;  
/\*\*  
 \* DAO Request interface  
 \*/  
public interface RequestDAO {  
 /\*\*  
 \* View all canceled tickets  
 \* @param requestStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 List<RequestEntity> getRequestsByStatus(int requestStatus) throws SQLException;

/\*\*  
 \* Insert Request  
 \* @throws SQLException  
 \* @throws ParseException  
 \*/  
 void insertRequest(String publicationTime, int requestStatus, String commentReceiver,  
 int itemSender, int itemReceiver) throws SQLException, ParseException;  
  
 /\*\*  
 \* Update Request Status  
 \* @param requestStatus  
 \* @param requestId  
 \* @throws SQLException  
 \*/  
 void updateRequestStatus(int requestStatus, int requestId) throws SQLException;  
  
 /\*\*  
 \* delete Request   
 \* @param requestId  
 \* @throws SQLException  
 \*/  
 void deleteRequest(int requestId) throws SQLException;  
  
 /\*\*  
 \* get Requests By User Id And Status  
 \* @param userId  
 \* @param requestStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 public List<RequestEntity> getRequestsByUserIdAndStatus(int userId, int requestStatus) throws SQLException;  
  
 /\*\*  
 \* get Request By Id  
 \* @param requestStatus  
 \* @return  
 \* @throws SQLException  
 \*/  
 public RequestEntity getRequestById(int requestStatus) throws SQLException;  
  
 /\*\*  
 \* delete Request By Item Id  
 \* @param itemId  
 \* @throws SQLException  
 \*/  
 void deleteRequestByItemId(int itemId) throws SQLException;  
  
 /\*\*  
 \* Update Status By Item  
 \* @param itemId  
 \* @param status  
 \* @throws SQLException  
 \*/  
 void updateStatusByItem(int itemId, int status) throws SQLException;  
}

# Библиографический список

1. Пруцков А. В. Сборник документов для учебных занятий 2020 года / Рязан. гос. радиотехн. ун-т им. В. Ф. Уткина. – Рязань, 2020. – 36 с. – № 5500.
2. Пруцков А. В. Сборник документов для учебных занятий 2022 года / Рязан. гос. радиотехн. ун-т им. В. Ф. Уткина. – Рязань, 2022. – 24 с. – № 7077.
3. Пруцков А. В. Программирование на языке Java. Введение в курс с примерами и практическими заданиями: учебник. — М.: КУРС, 2018. — 208 с.
4. Блинов И.Н., Романчик В.С. Java. Методы программирования: уч.-мет. пособие. — Минск: издательство «Четыре четверти», 2013. — 896 с.