Минобрнауки России | РГРТУ | Кафедра ВПМ

Дисциплина «УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ»

ОТЧЕТ О ПРАКТИЧЕСКИХ РАБОТАХ

Выполнил:
Вербицкая Ирина Сергеевна, Степанова Татьяна Евгеньевна
студент группы 143
электронная почта oora.frt@gmail.com, te.stepanowa@gmail.com

Проверил:
Пруцков Александр Викторович
д-р техн. наук, профессор кафедры ВПМ

Рязань 2023

1. Описание сетевой информационной системы

1.1. Цель работы

Цель работы – разработать и реализовать сетевую информационную систему "Интернет-аукцион", представляющую из себя удобную платформу для проведения аукционов в Интернете. Система позволит продавцам и покупателям взаимодействовать друг с другом: легко выставлять товары на аукцион, делать ставки и успешно заключать сделки. На платформе пользователю будет удобно совершать поиск по товарам, отслеживать процесс аукциона над собственными товарами.

1.2. Концептуальная и логическая модели базы данных

Данная предметная область разбивается на сущности, приведенные ниже.

Данные пользователей - эта сущность содержит информацию о всех пользователях системы. Каждый пользователь обладает набором атрибутов:

- Логин - уникальное имя пользователя, с которым он входит в систему и под которым взаимодействует с ней.
- Пароль - пароль пользователя, с помощью которого пользователь авторизуется.
- Группа - данный атрибут определяет вид пользователя; см. описание сущности "Виды пользователей".
- Статус сети - логическое значение, показывающее, авторизован ли пользователь в системе.
- Статус блокировки - логическое значение, показывающее, заблокирован ли пользователь.

Данные товаров - сущность, которая содержит информацию о товарах (лотах), выставляемых на аукцион, и которая обладает следующим набором атрибутов:

- Код - идентификатор товара.
- Название - название товара, определяемое при его добавлении в систему.
- Описание - краткое описание товара.

- Категория - атрибут, определяющий, к какой категории относится товар; см. описание сущности "Категории товаров".
- Начальная цена - первоначальная стоимость товара, определенная владельцем (продавцом).
- Цена - текущая, либо конечная (в зависимости от статуса) стоимость товара.
- Шаг цены - значение, на которое увеличивается цена товара в момент заключения сделки.
- Счетчик ставок - целочисленное значение, увеличивающееся каждый раз, когда с товаром заключают сделку.
- Дата создания - дата, когда пользователь выставил товар на аукцион.
- Дата продажи - атрибут, значение которого рассчитывается в зависимости от интервала продажи; это день, когда товар станет недоступен для покупки.
- Интервал продажи - период, в течение которого товар доступен для заключения сделок.
- Статус - значение, определяющее состояние товара; см. описание сущности "Статусы товаров".
- Продавец - логин пользователя, который выставил товар на аукцион.

Журнал сделок - сущность, содержащая в себе информацию обо всех сделках, заключаемых с товарами. Под сделкой подразумевается повышение пользователем цены товара на аукционе, это событие и регистрируется в журнале сделок. Сущность обладает рядом атрибутов:

- Код - идентификатор сделки.
- Дата и время - данный атрибут хранит информацию о том, в какой момент времени была совершена сделка.
- Цена - стоимость, которая была актуальна для товара в момент заключения сделки.
- Товар - код товара, задействованного в сделке.
- Покупатель - логин пользователя, заключившего сделку.

Виды пользователей - сущность, содержащая информацию о видах пользователей системы. Включает атрибут Вид, определяющий, какими функциями и правами обладает пользователь в системе. Всего пользователи будут делиться на три вида: администратор, модератор и обычный пользователь.

Категории товаров - сущность, содержащая информацию о категориях товаров. Включает атрибут "Категория". Всего будет определено 8 категорий: продукты, искусство, бытовая химия, электроника, мебель, красота и здоровье, музыка, прочее.

Статусы товаров - сущность, содержащая информацию о состояниях товаров. Включает атрибут "Статус". Всего для товаров будут определены три состояния: "В продаже", "Продан" и "Снят с продажи". Состоянием "В продаже" обладают товары, которые доступны для заключения сделок на аукционе. Состояние "Продан" означает, что с товаром были заключены сделки и дата продажи уже наступила. Состоянием "Снят с продажи" характеризуются товары, которые не были проданы за период продажи по причине отсутствия сделок с ними или по причине удаления товара с аукциона.

Рисунок 1 - Логическая модель базы данных

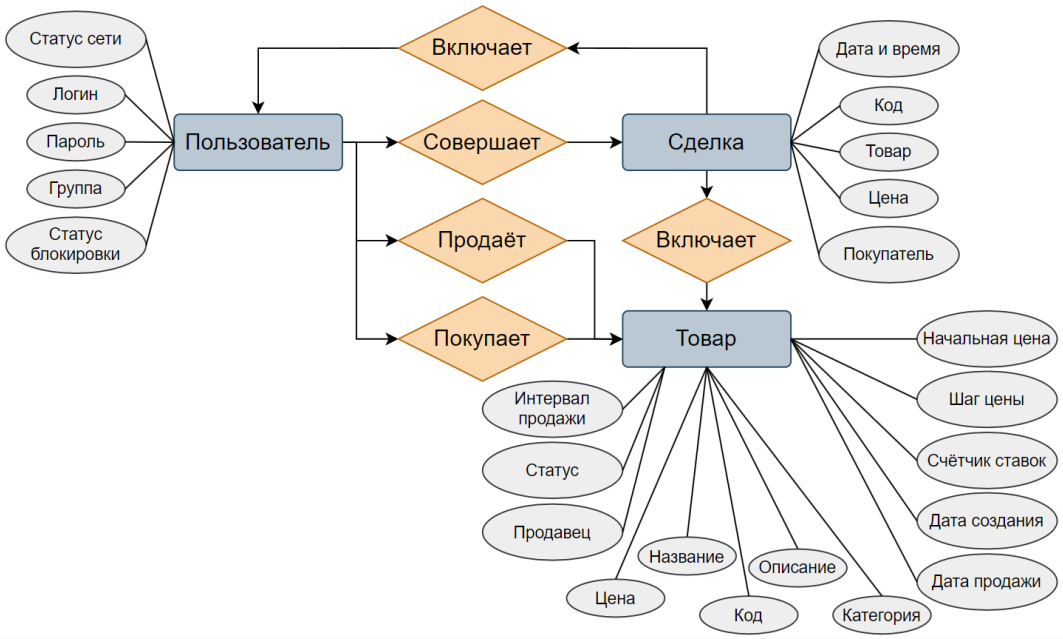

Рисунок 2 - Концептуальная модель базы данных

1.3. 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
Авторизация, деавторизация	Все	Поле "Статус сети" таблицы "Данные пользователей"
Добавление пользователя	Администратор	Новая запись в таблице "Данные пользователей"
Удаление пользователя	Администратор	Удаление записи из таблицы "Данные пользователей".
		Задание значения NULL поля "Продавец" для товаров этого пользователя со статусом "Продан".
		Удаление товаров этого пользователя со статусом "В продаже" или "Снят с продажи".
		Удаление записей о сделках этого пользователя из таблицы "Журнал сделок"

Редактирование данных пользователя	Администратор	Поля таблицы "Данные пользователей", кроме логина и пароля
Блокировка пользователей	Модератор	Поле "Статус блокировки" таблицы "Данные пользователей"
		Удаление записей о сделках этого пользователя из таблицы "Журнал сделок" для товаров, у которых статус "В продаже"
		Смена поля "Статус" товаров этого пользователя с "В продаже" на "Снят с продажи"
Разблокировка пользователей	Модератор	Поле "Статус блокировки" таблицы "Данные пользователей"
Задание срока продажи всех лотов	Модератор	Изменение значения переменной "Интервал продажи по умолчанию" и поля "Интервал продажи" всех записей таблицы "Данные товаров", а также связанного с ним поля "Дата продажи"
Задание срока продажи отдельных лотов	Модератор	Поле "Интервал продажи" отдельных записей таблицы "Данные товаров", а также связанное с ним поле "Дата продажи"
Задание шага повышения цены лота	Модератор	Изменение значения переменной "Шаг цены по умолчанию" и поля "Шаг цены" всех записей таблицы "Данные товаров". Или изменение поля "Шаг цены" отдельных записей таблицы "Данные товаров"
Снятие лота с продажи	Модератор, пользователь	Значение поля "Статус" таблицы "Данные товаров" изменяется с текущего на "Снят с продажи"
Выставление лота на	Пользователь	Новая запись в таблице

продажу и установление начальной цены		"Данные товаров"
Повышение цены лота на определенный шаг	Пользователь	Новая запись в таблице "Журнал сделок", изменение поля "Счётчик ставок" товара в таблице "Данные товаров", а также поля "Цена"
Редактирование данных лота	Пользователь	Следующие поля таблицы "Данные товаров": "Описание"

1.4. 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**
Администратор	Все пользователи
Администратор	Все авторизованные пользователи
Администратор	Все администраторы
Администратор	Все модераторы
Администратор, модератор, обычный пользователь	Все обычные пользователи
Модератор, обычный пользователь	Все продающиеся товары (по алфавиту, по дате продажи)
Модератор, обычный пользователь	Продающиеся товары по категориям
Модератор, обычный пользователь	"Горячие" товары (товары, у которых значение счётчика ставок наибольшее)
Модератор, обычный пользователь	Все продающиеся товары выбранного пользователя
Обычный пользователь	Свои продающиеся товары
Обычный пользователь	Свои проданные товары
Обычный пользователь	Свои снятые с продажи товары

Обычный пользователь	Свои завешенные торги: выигранные (купленные товары) и проигранные
Обычный пользователь	Свои текущие торги (проигрывающие и выигрывающие)

2. Требуемое программное обеспечение

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

Таблица 3 – Требуемое программное обеспечение

Название	Версия	Назначение
JDK	1.8.0.144	Комплект разработчика приложений на языке Java
Oracle XE Express Edition	21c	Реляционная система управления базами данных
SQL Developer	19.2.1	Интегрированная среда разработки на языке SQL; даёт возможность администрирования БД
Apache Tomcat	10.1.13	Контейнер сервлетов; самостоятельный веб-сервер
Библиотека JDBC		Взаимодействие Java-приложения с реляционной БД
Библиотека JSTL		Реализация динамического содержимого веб-страницы на стороне сервера

3. Список используемых запросов на языке SQL

3.1. Запросы создания таблиц базы данных

```
/* Создание и заполнение таблицы с типами пользователей */

CREATE TABLE UserType(
    type varchar2(20 char) not null
        check (type in ('Обычный пользователь','Администратор','Модератор')));
```

```
ALTER TABLE UserType
ADD CONSTRAINT PK_type primary key (type);

INSERT INTO UserType VALUES ('Обычный пользователь');
INSERT INTO UserType VALUES ('Администратор');
INSERT INTO UserType VALUES ('Модератор');


/* Создание и заполнение таблицы с данными пользователей */

CREATE TABLE UserData(
	login varchar2(15 char) not null,
	parole varchar2(15 char) default '1' not null,
	type varchar2(20 char) default 'Обычный пользователь' not null,
	online_status varchar2(1 char) default '0' not null
	check (online_status in ('0', '1')),
	blocking_status varchar2(1 char) default '0' not null
	check (blocking_status in ('0', '1')));

ALTER TABLE UserData
ADD CONSTRAINT PK_login primary key (login);

ALTER TABLE UserData
ADD CONSTRAINT FK_User_UserType foreign key (type)
	references UserType(type);

INSERT INTO UserData(login) VALUES ('red_pepper');
INSERT INTO UserData(login, blocking_status) VALUES ('blue_sky', 1);
INSERT INTO UserData(login) VALUES ('green_glass');
INSERT INTO UserData(login) VALUES ('pink_cloud');
INSERT INTO UserData(login) VALUES ('white_rabbit');
INSERT INTO UserData(login) VALUES ('black_swan');
INSERT INTO UserData(login, type) VALUES ('Blue_moon', 'Модератор');
INSERT INTO UserData(login, type) VALUES ('Blue_sea', 'Модератор');
INSERT INTO UserData(login, type) VALUES ('DARK_HORSE', 'Администратор');
INSERT INTO UserData(login, type) VALUES ('DARK_GLOOM', 'Администратор');


/* Создание и заполнение таблицы с категориями товаров */

CREATE TABLE ProductCategory(
	category varchar2(20 char) not null
		check (category in ('Продукты','Искусство','Бытоваяхимия','Электроника',
```

```
                'Мебель','Красота и здоровье','Музыка','Прочее')));

ALTER TABLE ProductCategory
ADD CONSTRAINT PK_category primary key (category);

INSERT INTO ProductCategory VALUES ('Продукты');
INSERT INTO ProductCategory VALUES ('Искусство');
INSERT INTO ProductCategory VALUES ('Бытоваяхимия');
INSERT INTO ProductCategory VALUES ('Электроника');
INSERT INTO ProductCategory VALUES ('Мебель');
INSERT INTO ProductCategory VALUES ('Красота и здоровье');
INSERT INTO ProductCategory VALUES ('Музыка');
INSERT INTO ProductCategory VALUES ('Прочее');


/* Создание и заполнение таблицы со статусами товаров */

CREATE TABLE ProductStatus(
    status varchar2(14 char) not null
        check (status in ('В продаже','Продан','Снят с продажи')));

ALTER TABLE ProductStatus
ADD CONSTRAINT PK_status primary key (status);

INSERT INTO ProductStatus VALUES ('В продаже');
INSERT INTO ProductStatus VALUES ('Продан');
INSERT INTO ProductStatus VALUES ('Снят с продажи');


/* Создание и заполнение таблицы с данными товаров */

CREATE TABLE ProductData(
    id int generated always as identity(start with 1),
    name varchar2(40 char) not null,
    description varchar2(200 char),
    category varchar2(20 char) default 'Прочее' not null,
    start_price int not null,
    price int not null,
    price_step int not null,
    bet_count int default 0,
    creating_date date not null,
    selling_date date not null,
    selling_interval int not null,
```

```
    status varchar2(14 char) default 'В продаже' not null,
    seller varchar2(15 char));

ALTER TABLE ProductData
ADD CONSTRAINT PK_product_id primary key (id);

ALTER TABLE ProductData
ADD CONSTRAINT FK_Product_ProductCategory foreign key (category)
    references ProductCategory(category);

ALTER TABLE ProductData
ADD CONSTRAINT FK_Product_ProductStatus foreign key (status)
    references ProductStatus(status);

ALTER TABLE ProductData
ADD CONSTRAINT FK_Product_User foreign key (seller)
    references UserData(login);

INSERT INTO ProductData(name, description, category, start_price, price, price_step,
  bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
  ('Гусли', 'Эти чудесные гусли мы нашли на чердаке', 'Музыка', 250,600,50,7,
  TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2024.01.30', 'yyyy.mm.dd'), 124, 'В продаже',
'green_glass');

INSERT INTO ProductData(name, description, category, start_price, price, price_step,
  bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
  ('Портрет Людмилы Петровны', 'Портрет неизвестной учительницы, найденный на барахолке',
'Искусство', 1000,1500,100,5,
  TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2024.01.30', 'yyyy.mm.dd'), 124, 'В продаже',
'red_pepper');

INSERT INTO ProductData(name, description, category, start_price, price, price_step,
  bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
  ('Портрет Виталия Витальевича', 'Портрет, спасённый от стаи собак в Анапе', 'Искусство',
1000,1500,100,5,
  TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2023.10.10', 'yyyy.mm.dd'), 12, 'Продан',
'red_pepper');

INSERT INTO ProductData(name, description, category, start_price, price, price_step,
  bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
  ('Портрет Александра Александровича', 'Портрет, купленный у городской сумашедшей в
Рыбинске', 'Искусство', 1000,1500,100,5,
  TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2023.10.10', 'yyyy.mm.dd'), 12, 'Продан',
'red_pepper');
```

```
INSERT INTO ProductData(name, description, category, start_price, price, price_step,
   bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
   ('Музыкальная шкатулка со Шреком', 'Шкатулка играет песню "All star" группы "Smash Mouth"', 'Музыка', 250,250,50,0,
   TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2023.10.10', 'yyyy.mm.dd'), 12, 'Снят с продажи', 'green_glass');

INSERT INTO ProductData(name, description, category, start_price, price, price_step,
   bet_count, creating_date, selling_date, selling_interval, status, seller) VALUES
   ('Консервы 20-го века', 'Приятного аппетита!', 'Продукты', 50,100,50,1,
   TO_DATE('2023.09.28', 'yyyy.mm.dd'), TO_DATE('2024.01.30', 'yyyy.mm.dd'), 124, 'Снят с продажи', 'blue_sky');


/* Создание и заполнение таблицы с данными о сделках */

CREATE TABLE DealJournal(
      id int generated always as identity(start with 1),
      datetime date not null,
      price int not null,
      product int not null,
      buyer varchar(15));

ALTER TABLE DealJournal
ADD CONSTRAINT PK_deal_id primary key (id);

ALTER TABLE DealJournal
ADD CONSTRAINT FK_Deal_Product foreign key (product)
      references ProductData(id);

ALTER TABLE DealJournal
ADD CONSTRAINT FK_Deal_User foreign key (buyer)
      references UserData(login);

INSERT INTO DealJournal(datetime, price, product, buyer) VALUES
(TO_DATE('2023.09.28 14:00', 'yyyy.mm.dd hh24:mi'), 450, 1, 'red_pepper');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.28 15:30', 'yyyy.mm.dd hh24:mi'), 500, 1, 'white_rabbit');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.29 12:00', 'yyyy.mm.dd hh24:mi'), 550, 1, 'black_swan');

INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
```

```
(TO_DATE('2023.09.30 12:00', 'yyyy.mm.dd hh24:mi'), 1300, 2, 'black_swan');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.30 12:30', 'yyyy.mm.dd hh24:mi'), 1400, 2, 'white_rabbit');

INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.28 12:00', 'yyyy.mm.dd hh24:mi'), 1200, 3, 'pink_cloud');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.29 12:00', 'yyyy.mm.dd hh24:mi'), 1300, 3, 'white_rabbit');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.30 12:00', 'yyyy.mm.dd hh24:mi'), 1400, 3, 'black_swan');

INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.28 14:00', 'yyyy.mm.dd hh24:mi'), 1200, 4, 'pink_cloud');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.29 14:30', 'yyyy.mm.dd hh24:mi'), 1300, 4, 'black_swan');
INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(TO_DATE('2023.09.30 15:00', 'yyyy.mm.dd hh24:mi'), 1400, 4, 'white_rabbit');
```

3.2. Запросы выборки данных из таблиц базы данных

```
/* Все пользователи*/
SELECT *
FROM UserData;

/* Все авторизованные пользователи */
SELECT *
FROM UserData
WHERE online_status = '1';

/* Все обычные пользователи */
SELECT *
FROM UserData
WHERE type = 'Обычный пользователь';

/* Все администраторы */
SELECT *
FROM UserData
WHERE type = 'Администратор';

/* Все модераторы */
SELECT *
FROM UserData
WHERE type = 'Модератор';
```

```
/* Все продающиеся товары по алфавиту */
SELECT *
FROM ProductData
WHERE Status = 'В продаже'
ORDER BY Name ASC;

/*Все продающиеся товары по алфавиту  */
SELECT *
FROM ProductData
WHERE Status = 'В продаже'
ORDER BY Name DESC;

/* Все продающиеся товары по дате продажи */
SELECT *
FROM ProductData
WHERE Status = 'В продаже'
ORDER BY Selling_date ASC;

/* Все продающиеся товары по дате продажи */
SELECT *
FROM ProductData
WHERE Status = 'В продаже'
ORDER BY Selling_date DESC;

/* Продающиеся товары по категориям */
SELECT *
FROM ProductData
WHERE category = '?' and status = 'В продаже';

/* "Горячие товары" */
SELECT *
FROM (
   SELECT *
   FROM ProductData
   ORDER BY bet_count DESC)
WHERE ROWNUM <= 5;

/* Продающиеся товары определенного пользователя
или собственные выставленные на продажу товары */
SELECT *
FROM ProductData
WHERE status = 'В продаже' AND seller = '?';
```

```
/* Свои проданные товары */
SELECT *
FROM ProductData
WHERE status = 'Продан' AND seller = '?';

/* Свои снятые с продажи товары */
SELECT *
FROM ProductData
WHERE status = 'Снят с продажи' AND seller = '?';

/* Свои выигранные торги */
SELECT ProductData.id, ProductData.name, ProductData.description, d3.price as final_price
FROM ProductData JOIN
   (SELECT d1.id, d1.buyer, d1.price, d2.*
   FROM (SELECT id, buyer, product, price
         FROM DealJournal) d1
         JOIN
      (SELECT product, max(price) as mx_price
         FROM DealJournal
         GROUP BY product) d2
         ON d1.product = d2.product
   WHERE Buyer = '?' and price = mx_price) d3
   ON ProductData.id = d3.product
WHERE ProductData.status = 'Продан';

/* Свои проигранные торги */
SELECT ProductData.id, ProductData.name, ProductData.description, d3.price, d3.mx_price as final_price
FROM ProductData JOIN
   (SELECT d1.id, d1.buyer, d1.price, d2.*
   FROM (SELECT id, buyer, product, price
         FROM DealJournal) d1
         JOIN
      (SELECT product, max(price) as mx_price
         FROM DealJournal
         GROUP BY product) d2
         ON d1.product = d2.product
   WHERE Buyer = '?' and price <> mx_price) d3
   ON ProductData.id = d3.product
WHERE ProductData.status = 'Продан';

/* Собственные текущие торги, где пользователь выигрывает*/
SELECT ProductData.id, ProductData.name, ProductData.description, d3.price
```

```
FROM ProductData JOIN
   (SELECT d1.id, d1.buyer, d1.price, d2.*
   FROM (SELECT id, buyer, product, price
         FROM DealJournal) d1
         JOIN
      (SELECT product, max(price) as mx_price
         FROM DealJournal
         GROUP BY product) d2
         ON d1.product = d2.product
   WHERE Buyer = '?' and price = mx_price) d3
   ON ProductData.id = d3.product
WHERE ProductData.status = 'В продаже';

/* Собственные текущие торги, где пользователь проигрывает*/
SELECT ProductData.id, ProductData.name, ProductData.description, d3.price, d3.mx_price as current_price
FROM ProductData JOIN
   (SELECT d1.id, d1.buyer, d1.price, d2.*
   FROM (SELECT id, buyer, product, price
         FROM DealJournal) d1
         JOIN
      (SELECT product, max(price) as mx_price
         FROM DealJournal
         GROUP BY product) d2
         ON d1.product = d2.product
   WHERE Buyer = '?' and price <> mx_price) d3
   ON ProductData.id = d3.product
WHERE ProductData.status = 'В продаже';
```

3.3. Запросы изменения данных в таблицах базы данных

```
/* Авторизация в системе */
UPDATE UserData
SET online_status = '1'
WHERE login = '?' AND blocking_status <> '1';

/* Деавторизация */
UPDATE UserData
SET online_status = '0'
WHERE login = '?';

/* Добавление нового пользователя */
INSERT INTO UserData(login, parole, type) VALUES ('?', '?', '?');
```

```
/* Удаление пользователя */
DELETE
FROM DealJournal
WHERE buyer = '?';

DELETE
FROM DealJournal
WHERE product IN (
   SELECT id
   FROM ProductData
   WHERE seller = '?' AND (status = 'В продаже' OR status = 'Снят с продажи'));

DELETE
FROM ProductData
WHERE seller = '?' AND (status = 'В продаже' OR status = 'Снят с продажи');

UPDATE ProductData
SET seller = NULL
WHERE seller = '?' AND status = 'Продан';

DELETE
FROM UserData
WHERE login = '?';

/* Редактирование данных пользователя */
UPDATE UserData
SET online_status = '0'
WHERE login = '?';

UPDATE UserData
SET blocking_status = '0'
WHERE login = '?';

UPDATE UserData
SET blocking_status = '1'
WHERE login = '?';

/* Блокировка пользователя */
UPDATE UserData
SET blocking_status = '1'
WHERE login = '?';
```

```
DELETE
FROM DealJournal
WHERE buyer = '?' AND product IN (
   SELECT id
   FROM ProductData
   WHERE status = 'В продаже');

UPDATE ProductData
SET status = 'Снят с продажи'
WHERE seller = '?' AND status = 'В продаже';

/* Разблокировка пользователя */
UPDATE UserData
SET blocking_status = '0'
WHERE login = '?';

/* Задание срока продажи всех лотов */
UPDATE ProductData
SET selling_interval = ?;

UPDATE ProductData
SET selling_date = creating_date + selling_interval;
UPDATE ProductData
SET status = 'Продан'
WHERE bet_count <> 0 AND selling_date <= SYSDATE AND status = 'В продаже';

UPDATE ProductData
SET status = 'Снят с продажи'
WHERE bet_count = 0 AND selling_date <= SYSDATE AND status = 'В продаже';

/* Задание срока продажи лота*/
UPDATE ProductData
SET selling_interval = ?
WHERE id = ?;

UPDATE ProductData
SET status = 'Продан'
WHERE bet_count <> 0 AND selling_date <= SYSDATE AND status = 'В продаже' AND id = ?;

UPDATE ProductData
SET status = 'Снят с продажи'
WHERE bet_count = 0 AND selling_date <= SYSDATE AND status = 'В продаже' AND id = ?;
```

```
/* Задание шага повышения цены всех лотов */
UPDATE ProductData
SET price_step = ?;

/* Залание шага повышения цены лота */
UPDATE ProductData
SET price_step = ?
WHERE id = ?;

/* Снятие лота с продажи */
UPDATE ProductData
SET status = 'Снят с продажи'
WHERE id = ? AND status = 'В продаже';

/* Выставление лота на продажу */
INSERT INTO ProductData(name, description, category, start_price, price, price_step,
   creating_date, selling_date, selling_interval, seller) VALUES
   ('?', '?', '?', ?,?, ?, SYSDATE, SYSDATE + ?,?, '?');

/* Повышение цены лота на определенный шаг (сделать ставку) */
UPDATE ProductData
SET bet_count = bet_count + 1
WHERE id = ?;

DELETE
FROM DealJournal
WHERE buyer = '?' AND product = ?;

INSERT INTO DealJournal(datetime,price,product,buyer) VALUES
(SYSDATE, (SELECT price FROM ProductData WHERE id = ?), ?, '?');

UPDATE ProductData
SET price = price + price_step
WHERE id = ?;

/* Редактирование данных лота */
UPDATE ProductData
SET description = '?'
WHERE id = ?;
```

4. Инструкции по работе с сетевой информационной системой

4.1. Инструкция администратора

- Авторизация

Перед тем как попасть в систему, администратор должен авторизоваться. Он должен ввести свои учетные данные: заполнить поле "Login" своим уникальным логином администратора и ввести свой пароль в поле "Password". После ввода учетных данных, он должен нажать на кнопку "Log in", чтобы войти в систему (рисунок 3). При успешном входе в систему администратор попадает на приветственную страницу (рисунок 4), иначе он получает сообщение о вводе неверных учетных данных и может попробовать ввести их снова.

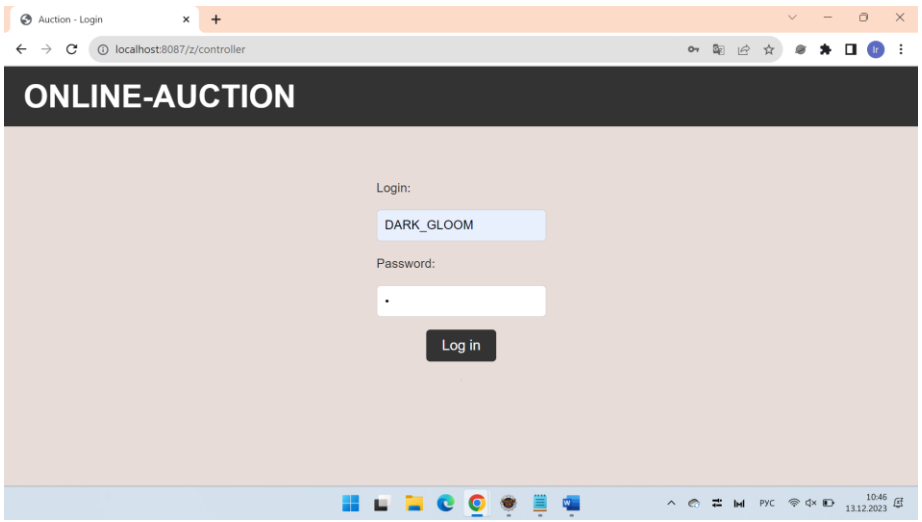

Рисунок 3 - Авторизация в системе

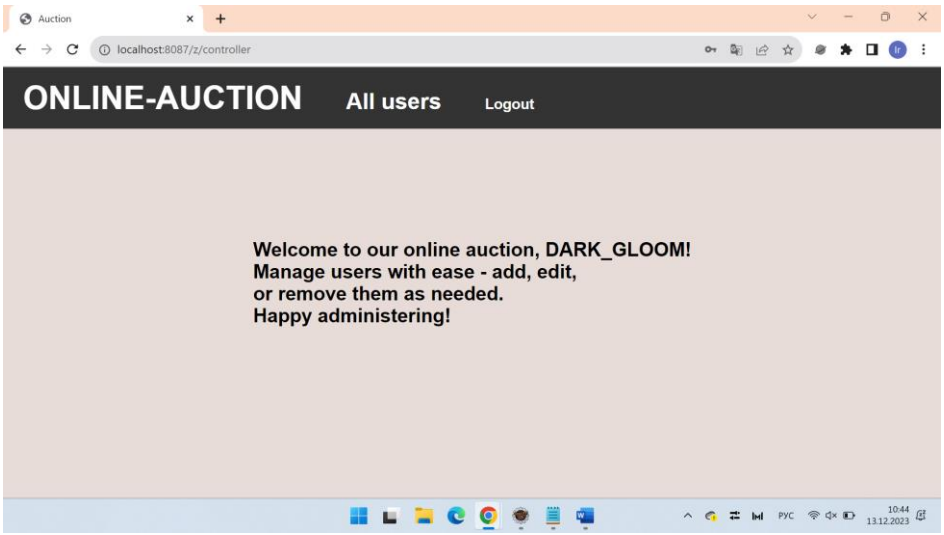

Рисунок 4 - Приветственная страница администратора

- Просмотр списка пользователей

Администратор может перейти на вкладку "All users", где хранится информация обо всех пользователях системы (рисунок 5). Он может выбрать какие категории пользователей ему нужно просмотреть: "All users" (по умолчанию), "Admins", "Moders", "Normal Users", "Authorized", "Unauthorized". В зависимости от выбранной категории-фильтра изменяется отображаемый список.

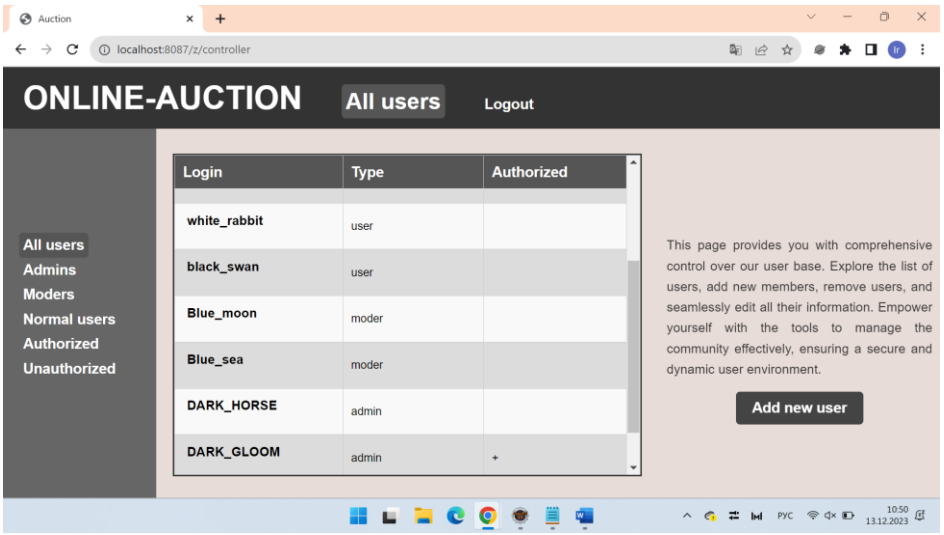

Рисунок 5 - Просмотр списка пользователей

- Просмотр данных пользователя

Во время просмотра списка пользователей администратор может нажимать на логины пользователей. После нажатия открывается карточка пользователя, на которой администратор может просматривать данные выбранного пользователя (рисунок 6).

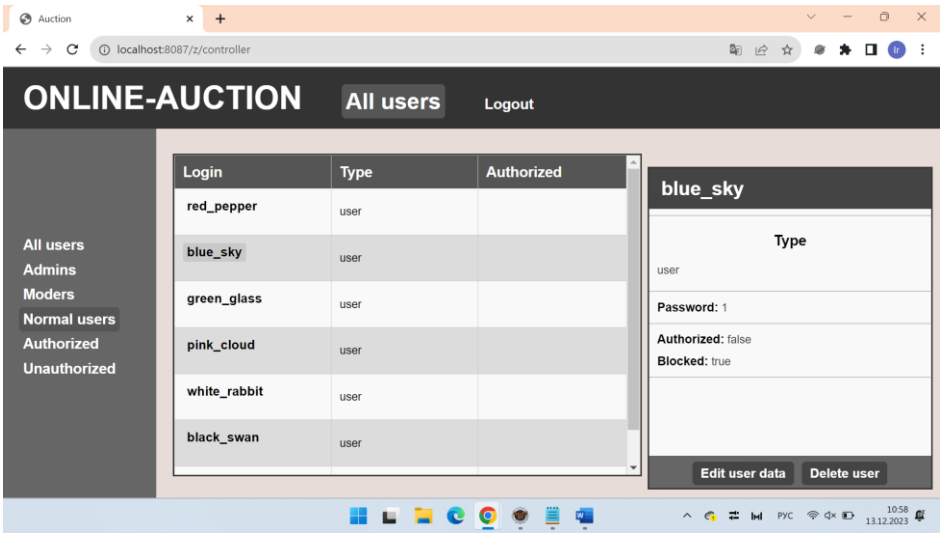

Рисунок 6 - Просмотр данных пользователя

- Управление данными пользователя

Администратор может не только просматривать, но и редактировать данные любого выбранного пользователя: блокировать/разблокировать, изменять пароль пользователя. Для этого ему надо нажать на кнопку "Edit user data", и после заполнения полей нажать на кнопку "Save changes" (рисунок 7). Если какие-то данные не введены, изменения сохранены не будут и отобразится соответствующее предупреждение.

Также администратор может удалить выбранного пользователя, выбрав кнопку "Delete user". На карточке самого администратора данная кнопка отсутствует - администратор не может удалить сам себя.

Обе кнопки: "Edit user data" и "Delete user" расположены на карточке выбранного пользователя (рисунок 6).

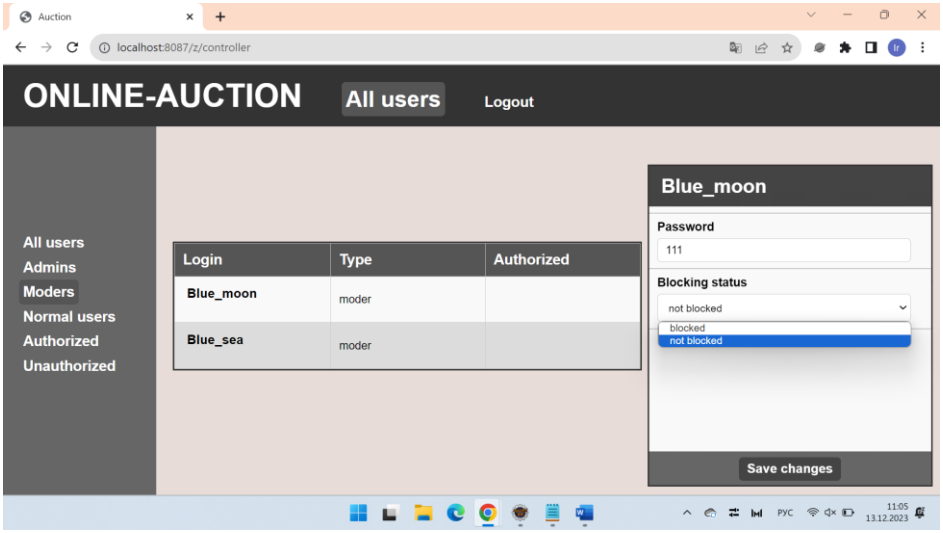

Рисунок 7 - Изменение данных пользователя

- Добавление нового пользователя

Администратор может добавить нового пользователя, нажав на кнопку "Add new user" (рисунок 5), и после заполнения полей нажать на кнопку "Save new user" (рисунок 8). Если какие-то данные не введены или введенный логин неуникален (пользователь с таким логином уже есть в системе), администратор получает соответствующее сообщение и может попробовать ввести данные снова.

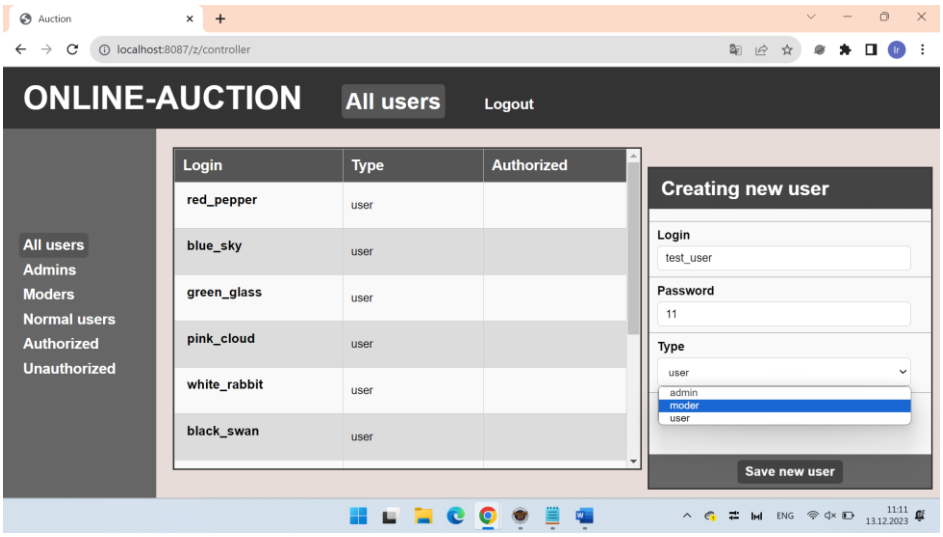

Рисунок 8 - Добавление нового пользователя

- Переход к начальной странице и деавторизация

В любой момент работы с системой администратор может вернуться на начальную страницу, нажав на кнопку "ONLINE-AUCTION". После окончания работы администратор может выйти из системы, нажав на кнопку "Logout".

4.2. Инструкция модератора

- Авторизация

Перед тем как попасть в систему, модератор должен авторизоваться: заполнить поле "Login" и поле "Password". После ввода данных, он должен нажать на кнопку "Log in" (рисунок 3). При успешном входе модератор попадает на приветственную страницу (рисунок 9), иначе он получает сообщение о вводе неверных учетных данных и может попробовать ввести их снова.

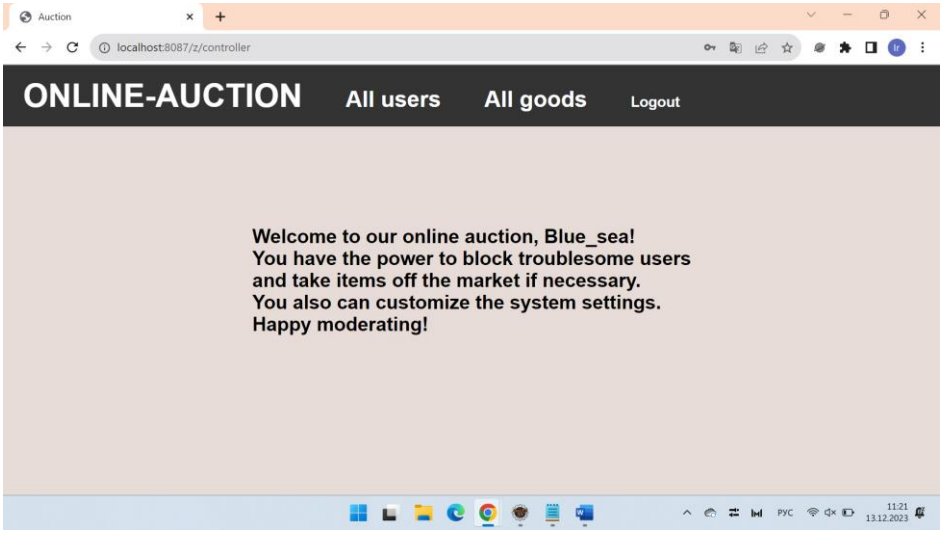

Рисунок 9 - Приветственная страница модератора

- Просмотр списка пользователей

Модератор может перейти на вкладку "All users", где хранится информация обо всех обычных пользователях системы (рисунок 10). Он может выбрать какие категории пользователей ему нужно просмотреть: "All users" (по умолчанию), "Blocked", "Not blocked.

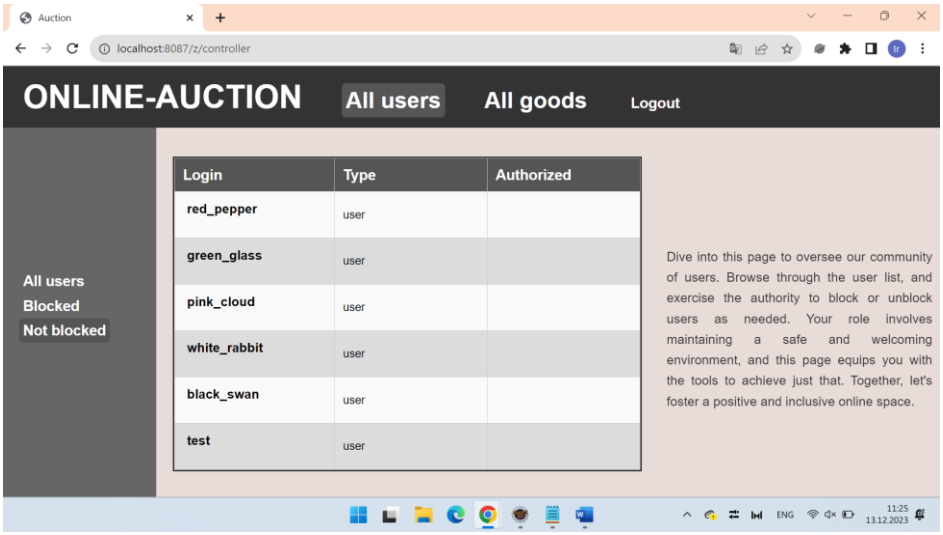

Рисунок 10 - Просмотр списка обычных пользователей

- Просмотр данных пользователя, блокировка и разблокировка пользователей

Во время просмотра списка пользователей модератор может нажимать на логины пользователей. После нажатия открывается карточка с данными пользователя (рисунок 11). Доступна блокировка выбранного пользователя - для этого на карточке отображается кнопка "Block user" (если пользователь не заблокирован), либо "Unblock user" (если заблокирован).

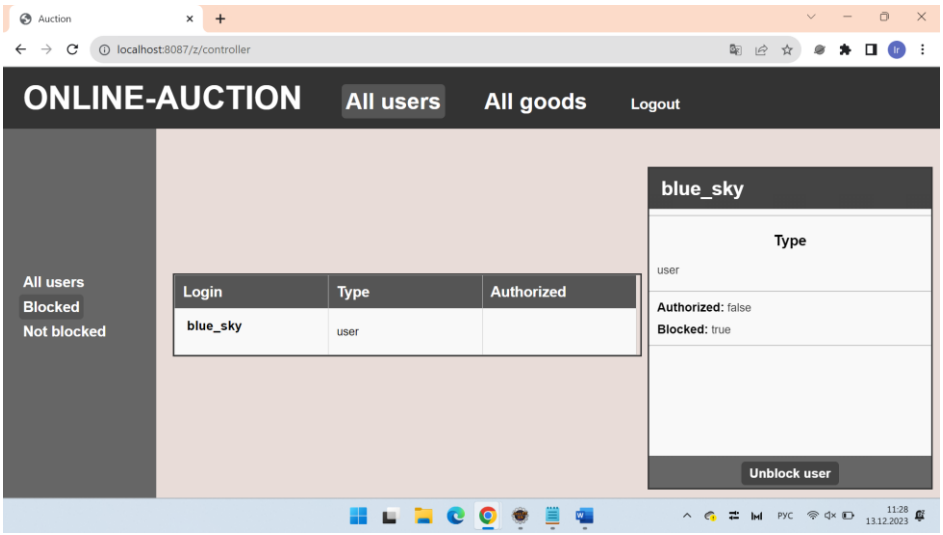

Рисунок 11 - Просмотр данных пользователя

- Просмотр списка товаров

Модератор может перейти на вкладку "All goods", где хранится информация обо всех продающихся лотах системы (рисунок 12). Там он может выбрать какие категории товаров ему надо просмотреть: "All goods"(по умолчанию), "Hot goods", "Food", "Art", "Household chemicals", "Electronics", "Furniture", "Beauty and health", "Music", "Other".

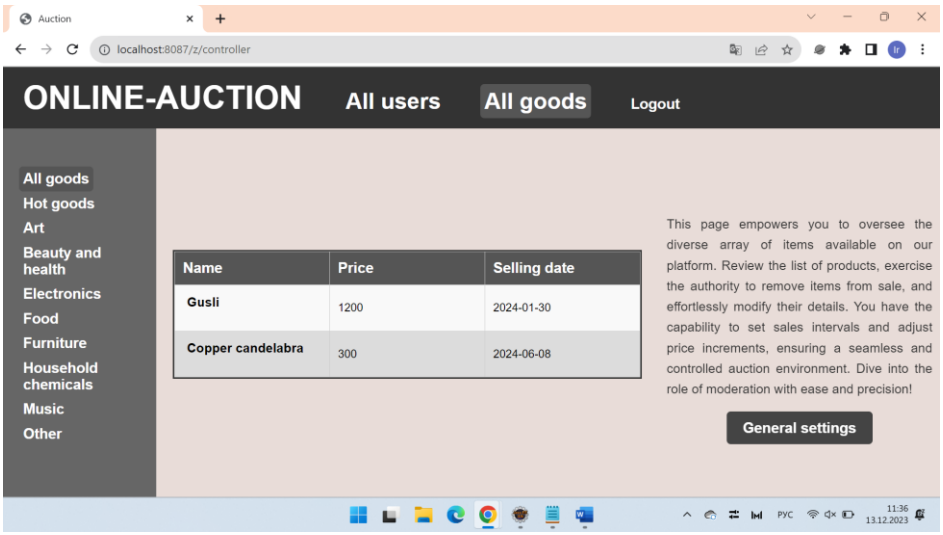

Рисунок 12 - Просмотр списка продающихся товаров

- Просмотр данных товара

Модератор может просматривать данные выбранного товара, нажимая на его название из списка (рисунок 13).

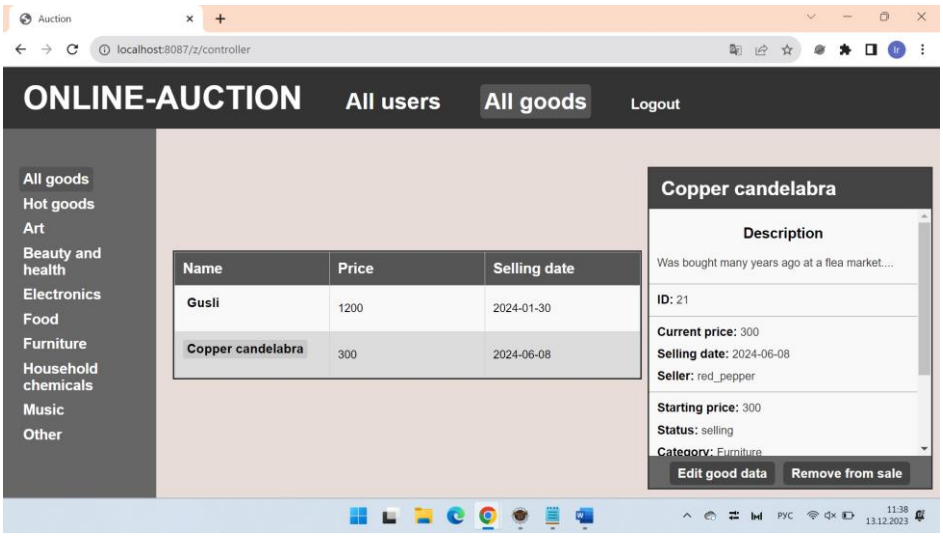

Рисунок 13 - Просмотр данных товара

- Управление данными товара

Модератор может нажать на кнопку "Edit good data", чтобы редактировать данные продающихся товаров, изменяя для них шаг цены и интервал продажи. После заполнения полей

необходимо нажать на кнопку "Save changes" (рисунок 14). Если данные введены некорректно или если какие-то поля не заполнены, изменения сохранены не будут и модератор получит соответствующее предупреждение.

Кроме того, модератор имеет возможность снимать лоты с продажи, нажимая на кнопку "Remove from sale".

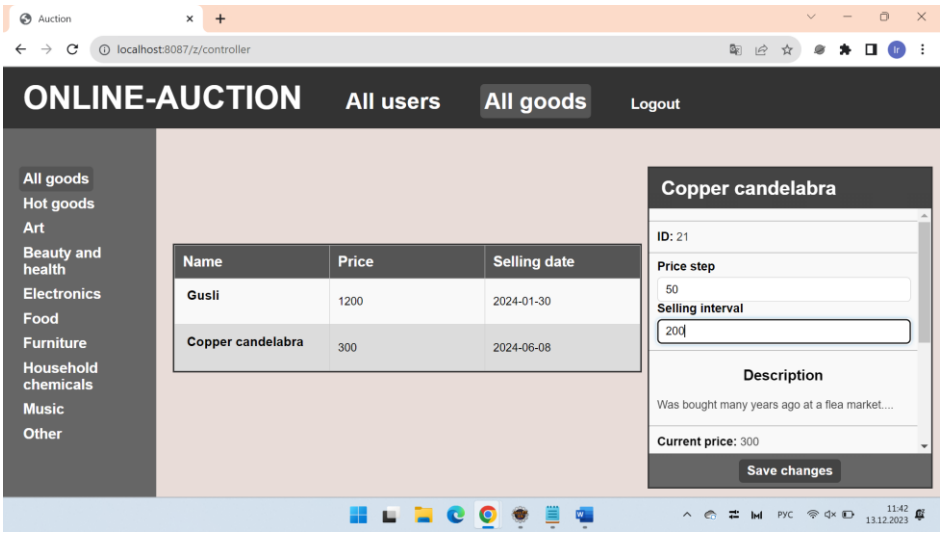

Рисунок 14 - Изменение данных товара

- Управление настройками системы

На странице "All goods" модератору доступна кнопка "General settings" (рисунок 12), позволяющая настраивать значения констант шаг цены и интервал продажи, которые по умолчанию действуют для всех добавляемых в систему товаров (рисунок 15). После ввода данных нужно нажать на кнопку "Save settings". Если данные введены некорректно, модератор увидит соответствующее предупреждение.

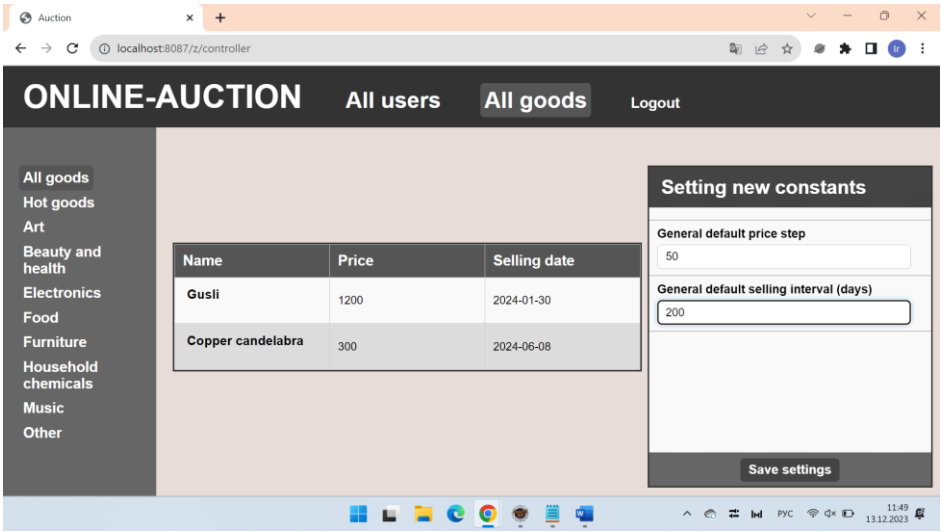

Рисунок 15 - Изменение настроек аукциона

- Переход к начальной странице и деавторизация

В любой момент работы с системой модератор может вернуться на начальную страницу, нажав на кнопку "ONLINE-AUCTION". После окончания работы модератор может выйти из системы, нажав на кнопку "Logout".

4.3. Инструкция обычного пользователя

- Авторизация

Перед тем как попасть в систему, пользователь должен авторизоваться. Он должен ввести свои учетные данные: заполнить поле "Login" своим уникальным логином пользователя и ввести свой пароль в поле "Password". После ввода учетных данных, он должен нажать на кнопку "Log in", чтобы войти в систему (рисунок 3). При успешном входе в систему пользователь попадает на приветственную страницу (рисунок 16), иначе он получает сообщение о вводе неверных учетных данных и может попробовать ввести их снова. Если пользователь заблокирован в системе, он также получает соответствующее сообщение.

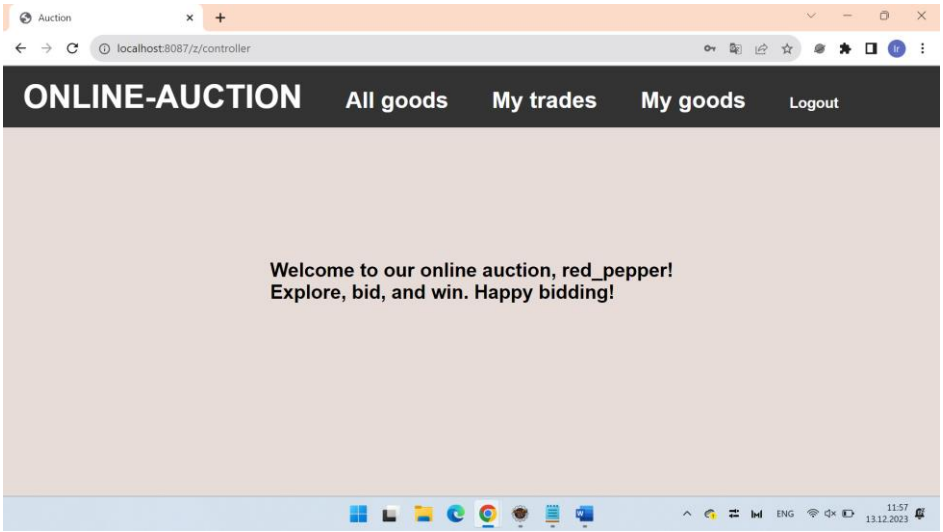

Рисунок 16 - Приветственная страница обычного пользователя

- Просмотр списка продающихся товаров

Пользователь может перейти на вкладку "All goods", где хранится информация обо всех продающихся лотах системы (рисунок 17). Там он может выбрать какие категории товаров ему надо просмотреть: "All goods" (по умолчанию), "Hot goods", "Food", "Art", "Household chemicals", "Electronics", "Furniture", "Beauty and health", "Music", "Other".

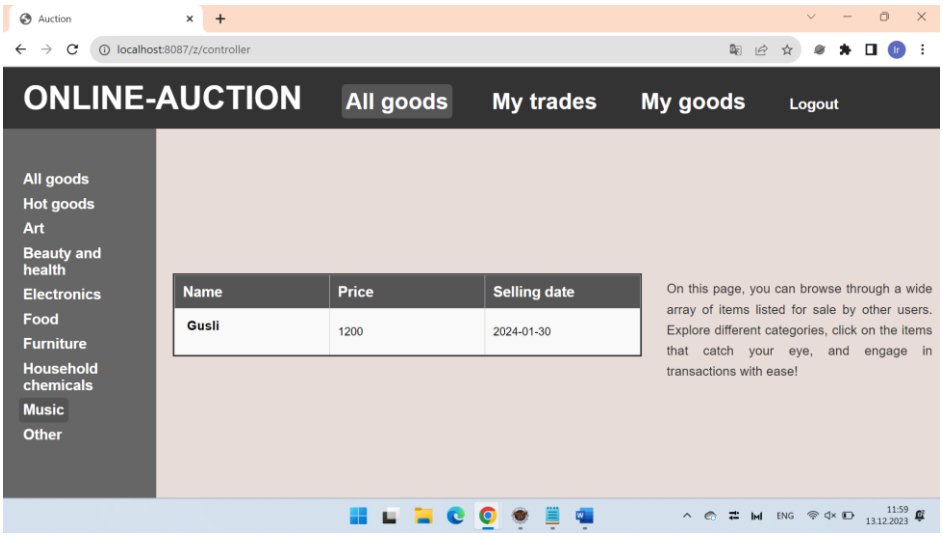

Рисунок 17 - Просмотр списка продающихся товаров

- Просмотр списка собственных товаров

Пользователь может перейти на вкладку "My goods", где хранится информация обо всех его лотах (рисунок 18). Там он может выбрать какие категории товаров ему надо просмотреть: "On sale" (по умолчанию), "Sold", "Removed from sale".

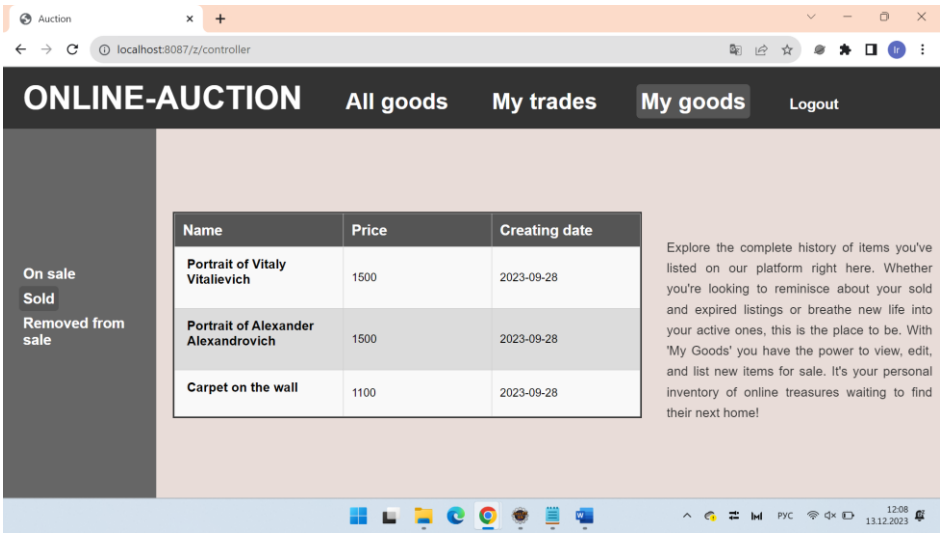

Рисунок 18 - Просмотр списка своих товаров

- Выставление товара на продажу

Пользователь может добавлять новые товары на продажу, находясь в разделе "Sold" на вкладке "My goods". Для этого необходимо нажать на кнопку "Create new good" (рисунок 19) и после заполнения полей (название, описание, категория и цена) сохранить данные, кликнув на кнопку "Save new good" (рисунок 20). Если данные введены некорректно или если какие-то поля не заполены, то товар не будет добавлен, а

пользователю будет показано соответствующее предупреждение.

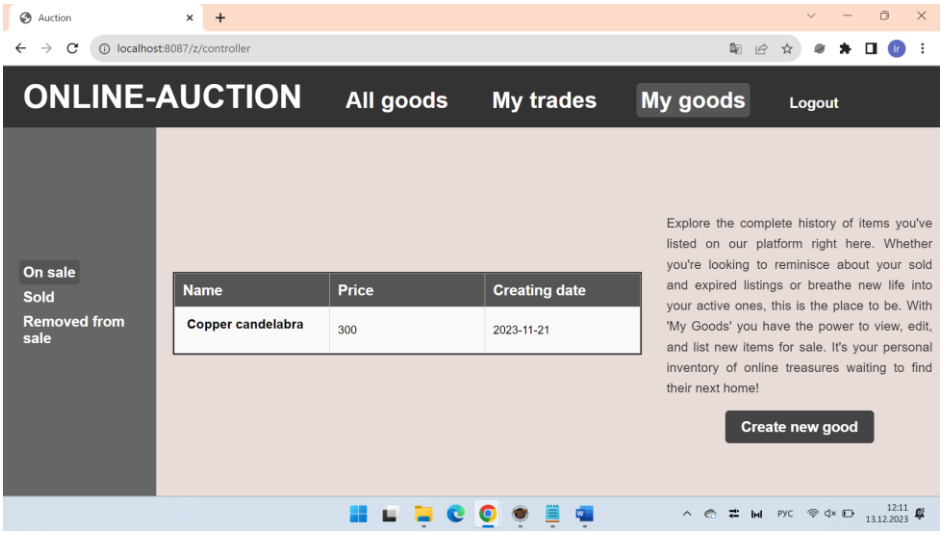

Рисунок 19 - Добавление данных нового товара в список своих продающихся товаров

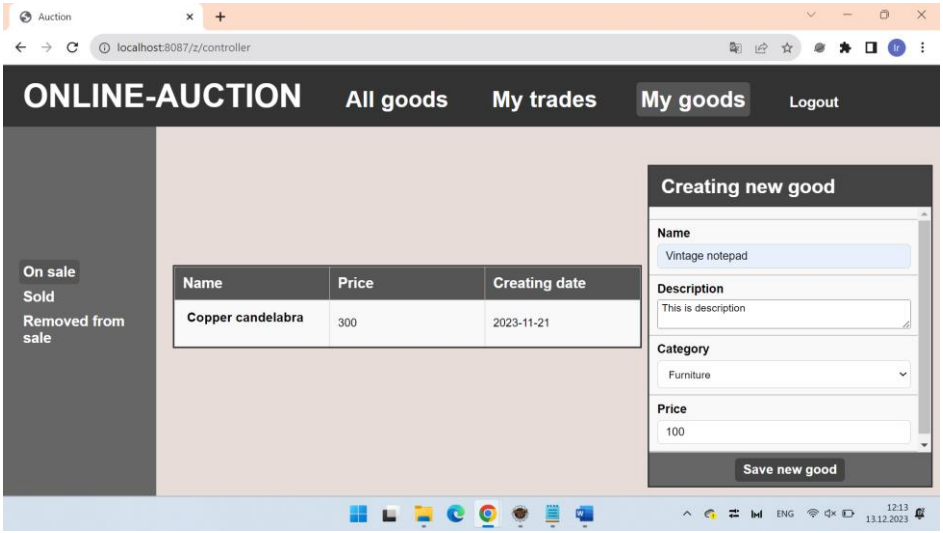

Рисунок 20 - Заполнение данных нового товара

- Просмотр данных товара

Во время просмотра списка продающихся товаров на вкладке "All goods" или списка собственных товаров на вкладке "My goods" пользователь может нажимать на названия товаров. После нажатия открывается карточка товара, на которой пользователь может просматривать данные выбранного товара. Вид карточки товара может отличаться в зависимости от того, принадлежит ли данный товар пользователю (рисунок 21, рисунок 22).

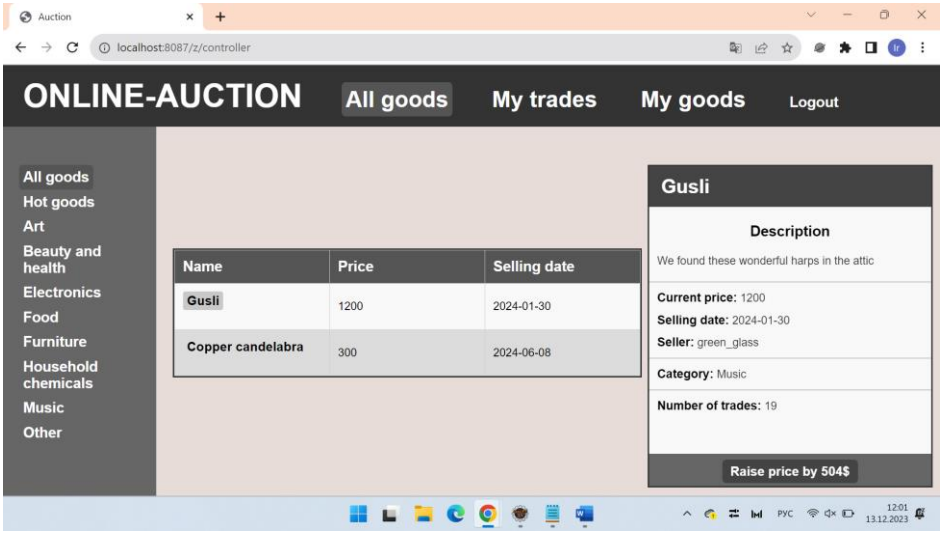

Рисунок 21 - Просмотр данных чужого товара

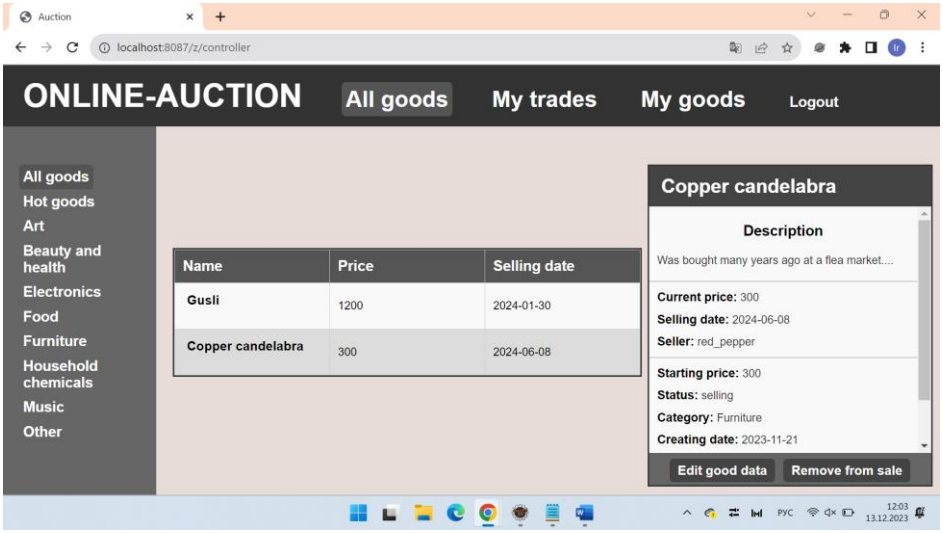

Рисунок 22 - Просмотр данных собственного товара

- Управление данными товара

На карточке собственного товара пользователю доступны две кнопки - "Edit good data" и "Remove from sale" (рисунок 22).

Он может нажать на кнопку "Edit good data", чтобы редактировать данные товара (его описание). После заполнения поля необходимо нажать на кнопку "Save changes" (рисунок 23).

Пользователь имеет возможность снимать свои лоты с продажи, нажав на кнопку "Remove from sale".

Когда пользователь просматривает карточку своего товара, но тот уже не находится в продаже, никаких изменений в товар внести уже нельзя и на карточке не будет отображено никаких кнопок (рисунок 24).

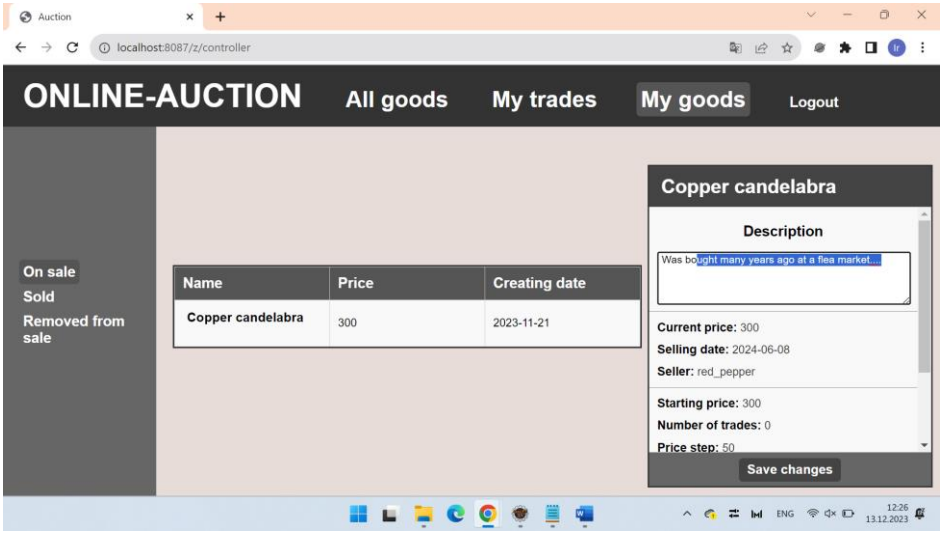

Рисунок 23 - Изменение данных своего товара

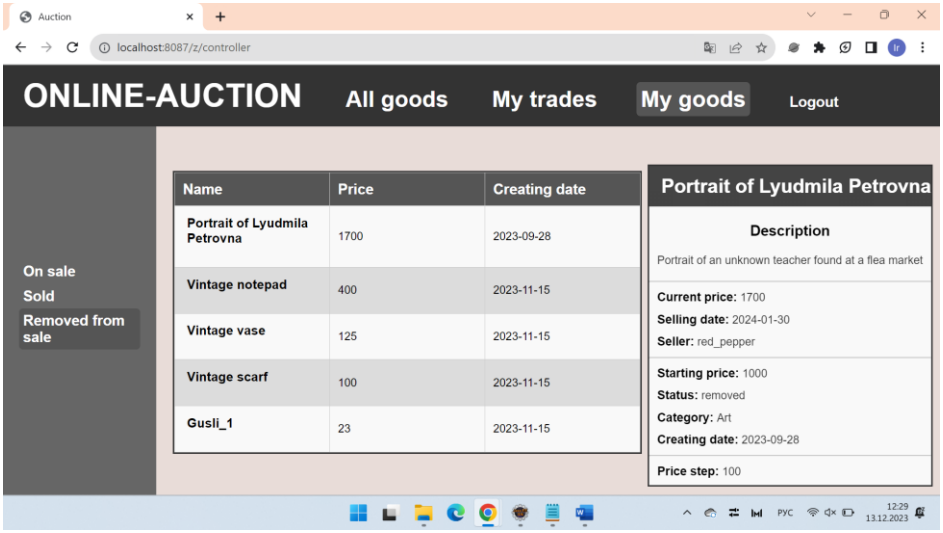

Рисунок 24 - Просмотр данных снятого с продажи товара

- Повышение цены на товар

На карточке чужого продающегося товара пользователю доступна кнопка - "Raise price by n$", где n - шаг цены для конкретного товара (рисунок 25). Нажимая на эту кнопку, пользователь может поучаствовать в аукционе (торгах за товар). После нажатия цена товара и счетчик ставок увеличивается, а данные товара (если их еще там нет) сохраняются в раздел "My trades" пользователя.

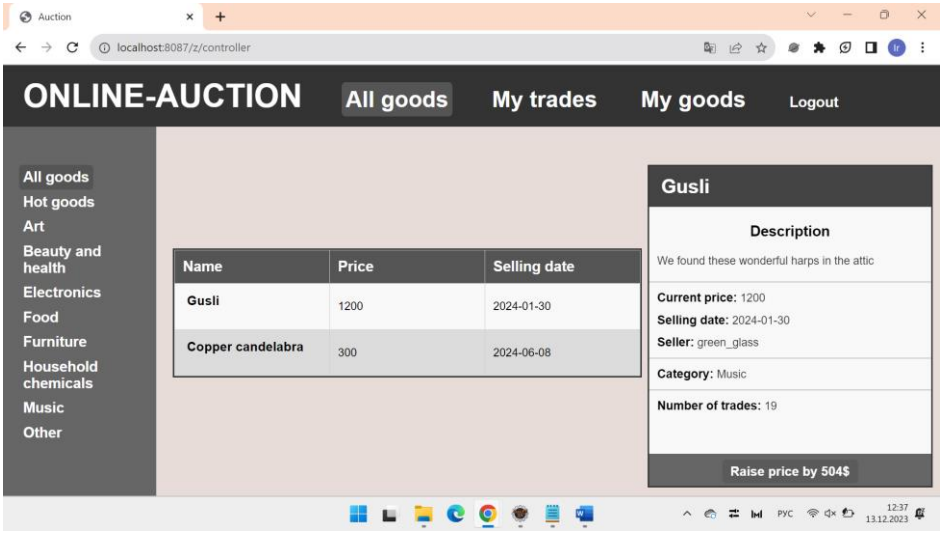

Рисунок 25 - Повышение цены на товар

- Просмотр списка торгов

Пользователь может перейти на вкладку "My trades", где хранится информация обо всех его торгах. Там он может выбрать какие категории ставок ему надо просмотреть: "All trades" (по умолчанию), "Successful", "Unsuccessful".

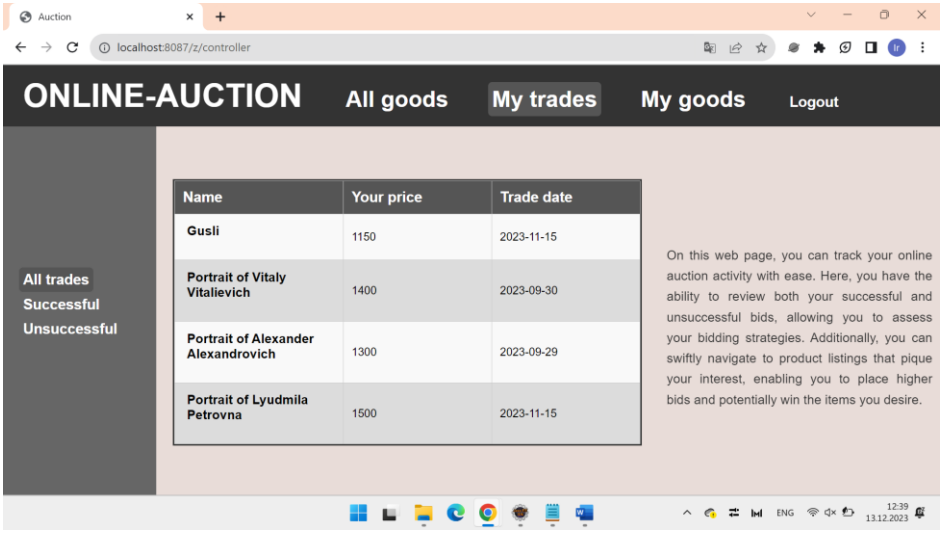

Рисунок 26 - Просмотр списка торгов

- Просмотр данных торга по товару

Во время просмотра списка торгов пользователь может нажимать на названия товаров. После нажатия открывается карточка торга (рисунок 27). На карточке отображаются краткие сведения о товаре и состоянии торгов за товар. Если товар все еще находится в продаже и на него можно поднять цену - отображается кнопка "Go to product card", при нажатии на которую открывается карточка товара. В любом случае

пользователь может перейти к карточке товара, нажав на название товара в верхней части карточки торга.

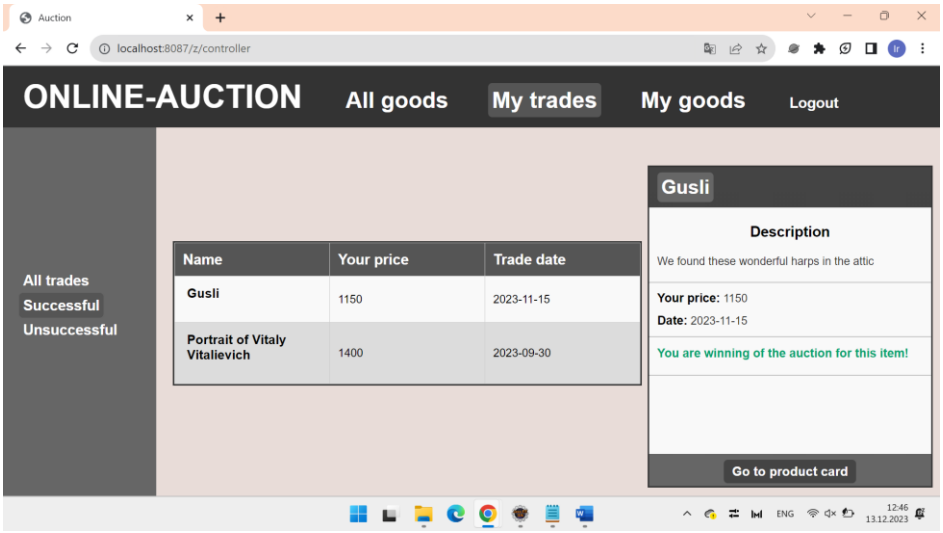

Рисунок 27 - Просмотр данных торгов по товару

- Переход к начальной странице и деавторизация

В любой момент работы с системой пользователь может вернуться на начальную страницу, нажав на кнопку "ONLINE-AUCTION". После окончания работы пользователь может выйти из системы, нажав на кнопку "Logout".

5. Листинг основных классов программы с комментариями Javadoc

5.1. FrontController

```
/**
 * The FrontController class extends HttpServlet and serves as the main controller
 * for handling HTTP requests.
 */
public class FrontController extends HttpServlet {

    private static final long serialVersionUID = 1L;

    /**
   * Handles HTTP GET requests by calling the processRequest method.
   *
   * @param request - the HttpServletRequest object.
   * @param response - the HttpServletResponse object.
   * @throws ServletException - if a servlet-specific error occurs.
   * @throws IOException - if an I/O error occurs.
   */
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
* Handles HTTP POST requests by calling the processRequest method.
*
* @param request - the HttpServletRequest object.
* @param response - the HttpServletResponse object.
* @throws ServletException - if a servlet-specific error occurs.
* @throws IOException - if an I/O error occurs.
*/
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
* Processes the HTTP request by setting character encoding, determining the
* command from the JSP, and executing the corresponding action command.
*
* @param request - the HttpServletRequest object.
* @param response - the HttpServletResponse object.
* @throws ServletException - if a servlet-specific error occurs.
* @throws IOException - if an I/O error occurs.
*/
    private void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String page = null;
        // определение команды, пришедшей из JSP
        ActionFactory client = new ActionFactory();
        ActionCommand command = client.defineCommand(request);
        /*
         * вызов реализованного метода execute() и передача параметров
         * классу-обработчику конкретной команды
         */
        page = command.execute(request);
        // метод возвращает страницу ответа
        // page = null; // поэксперементировать!
        if (page != null) {
            RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(page);
```

```
                // вызов страницы ответа на запрос
                dispatcher.forward(request, response);
            } else {
                // установка страницы с сообщением об ошибке
                page = ConfigurationManager.getProperty("path.page.index");
                request.getSession().setAttribute("nullPage",
MessageManager.getProperty("message.nullpage"));
                response.sendRedirect(request.getContextPath() + page);
            }
        }
}
```

5.2. Good

```
/**
 * Class representing a good.
 *
 * @author Verbickaya, Stepanova
 */
public class Good {

    /** Unique identifier for the good. */
    private int id;

    /** Name of the good. */
    private String name;

    /** Description of the good. */
    private String description;

    /** Category of the good. */
    private String category;

    /** Starting price of the good. */
    private int startPrice;

    /** Price of the good. */
    private int price;

    /** Price step for the good. */
    private int priceStep;

    /** Number of bids on the good */
    private int betCount;
```

```
    /** Creation date of the good */
    private Date creatingDate;

    /** Selling date of the good */
    private Date sellingDate;

    /** Interval for selling the good. */
    private int sellingInterval;

    /** Status of the good (selling, sold, removed). */
    private GoodStatusEnum status;

    /** Seller's name for the good. */
    private String seller;

    /**
  * Default constructor.
  */
    public Good() {
    }

    /**
  * Constructor with parameters.
  *
  * @param id - unique identifier for the good.
  * @param name - name of the good.
  * @param description - description of the good.
  * @param category - category of the good.
  * @param startPrice - starting price of the good.
  * @param price - current price of the good.
  * @param priceStep - price step for the good.
  * @param betCount - number of bids on the good.
  * @param creatingDate - creation date of the good.
  * @param sellingDate - selling date of the good.
  * @param sellingInterval - interval for selling the good.
  * @param status - status of the good (selling, sold, removed).
  * @param seller -seller's name for the good.
  */
    public Good(int id, String name, String description, String category, int startPrice, int price, int
priceStep,
                int betCount, Date creatingDate, Date sellingDate, int sellingInterval, GoodStatusEnum
status,
```

```
            String seller) {
        setId(id);
        setName(name);
        setDescription(description);
        setCategory(category);
        setStartPrice(startPrice);
        setPrice(price);
        setPriceStep(priceStep);
        setBetCount(betCount);
        setCreatingDate(creatingDate);
        setSellingDate(sellingDate);
        setSellingInterval(sellingInterval);
        setStatus(status);
        setSeller(seller);
    }

    /**
  * Compares this good to another object for equality.
  *
  * @param obj - object to compare to.
  * @return true if the objects are equal, false otherwise.
  */
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Good good = (Good) obj;
        return id == good.getId() && name.equals(good.getName()) &&
description.equals(good.getDescription())
                && category.equals(good.getCategory()) && startPrice == good.getStartPrice() &&
price == good.getPrice()
                && priceStep == good.getPriceStep() && betCount == good.getBetCount()
                && creatingDate.equals(good.getCreatingDate()) &&
sellingDate.equals(good.getSellingDate())
                && sellingInterval == good.getSellingInterval() && status.equals(good.getStatus())
                && seller.equals(good.getSeller());
    }

    /**
  * Calculates a hash code for this good.
  *
```

```
    * @return hash code value.
    */
      @Override
      public int hashCode() {
            return Objects.hash(id, name, description, category, startPrice, price, priceStep, betCount,
creatingDate,
                        sellingDate, sellingInterval, status, seller);
      }

      /**
    * Gets the unique identifier for the good.
    *
    * @return unique identifier for the good.
    */
      public int getId() {
            return id;
      }

      /**
    * Sets the unique identifier for the good.
    *
    * @param id - unique identifier for the good.
    */
      public void setId(int id) {
            this.id = id;
      }

      /**
    * Gets the name of the good.
    *
    * @return name of the good.
    */
      public String getName() {
            return name;
      }

      /**
    * Sets the name of the good.
    *
    * @param name - name of the good.
    */
      public void setName(String name) {
            this.name = name;
```

```
    }

    /**
  * Gets the category of the good.
  *
  * @return category of the good.
  */
    public String getCategory() {
            return category;
    }

    /**
  * Sets the category of the good.
  *
  * @param category - category of the good.
  */
    public void setCategory(String category) {
            this.category = category;
    }

    /**
  * Gets the description of the good.
  *
  * @return description of the good.
  */
    public String getDescription() {
            return description;
    }

    /**
  * Sets the description of the good.
  *
  * @param description - description of the good.
  */
    public void setDescription(String description) {
            this.description = description;
    }

    /**
  * Gets the starting price of the good.
  *
  * @return Starting price of the good.
  */
```

```
    public int getStartPrice() {
        return startPrice;
    }

    /**
* Sets the starting price of the good.
*
* @param startPrice - starting price of the good.
*/
    public void setStartPrice(int startPrice) {
        this.startPrice = startPrice;
    }

    /**
* Gets the price of the good.
*
* @return price of the good.
*/
    public int getPrice() {
        return price;
    }

    /**
* Sets the price of the good.
*
* @param price - price of the good.
*/
    public void setPrice(int price) {
        this.price = price;
    }

    /**
* Gets the price step for the good.
*
* @return price step for the good.
*/
    public int getPriceStep() {
        return priceStep;
    }

    /**
* Sets the price step for the good.
*
```

```
* @param priceStep - price step for the good.
*/
   public void setPriceStep(int priceStep) {
         this.priceStep = priceStep;
   }

   /**
* Gets the number of bids on the good.
*
* @return number of bids on the good.
*/
   public int getBetCount() {
         return betCount;
   }

   /**
* Sets the number of bids on the good.
*
* @param betCount - number of bids on the good.
*/
   public void setBetCount(int betCount) {
         this.betCount = betCount;
   }

   /**
* Gets creation date of the good.
*
* @return creation date of the good.
*/
   public Date getCreatingDate() {
         return creatingDate;
   }

    /**
* Sets creation date of the good.
*
* @param creatingDate - creation date of the good.
*/
   public void setCreatingDate(Date creatingDate) {
         this.creatingDate = creatingDate;
   }

   /**
```

```
* Gets selling date of the good.
*
* @return selling date of the good.
*/
   public Date getSellingDate() {
         return sellingDate;
   }

   /**
* Sets selling date of the good.
*
* @param sellingDate - selling date of the good.
*/
   public void setSellingDate(Date sellingDate) {
         this.sellingDate = sellingDate;
   }

   /**
* Gets the interval for selling the good.
*
* @return interval for selling the good.
*/
   public int getSellingInterval() {
         return sellingInterval;
   }

   /**
* Sets the interval for selling the good.
*
* @param sellingInterval - interval for selling the good.
*/
   public void setSellingInterval(int sellingInterval) {
         this.sellingInterval = sellingInterval;
   }

   /**
* Gets the status of the good.
*
* @return status of the good.
*/
   public GoodStatusEnum getStatus() {
         return status;
   }
```

```
    /**
   * Sets the status of the good.
   *
   * @param status - status of the good.
   */
    public void setStatus(GoodStatusEnum status) {
        this.status = status;
    }

    /**
   * Gets the seller's name for the good.
   *
   * @return seller's name for the good.
   */
    public String getSeller() {
        return seller;
    }

    /**
   * Sets the seller's name for the good.
   *
   * @param seller - seller's name for the good.
   */
    public void setSeller(String seller) {
        this.seller = seller;
    }
}
```

5.3. Trade

```
/**
 * Class representing a trade in the system.
 *
 * @author Verbickaya, Stepanova
 */
public class Trade {

    /** Unique identifier of the trade. */
    private int id;

    /** Name of the trade. */
    private String name;
```

```
    /** Description of the trade. */
    private String description;

    /** Date of the trade. */
    private Date date;

    /** Price of the trade. */
    private int price;

    /** Current price of the trade. */
    private int currentPrice;

    /** Unique identifier of the good. */
    private int productId;

    /** Buyer's login for the trade. */
    private String buyer;

    /**
     * Default constructor.
     */
    public Trade() {

    }

    /**
* Constructor with parameters.
*
* @param id - unique identifier of the trade.
* @param name - name of the trade.
* @param description - description of the trade.
* @param date - date of the trade.
* @param price - price of the trade.
* @param currentPrice - current price of the trade.
* @param product - unique identifier of the good.
* @param buyer - buyer's login for the trade.
*/
    public Trade(int id, String name, String description, Date date, int price, int currentPrice, int product,
                String buyer) {
        setId(id);
        setName(name);
        setDescription(description);
        setDate(date);
```

```
            setPrice(price);
            setCurrentPrice(currentPrice);
            setProductId(product);
            setBuyer(buyer);
        }

        /**
     * Compares this trade to another object for equality.
     *
     * @param obj - object to compare to.
     * @return true if the objects are equal, false otherwise.
     */
        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null || getClass() != obj.getClass())
                return false;
            Trade trade = (Trade) obj;
            return id == trade.getId() && name.equals(trade.getName()) &&
description.equals(trade.getDescription())
                    && date.equals(trade.getDate()) && price == trade.getPrice() && currentPrice ==
trade.getCurrentPrice()
                    && productId == trade.getProductId() && buyer.equals(trade.getBuyer());
        }

        /**
     * Calculates a hash code for this trade.
     *
     * @return hash code value.
     */
        @Override
        public int hashCode() {
            return Objects.hash(id, name, description, date, price, currentPrice, productId, buyer);
        }

        /**
     * Gets the identifier for the trade.
     *
     * @return identifier for the trade.
     */
        public int getId() {
            return id;
```

```
    }

    /**
 * Sets identifier for the trade.
 *
 * @param id - unique identifier of the trade.
 */
    public void setId(int id) {
            this.id = id;
    }

    /**
 * Gets the date of the trade.
 *
 * @return date of the trade.
 */
    public Date getDate() {
            return date;
    }

    /**
 * Sets the date of the trade.
 *
 * @param date - date of the trade.
 */
    public void setDate(Date date) {
            this.date = date;
    }

    /**
 * Gets the price of the trade.
 *
 * @return price of the trade.
 */
    public int getPrice() {
            return price;
    }

    /**
 * Sets the price of the trade.
 *
 * @param price - price of the trade.
 */
```

```
    public void setPrice(int price) {
        this.price = price;
    }

    /**
 * Gets the buyer's login for the trade.
 *
 * @return buyer's login for the trade.
 */
    public String getBuyer() {
        return buyer;
    }

    /**
 * Sets the buyer's login for the trade.
 *
 * @param buyer - buyer's login for the trade.
 */
    public void setBuyer(String buyer) {
        this.buyer = buyer;
    }

    /**
 * Gets the name of the trade.
 *
 * @return name of the trade.
 */
    public String getName() {
        return name;
    }

    /**
 * Sets the name of the trade.
 *
 * @param name - name of the trade.
 */
    public void setName(String name) {
        this.name = name;
    }

    /**
 * Gets the current price of the trade.
 *
```

```
* @return current price of the trade.
*/
   public int getCurrentPrice() {
         return currentPrice;
   }

   /**
* Sets the current price of the trade.
*
* @param currentPrice - current price of the trade.
*/
   public void setCurrentPrice(int currentPrice) {
         this.currentPrice = currentPrice;
   }

   /**
* Gets the identifier of the good.
*
* @return identifier of the good.
*/
   public int getProductId() {
         return productId;
   }

   /**
* Sets the identifier of the good.
*
* @param productId - identifier of the good.
*/
   public void setProductId(int productId) {
         this.productId = productId;
   }

   /**
* Gets the description of the trade.
*
* @return description of the trade.
*/
   public String getDescription() {
         return description;
   }

   /**
```

```
    * Sets the description of the trade.
    *
    * @param description - description of the trade.
    */
       public void setDescription(String description) {
              this.description = description;
       }
}
```

5.4. User

```
/**
 * Class representing a user in the system.
 *
 * @author Verbickaya, Stepanova
 */
public class User {

       /** User's login. */
       private String login;

        /** User's password. */
       private String parole;

       /** User's type (admin, moder, user). */
       private UserTypeEnum type;

       /** User's online status (true if online, false otherwise). */
       private boolean onlineStatus;

       /** User's blocking status (true if blocked, false otherwise). */
   private boolean blockingStatus;

   /**
    * Default constructor.
    */
   public User() {

       }

   /**
    * Constructor with parameters.
    *
    * @param login - user's login.
```

```
    * @param parole - user's password.
    * @param type - user's type (admin, moder, user).
    * @param onlineStatus - user's online status (true if online, false otherwise).
    * @param blockingStatus - user's blocking status (true if blocked, false otherwise).
    */
      public User(String login, String parole, UserTypeEnum type, boolean onlineStatus, boolean
blockingStatus) {
            setLogin(login);
            setParole(parole);
            setType(type);
            setOnlineStatus(onlineStatus);
            setBlockingStatus(blockingStatus);
      }

      /**
    * Gets the user's login.
    *
    * @return user's login.
    */
      public String getLogin() {
            return login;
      }

       /**
    * Sets the user's login.
    *
    * @param login - user's login.
    */
      public void setLogin(String login) {
            this.login = login;
      }

      /**
    * Gets the user's type.
    *
    * @return user's type.
    */
      public UserTypeEnum getType() {
            return type;
      }

      /**
    * Sets the user's type.
```

```
*
* @param type - user's type.
*/
   public void setType(UserTypeEnum type) {
         this.type = type;
   }

   /**
* Gets the user's online status.
*
* @return user's online status.
*/
   public boolean isOnlineStatus() {
         return onlineStatus;
   }

   /**
* Sets the user's online status.
*
* @param onlineStatus - user's online status (true if online, false otherwise).
*/
   public void setOnlineStatus(boolean onlineStatus) {
         this.onlineStatus = onlineStatus;
   }

   /**
* Gets the user's blocking status.
*
* @return user's blocking status.
*/
   public boolean isBlockingStatus() {
         return blockingStatus;
   }

   /**
* Sets the user's blocking status.
*
* @param blockingStatus - user's blocking status (true if blocked, false otherwise).
*/
   public void setBlockingStatus(boolean blockingStatus) {
         this.blockingStatus = blockingStatus;
   }
```

```
    /**
   * Gets the user's password.
   *
   * @return user's password.
   */
    public String getParole() {
        return parole;
    }

    /**
   * Sets the user's password.
   *
   * @param parole - user's password.
   */
    public void setParole(String parole) {
        this.parole = parole;
    }
}
```

5.5. GoodDAO

```
/**
 * Interface representing Data Access Object for Good entities.
 * Defines methods to retrieve, update, and manage information about goods in the system.
 *
 * @author Verbickaya, Stepanova
 */
public interface GoodDAO {

    /**
   * Gets list of all goods available for sale.
   *
   * @return list of Good objects representing all selling goods.
   */
    List<Good> getAllSellingGoods();

    /**
   * Gets list of hot selling goods.
   *
   * @return list of Good objects representing hot selling goods.
   */
    List<Good> getSellingHotGoods();

    /**
```

```
* Gets list of selling goods in a specific category.
*
* @param filter - category by which to filter selling goods.
* @return list of Good objects representing selling goods in the specified category.
*/
   List<Good> getSellingGoodsByCategory(String filter);

   /**
* Gets list of goods being sold by a specific seller.
*
* @param login - login of the seller.
* @return list of Good objects representing goods being sold by the specified seller.
*/
   List<Good> getMySellingGoods(String login);

   /**
* Gets list of goods sold by a specific seller.
*
* @param login - login of the seller.
* @return list of Good objects representing goods sold by the specified seller.
*/
   List<Good> getMySoldGoods(String login);

   /**
* Gets list of goods removed by a specific seller.
*
* @param login - login of the seller.
* @return list of Good objects representing goods removed by the specified seller.
*/
   List<Good> getMyRemovedGoods(String login);

   /**
* Gets list of available categories for goods.
*
* @return list of category names.
*/
   List<String> getCategories();

   /**
* Gets information about a specific good by its identifier.
*
* @param id - identifier of the good.
* @return good.
```

```
*/
   Good getGood(String id);

   /**
* Increases the price of a specific good by a specific seller.
*
* @param id - identifier of the good.
* @param login - login of the buyer.
*/
   void raiseGoodPrice(String id, String login);

   /**
* Removes specific good from the system.
*
* @param id - identifier of the good.
*/
   void removeGood(String id);

    /**
 * Sets the description of a specific good.
 *
 * @param id - identifier of the good.
 * @param description - description for the good.
 */
   void setGoodDescription(String id, String description);

   /**
* Adds a new good.
*
* @param name - name of the good.
* @param description - description of the good.
* @param category - category of the good.
* @param price - starting price of the good.
* @param seller -seller's name for the good.
*/
   void addGood(String name, String description, String category, String price, String seller);

   /**
* Sets the constant of the price step and the selling interval.
*
* @param priceStep - price step for the good.
* @param sellingInterval - interval for selling the good.
*/
```

```
        void setConstants(String priceStep, String sellingInterval);

        /**
     * Gets the current price step setting for goods.
     *
     * @return current price step setting.
     */
        int getPriceStepSetting();

        /**
     * Gets the current selling interval setting for goods.
     *
     * @return current selling interval setting.
     */
        int getSellingIntervalSetting();

        /**
     * Sets the price step for a specific good.
     *
     * @param id - identifier for the good.
     * @param priceStep - price step for the good.
     */
        void setPriceStep(String id, String priceStep);

        /**
     * Sets the selling interval for a specific good.
     *
     * @param id - identifier for the good.
     * @param sellingInterval - interval for selling the good.
     */
        void setSellinInterval(String id, String sellingInterval);
}
```

5.6. TradeDAO

```
/**
 * Interface representing Data Access Object for Trade entities.
 * Defines methods for accessing and managing trade-related data in the database.
 *
 * @author Verbickaya, Stepanova
 */
public interface TradeDAO {

        /**
```

```
* Retrieves the buyer of a specific good identified by its identifier.
*
* @param goodId - identifier of the good.
* @return buyer's name.
*/
   String getBuyerOfGood(String goodId);

   /**
* Retrieves a list of all trades associated with a specific user login.
*
* @param login - user's login.
* @return list of all trades for the user.
*/
   List<Trade> getAllTrades(String login);

   /**
* Retrieves a list of successful trades associated with a specific user login.
*
* @param login - user's login.
* @return list of successful trades for the user.
*/
   List<Trade> getSuccessfulTrades(String login);

   /**
* Retrieves a list of unsuccessful trades associated with a specific user login.
*
* @param login - user's login.
* @return list of unsuccessful trades for the user.
*/
   List<Trade> getUnsuccessfulTrades(String login);

   /**
* Retrieves a specific trade identified by its identifier.
*
* @param id - identifier of the trade.
* @return trade object.
*/
   Trade getTrade(String id);

   /**
* Retrieves a list of identifiers for losing trades associated with a specific user login.
*
* @param login - user's login.
```

```
    * @return list of identifiers for losing trades.
    */
       List<Integer> getLosingTradesId(String login);

       /**
    * Retrieves a list of identifiers for winning trades associated with a specific user login.
    *
    * @param login - user's login.
    * @return list of identifiers for winning trades.
    */
       List<Integer> getWinningTradesId(String login);

       /**
    * Retrieves a list of identifiers for won trades associated with a specific user login.
    *
    * @param login - user's login.
    * @return list of identifiers for won trades.
    */
       List<Integer> getWinnedTradesId(String login);

       /**
    * Retrieves a list of identifiers for lost trades associated with a specific user login.
    *
    * @param login - user's login.
    * @return list of identifiers for lost trades.
    */
       List<Integer> getLosedTradesId(String login);
}
```

5.7. UserDAO

```
/**
 * Interface representing Data Access Object for User entities.
 * Defines methods to retrieve, update, and manage users information in the database.
 *
 * @author Verbickaya, Stepanova
 */
public interface UserDAO {

       /**
    * Gets user by their login.
    *
    * @param login - user's login.
    * @return user object representing the user with the specified login.
```

```
*/
   User getUser(String login);

   /**
* Gets list of all users.
*
* @return list of User objects representing all users in the system.
*/
   List<User> getAllUsers();

   /**
* Gets list of users with the admin type.
*
* @return list of User objects representing users with the admin type.
*/
   List<User> getAdminUsers();

   /**
* Gets list of users with the moder type.
*
* @return list of User objects representing users with the moder type.
*/
   List<User> getModerUsers();

   /**
* Gets list of users with the user type.
*
* @return list of User objects representing users with the user type.
*/
   List<User> getNormalUsers();

   /**
* Gets list of authorized users.
*
* @return list of User objects representing authorized users.
*/
   List<User> getAuthorizedUsers();

   /**
* Gets list of unauthorized users.
*
* @return list of User objects representing unauthorized users.
*/
```

```
    List<User> getUnauthorizedUsers();

    /**
  * Gets list of blocked users.
  *
  * @return list of User objects representing blocked users.
  */
    List<User> getBlockedUsers();

    /**
  * Gets list of unblocked users.
  *
  * @return list of User objects representing unblocked users.
  */
    List<User> getNotBlockedUsers();

    /**
     * Blocks the user.
     *
     * @param login - user's login.
     */
    void block(String login);

    /**
     * Unblocks the user.
     *
     * @param login - user's login.
     */
    void unblock(String login);

    /**
  * Changes the blocking status of a user.
  *
  * @param login - user's login.
  */
    void changeBlockingStatus(String login);

    /**
  * Adds a new user.
  *
  * @param login - user's login.
  * @param password - user's password.
  * @param type - user's type (admin, moder, user).
```

```
    */
      void addUser(String login, String password, String type);

      /**
    * Sets or updates a user's information.
    *
    * @param login - user's login.
    * @param newPassword - new user's password.
    * @param newBlockingStatus - new blocking status for the user.
    */
      void setUser(String login, String newPassword, String newBlockingStatus);

      /**
    * Deletes a user.
    *
    * @param login - user's login.
    */
      void deleteUser(String login);

      /**
    *  logs out the user of the system.
    *
    * @param login - user's login.
    */
      void logout(String login);

      /**
    * Logs user in the system
    *
    * @param login - user's login.
    */
      void login(String login);
}
```

6. Библиографический список

1. Блинов И. Н., Романчик В. С. Java. Методы программирования: учеб.-метод. пособие. - Минск: Четыре четверти, 2013. - 896 с.
2. Пруцков А. В., Сборник документов для учебных занятий 2020 года. - Рязань: РГРТУ, 2020 - 36 с. - №5500.