Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Онлайн сервис-аукцион»

Студент _____ИУ7-65Б_____ _____ _____Н.А.Шестовских___
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ ____П.В.Клорикьян_____
(Подпись, дата) (И.О.Фамилия)

2019 г.

Содержание

Введение

С повсеместным распространением интернета онлайн-сервисы стали неотъемлемой частью жизни современного человека. Большинство из нас пользуется электронной почтой, социальными сетями, видеохостингами. Онлайн-сервисы выполняют самые разные функции, но главная их цель — предоставить удобный интерфейс для выполнения той или иной задачи.

Целью данной курсовой работы является создание клиент-серверного приложения «онлайн-аукцион», которое позволяет размещать на сайте свой лот и участвовать в торгах.

Для достижения цели необходимо выполнить следующие задачи:

- формализовать задачу для определения необходимого функционала;
- провести анализ видов СУБД, языков программирования, фреймворков для определения технологического стека курсовой работы;
- спроектировать и реализовать базу данных для хранения и структурирования информации о лотах и о пользователях;
- спроектировать и реализовать прилоложение для взаимодействия пользователей с базой данных.

1. Аналитический раздел

В данном разделе будет представлена формализация задачи, анализ СУБД и предметной области.

1.1 Формализация задачи

В соответствие с ТЗ требуется разработать пользовательское веб-приложение для проведения интернет-аукциона. Пользователю доступен список активных лотов, при авторизации он может участвовать в торгах по любому из них. Также он может разместить свой лот. Торги организуются с повышением цены. Пока не истек таймер, пользователь может поднять ставку. При поднятии ставки таймер сбрасывается. Победителем в торге объявляется тот, чья ставка была последней, и при этом истек таймер. Требуется создать базу данных для хранения информации о лотах и о пользователях, также предоставить пользователю возможность регистрации, авторизации и выхода из аккаунта.

1.2 Веб-приложения

Веб-приложение — это клиент-серверное приложение, основная часть которого содержится на удаленном сервере, а пользовательский интерфейс (UI) отображается в браузере в виде веб-страниц.

Для запуска веб-приложения пользователю не нужно устанавливать никаких дополнительных программ, оно запускается на любом устройстве с браузером и доступом в интернет.

В клиентской части веб-приложений используются языки HTML и CSS для описания структуры веб-документов и оформления их внешнего вида. Серверная часть не накладывает жестких ограничений на используемые языки.

1.3 Анализ предметной области.

Аукционы - это специально организованные действующие рынки, на которых путем публичных торгов в заранее установленное время и в специально назначенном месте производится продажа [1].

Порядок проведения аукциона включает в себя:

- подготовительные операции;
- операции, непосредственно связанные с проведением торгов;
- операции, связанные с взаимными расчетами между организаторами и участниками торгов (оформление аукционных сделок).

Товары сортируют в зависимости от качества по партиям, которые носят название лотов.

Лот - это стандартная партия или типовая единица товара в натуральном выражении. Из каждого лота отбирается образец. Лот может содержать одно изделие или несколько единиц (соболь — 30-50 шкурок, норка — до 300 шкурок, каракуль — до 500 шкурок). Каждому лоту присваивается номер, по которому он будет выставлен на аукцион для продажи.

Электронные торги – это способ выбора поставщика (или покупателя), при котором процесс покупки и продажи совершается на специализированных сайтах — электронных торговых площадках в интернете. Покупать и продавать можно товары, работы или услуги.

В электронных торгах участвуют две стороны:

- **Организатор электронных торгов (заказчик)** формирует свой заказ, в котором должны быть указаны требования к поставке товаров или услуг, веб-приложеникцена и сроки проведения торгов. Далее заказчик размещает его на одной из электронных торговых площадок.
- **Участники размещения заказа (поставщики)** подают заявки на выполнение заказа организатора и вносят свои ценовые предложения.

1.4 Общие сведения о базах данных

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе [2]. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также

приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

Данные в наиболее распространенных типах современных баз данных обычно формируются в виде строк и столбцов в ряде таблиц, чтобы обеспечить эффективность обработки и запросов данных. Затем можно легко получать доступ к данным, управлять ими, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL).

1.5 Классификация СУБД

Существует множество различных типов баз данных.

- **Реляционные базы данных.** Реляционные базы данных стали преобладать в 1980-х годах. Элементы в реляционной базе данных организованы в виде набора таблиц со столбцами и строками. Технология реляционных баз данных обеспечивает наиболее эффективный и гибкий способ доступа к структурированной информации.
- **Объектно-ориентированные базы данных.** Информация в объектно-ориентированной базе данных представлена в форме объекта, как в объектно-ориентированном программировании.
- **Распределенные базы данных.** Распределенная база данных состоит из двух или более файлов, расположенных на разных узлах. Такая база данных может храниться на нескольких компьютерах, расположенных в одном физическом месте или распределенных по разным сетям.
- **Хранилища данных.** Будучи централизованным репозиторием для данных, хранилище данных представляет собой тип базы данных, специально предназначенной для быстрого выполнения запросов и анализа.
- **Базы данных NoSQL.** База данных NoSQL, или нереляционная база данных, позволяет хранить и обрабатывать неструктурированные или слабоструктурированные данные (в отличие от реляционной базы данных,

задающей структуру содержащихся в ней данных). Популярность баз данных NoSQL растет по мере распространения и усложнения веб-приложений.

- **Графовые базы данных.** Графовая база данных хранит данные в контексте сущностей и связей между сущностями.
- **Базы данных OLTP.** База данных OLTP — это быстрая база данных аналитического типа, предназначенная для большого объема транзакций, выполняемых множеством пользователей.

Самой интересной моделью

1.6 Выделение сущностей и связей приложения

В соответствие с анализом предметной области можно выделить две основные сущности: лот — выставляемая на аукционе единица, цель торгов и пользователь — тот, кто выставляет лот либо участвует в торгах. Также можно выделить сущность категория — она позволит пользователю фильтровать лоты для поиска тех, которые интересуют именно его.

У любого лота есть тот, кто его выставляет. Назовем его владельцем. Также у каждого лота есть пользователь, который поставил на данный момент самую выгодную сумму, то есть потенциальный победитель торгов. Назовем его текущий покупатель. Таким образом, между таблицей пользователей и таблицей лотов есть две независимые связи типа один ко многим — намерение купить и владение лотом. Между категорией и лотом наблюдается связь один ко многим, так как лотов в одной категории может быть сколько угодно, но категория у лота — только одна.

1.7 Диаграмма сущностей-связей

На рисунке 1.1 приведена ER-диаграмма приложения.

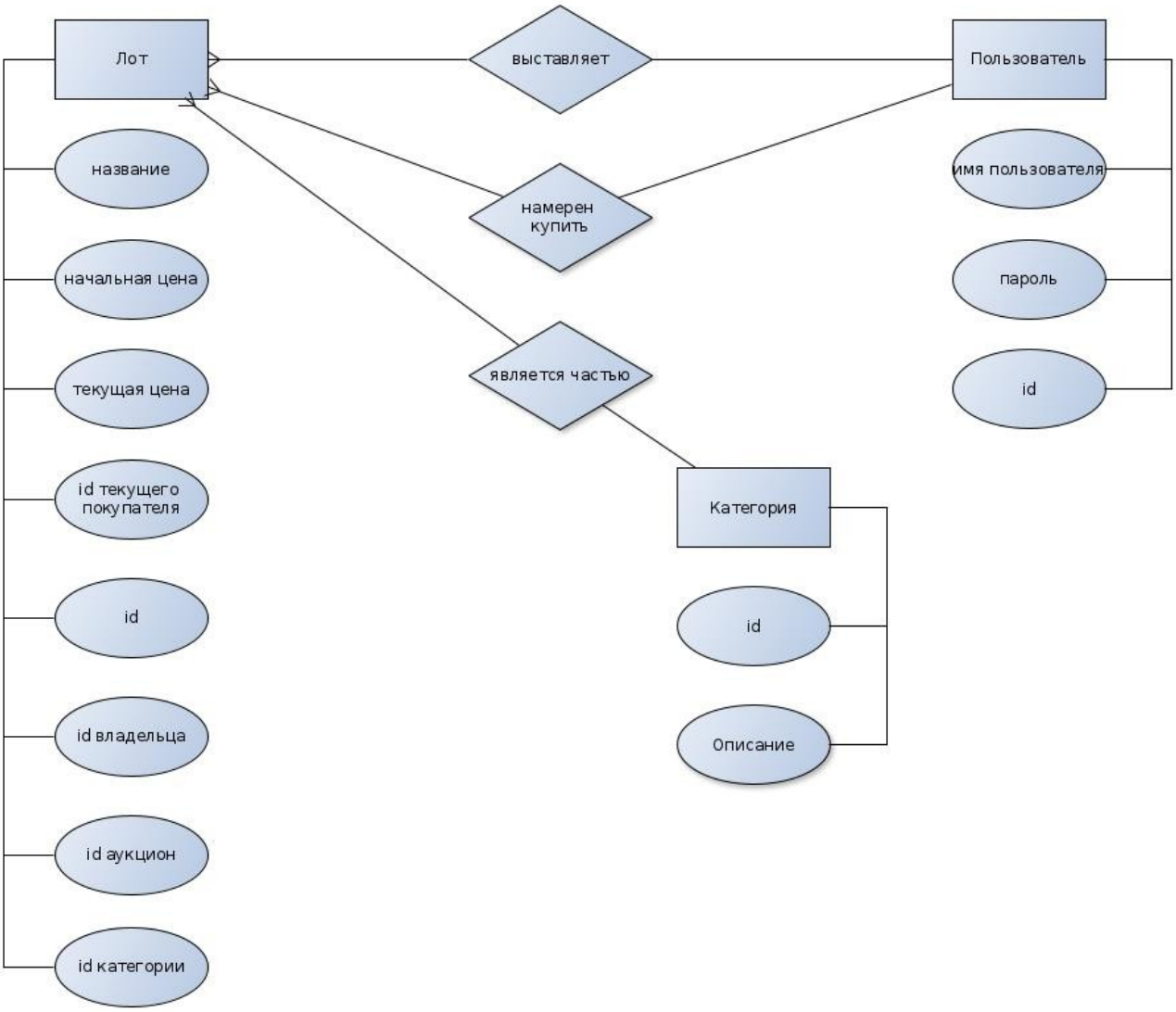

Рис. 1.1 ER-диаграмма базы данных

Вывод.

В данном разделе была произведена формализация задачи, приведена классификация СУБД для дальнейшего выбора одной из них для данной работы, проанализирована предметная область и в связи с ней выделены сущности и связи между ними. Хотя с точки зрения предметной области

пользователь-покупатель и пользователь-продавец являются разными субъектами, однако с точки зрения системы один и тот же пользователь должен иметь право как выставлять лоты на продажу, так и участвовать в торгах.

2. Конструкторский раздел

В данном разделе будет приведена архитектура приложения.

2.1 Паттерн модлель-представление-контроллер

Паттерн MVC широко используется для веб-приложений [3]. Этот шаблон разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты. Схема паттерна приведена на рисунке 2.1

В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

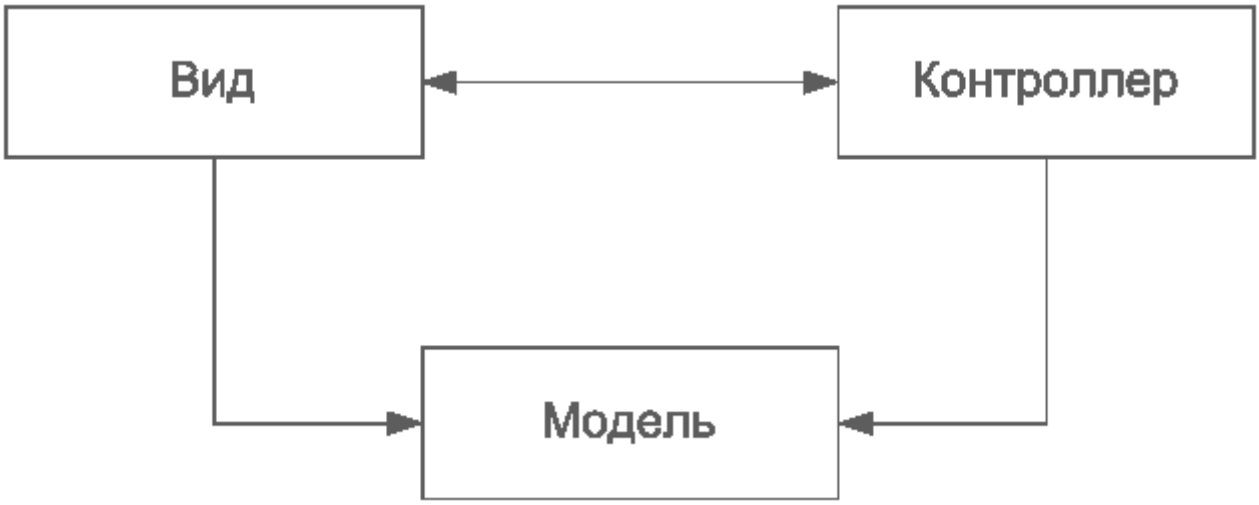

Рис. 2.1 схема паттерна модель-представление-контроллер

2.2 Модель

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими. В случае с приложениями, использующих БД для хранения информации модели — это прямые отображения таблиц баз данных для предоставления доступа данным из других компонентов системы.

В связи с этим можно выделить три модели:

- пользователь;
- категория;
- лот.

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) — технология программирования [4], которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

ORM позволяет удобно интегрировать модели в приложения с объектно-ориентированным стилем программирования. В данной работе эта технология будет использоваться.

2.3 Вид (представление)

В концепции MVC представление — это компонент системы, нужный для отображения пользователю модели. По сути представление — это пользовательский интерфейс. В случае веб-приложений представление — это HTML — страница, с помощью которой клиент взаимодействует с приложением.

В данной работе можно выделить следующие представления:

- Страница со списком лотов (LotList);
- Страница для участия в торгах (LotDetail);
- Различные представления пользователей — страницы с регистрацией и авторизацией.

2.1 Контроллер

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

В данной работе контроллеры обеспечивают обработку HTTP-запросов GET и POST, а также предоставляют данные о состоянии моделей либо же вносят изменения в них. Контроллер есть у каждого представления, так как именно он отправляет пользователю запрашиваемую HTML — страницу.

Список контроллеров и задач, которые они выполняют:

1. LotlList:
   - по GET-запросу возвращает список всех лотов;
   - по POST-запросу добавляет новый лот в таблицу.
2. LotDetail:
   - по GET-запросу возвращает информацию о лоте и форму для поднятия цены в зависимости от пользователя (если пользователь — владелец лота, то он не может участвовать в торгах);
   - по POST-запросу отправляет значение, на которое пользователь повысил цену.
3. Register:
   - по GET-запросу возвращает форму для создания нового пользователя;
   - по POST-запросу проверяет правильность введенных данных и в случае успеха добавляет пользователя.
4. Login:

- по GET-запросу возвращает форму для авторизации;
- по POST-запросу проверяет правильность введенных логина и пароля.

5. PersonalOffice:
   - по GET-запросу возвращает основную информацию о пользователе — выставленные и пробретенные лоты.

Вывод

В данном разделе была приведена архитектура приложения.

3. Технологический раздел

В данном разделе будут приведены примеры реализации представления, модели, шаблона, представлена диаграмма БД и диаграмма классов, а также приведены скриншоты с интерфейсом приложения.

3.1 Выбор технологического стека.

В качестве языка программирования был выбран python [5] из-за совмещения нескольких парадигм программирования, совмещенных в нем. Также python обладает большим количеством фреймворков и библиотек, в том числе для доступа к различным СУБД. Помимо этого, python — интерпретируемый язык, что позволяет не заниматься перекомпоновкой всего проект для обновления того или иного компонента.

Фреймворком для веб-приложения был выбран django [6], так как он поддерживает основные СУБД. Среди них:

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

Помимо этого, django построен вокруг паттерна MTV (модель-шаблон-представление). Этот паттерн представляет из себя то же самое, что и MVC с точностью до переименования компонентов (шаблон в MTV это представление в MVC, то же самое с представлением и контроллером), и имеет базовые классы для моделей и контроллеров, что упрощает разработку на этом фреймворке.

В качестве СУБД был выбран PostgreSQL, так как он поддерживается django, помимо этого он обладает механизмом наследования, что позволит в дальнейшем масштабировать проект.

3.2 Диаграмма базы данных.

Фреймворк django имеет свою модель для хранения данных о пользователе, так же как и стандартные представления для авторизации, регистрации и выхода из аккаунта. Чтобы добавить новые поля, была добавлена новая модель профиль, содержащая недостающие поля и связанная со стандартной моделью пользователя отношением один-к-одному. На рисунке 3.1 представлена диаграмма БД.

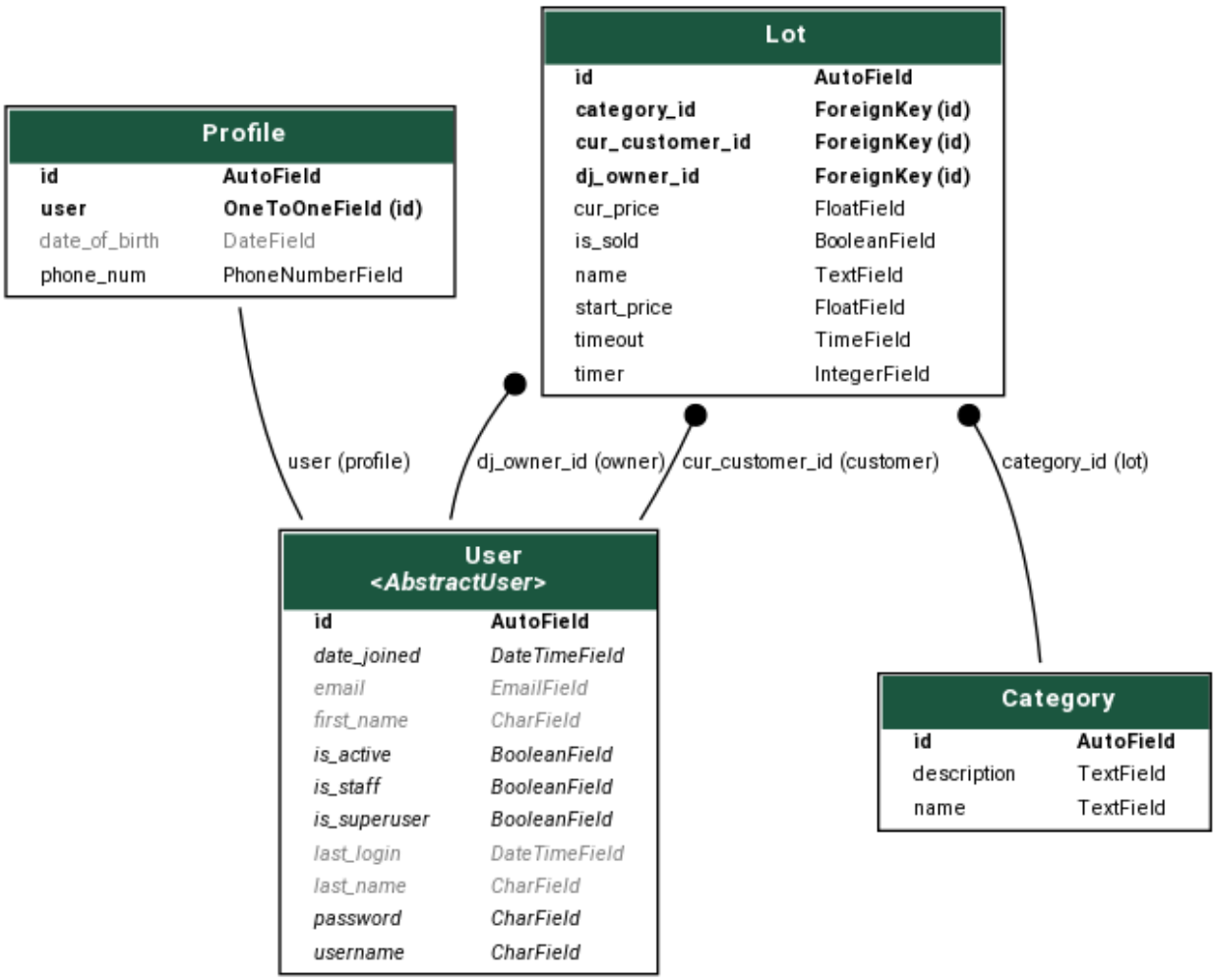

Рис. 3.1: диаграмма БД

3.3 Диаграмма классов

На рисунке 3.2 представлена диаграмма классов приложения.

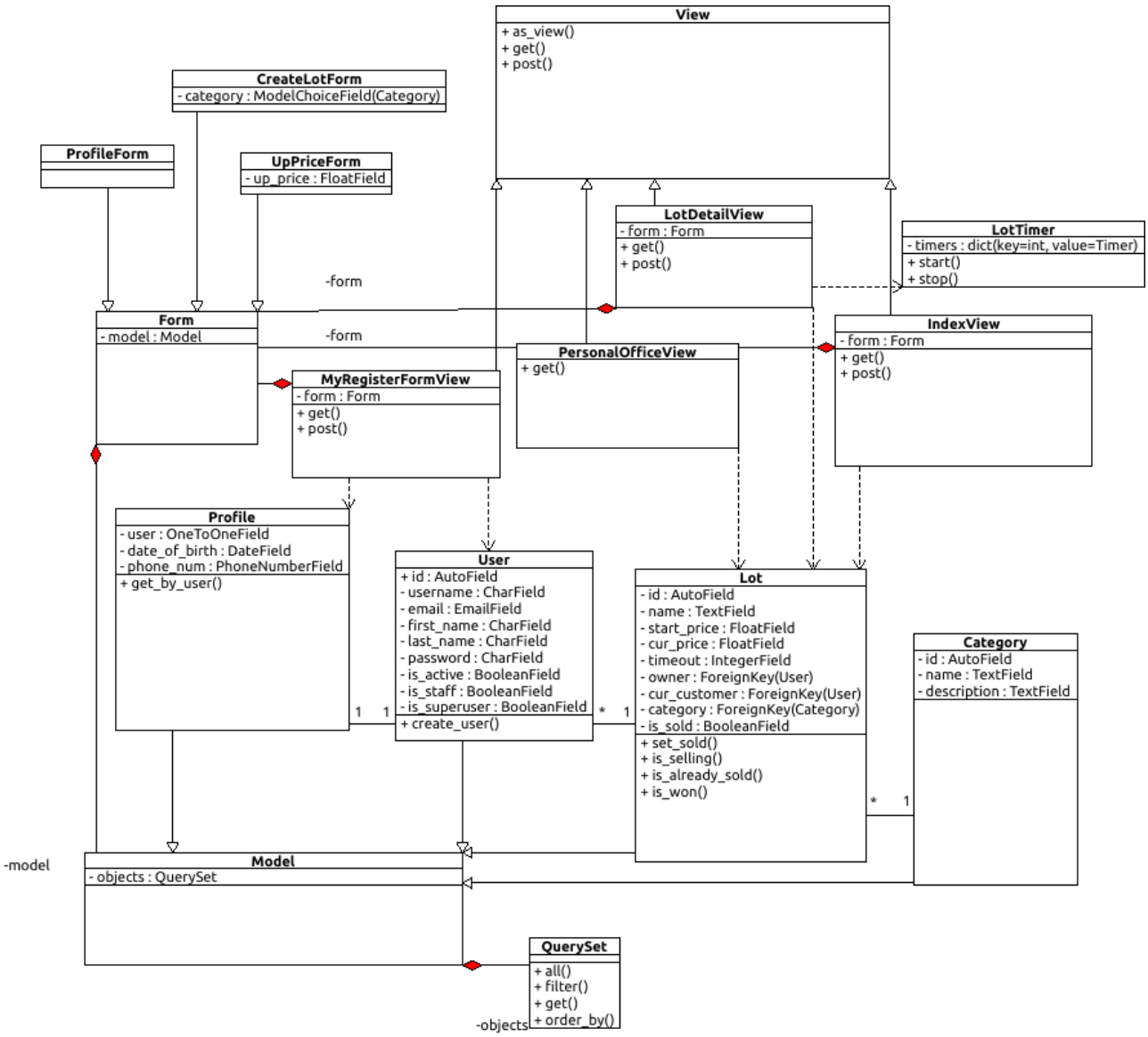

Рис. 3.2: диаграмма классов приложения

3.4 Реализация представления(шаблона)

Django предоставляет стандартный API для загрузки и рендеринга шаблонов, незавимисо от используемого бэкенда. Загрузка включает в себя поиск шаблона по названию и предварительную обработку, обычно выполняется загрузка шаблона в память. Рендеринг означает передачу данных контекста в шаблон и возвращение строки с результатом. Пример шаблона представлен на листинге 3.1.

Листинг 3.1: шаблон личного кабинета

```
1. <html>
2.     <head>
3.         <title>Личный кабинет</title>
4.     </head>
5.     <body>
6.                 {% if user.is_authenticated %}
7.                         <li>Пользователь: {{ user.get_username }}</li>
8.                         <li><a href="{% url 'logout'%}?next={{request.path}}">Выйти из
    аккаунта</a></li>
9.                 {% else %}
10.                        <li><a href="{% url
    'login'%}?next={{request.path}}">Войти</a></li>
11.                        <li><a href=/accounts/register>Регистрация</a></li>
12.                {% endif %}
13.
14.                <div><h2>Список Ваших лотов, по которым идут торги</h2></div>
15.            {% for lot in lots_selling %}
16.                        <h3><a href="/lot/{{lot.id}}">{{ lot.name }}</a></h3>
17.                {% endfor %}
18.                <div><h2>Список Ваших лотов, по которым определился победитель</
    h2></div>
19.            {% for lot in lots_sold %}
20.                        <h3><a href="/lot/{{lot.id}}">{{ lot.name }}</a></h3>
21.                {% endfor %}
22.                <div><h2>Список лотов, выигранных Вами</h2></div>
23.            {% for lot in lots_won %}
24.                        <h3><a href="/lot/{{lot.id}}">{{ lot.name }}</a></h3>
25.                {% endfor %}
26.    </body>
27. </html>
```

3.4. Реализация моделей

Django обладает функционалом ORM для интеграции СУБД приложениями в объектно-ориентированном стиле. Для использования этого функционала можно отнаследовать свой класс модели от базового класса Model, который предоставляет интерфейс для взаимодействия со всеми поддерживаемыми jango СУБД. Пример такой модели приведен на листинге 3.2.

Листинг 3.2: модель лот

```
1. class Lot(models.Model):
2.     id = models.AutoField(primary_key=True)
3.     name = models.TextField()
```

```
    start_price = models.FloatField(validators=[MinValueValidator(0.01)])
    cur_price = models.FloatField(null=True)

    timeout = models.TimeField(null = True)
    timer = models.IntegerField(null = True)

    dj_owner_id = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        null=True,
        related_name="owner"
    )
    cur_customer_id = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        null=True,
        on_delete=models.SET_NULL,
        related_name="customer"
    )
    category_id = models.ForeignKey('Category', on_delete=models.SET_NULL,
null=True)

    is_sold = models.BooleanField(default = False, null=True)

    def __str__(self):
        return str(self.name)

    def set_sold(self):
        self.is_sold = True
        self.save()

    def is_selling(self, user):
        print(user, self.dj_owner_id, self.is_sold == False)
        return user == self.dj_owner_id and self.is_sold == False

    def is_already_sold(self, user):
        print(user, self.dj_owner_id, self.is_sold == True)
        return user == self.dj_owner_id and self.is_sold == True

    def is_won(self, user):
        print(user, self.cur_customer_id, self.is_sold == True)
        return user == self.cur_customer_id and self.is_sold == True
```

3.6 Реализация контроллеров

Как было сказано в разделе 2.3 контроллер представляет из себя обработчик HTTP запросов типа GET и POST, в django реализуется это путем наследования от базового класса View и переопределения методов get() и post(). Пример контроллера представлен на листинге 3.3.

Листинг 3.3: контроллер для страницы регистрации.

```
1.  class MyRegisterView(View):
2.      def get(self, request, *args, **kwargs):
3.          user_form = UserForm()
4.          profile_form = ProfileForm()
5.          return render(request, 'registration/registration.html', {'user_form': user_form,
    'profile_form': profile_form})
6.
7.      def post(self, request, *args, **kwargs):
8.          user_form = UserForm(request.POST)
9.          profile_form = ProfileForm(request.POST)
10.
11.         if user_form.is_valid() and profile_form.is_valid():
12.             data = user_form.cleaned_data
13.             data['password'] = data['password1']
14.             data.pop('password1')
15.             data.pop('password2')
16.             new_user = User.objects.create_user(**user_form.cleaned_data)
17.             login(request,new_user)
18.             profile_data = profile_form.cleaned_data
19.             new_profile = Profile.objects.create(
20.                     phone_num=profile_data['phone_num'],
21.                     date_of_birth=profile_data['date_of_birth'],
22.                     user = new_user,
23.             )
24.             return HttpResponseRedirect('/')
25.         return HttpResponseRedirect('accounts/register/')
```

3.7 Реализация обратного отсчета

С точки зрения интернет-торгов важным аспектом является реализация обратного отсчета до выявления победителя. Одновременно могут проводится несколько торгов, и при этом необходимо не нагружать поток, обрабатывающий HTTP-запросы. Поэтому был использован класс threading.Timer, запускающий обратный отсчет в отдельном потоке. Был создан класс, предоставляющий доступ к ассоциативному массиву, состоящему из объектов типа timer. Метод start(), принимающий в качестве параметра id лота, по которому ведутся торги, запускает таймер, по истечении которого лот отмечается как проданный. Метод stop(), принимающий в качестве параметра id лота, останавливает текущий таймер. Реализация класса LotTimer представлена на листинге 3.4. Класс был разработан с помощью паттерна одиночка, так как объект этого класса на все приложение один.

Листинг 3.4: реализация обратного отсчета в приложении.

```
1.  class Singleton(type):
2.      _instances = {}
3.      def __call__(cls, *args, **kwargs):
4.          if cls not in cls._instances:
5.              cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
6.          return cls._instances[cls]
7.
8.  class LotTimer(metaclass=Singleton):
9.              timers = dict()
10.             def start(self, lot_id):
11.                     try:
12.                             timer = self.timers[lot_id]
13.                     except KeyError:
14.                             lot = Lot.objects.get(pk = lot_id)
15.                             self.timers[lot_id] = Timer(float(str(lot.timer)), lambda:
    Lot.sold(lot_id))
16.                             timer = self.timers[lot_id]
17.                     else:
18.                             self.timers.pop(lot_id)
19.                             lot = Lot.objects.get(pk = lot_id)
20.                             self.timers[lot_id] = Timer(float(str(lot.timer)), lambda:
    Lot.sold(lot_id))
21.                             timer = self.timers[lot_id]
22.                     finally:
23.                             timer.start()
24.
25.             def stop(self, lot_id):
26.                     try:
27.                             timer = self.timers[lot_id]
28.                             timer.cancel()
29.                     except KeyError:
30.                             lot = Lot.objects.get(pk = lot_id)
31.                             self.timers[lot_id] = Timer(float(str(lot.timer)), lambda:
    Lot.sold(lot_id))
```

3.8 Примеры интерфейса

На рисунках 3.3 — 3.8 представлены примеры интерфейса программы.

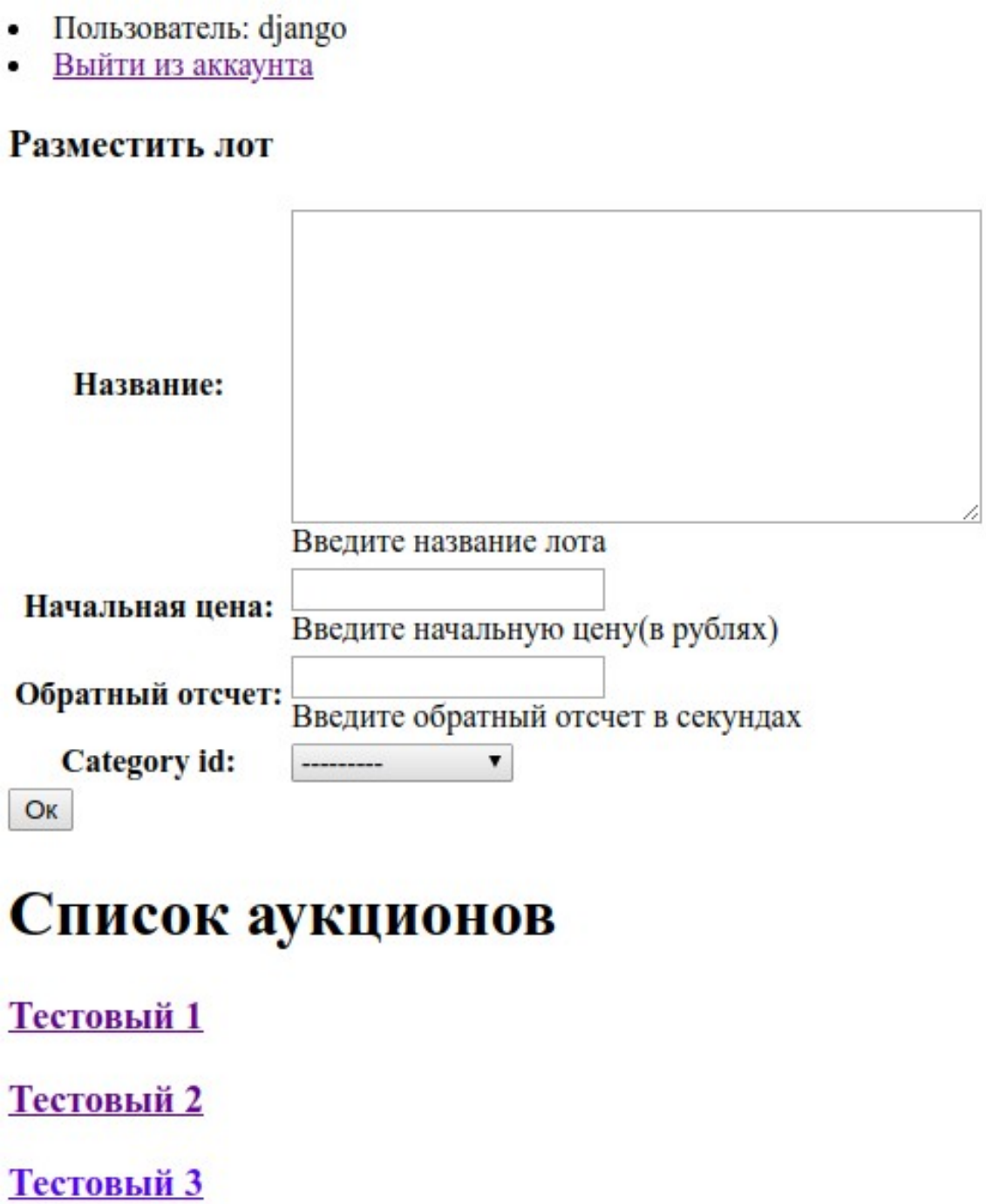

Рис. 3.3 — главная страница

-

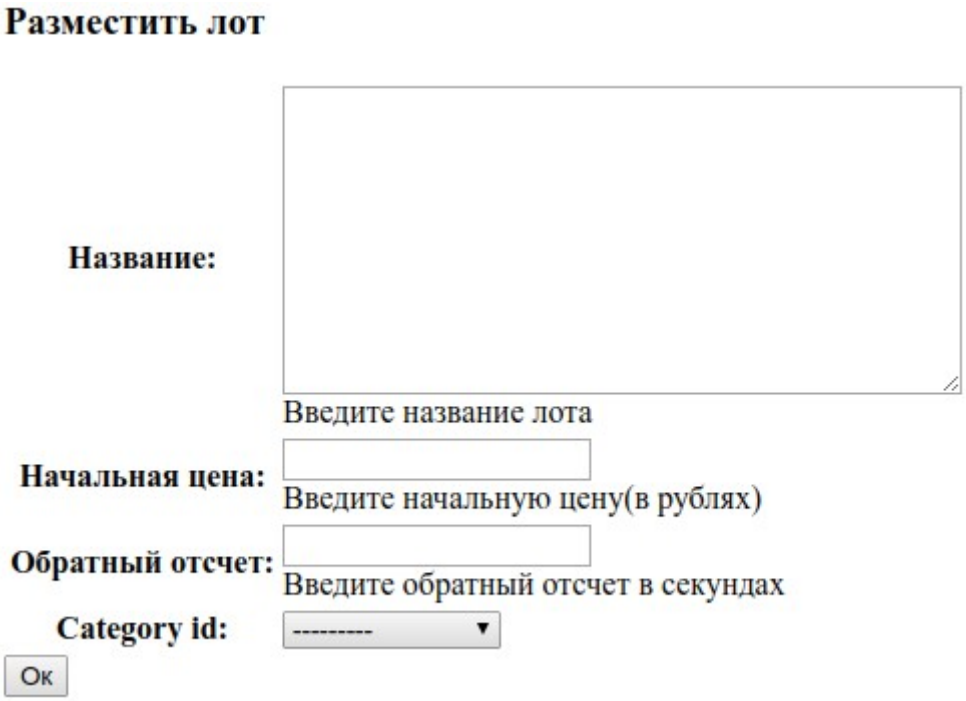

Рис. 3.4 — форма для размещения лота

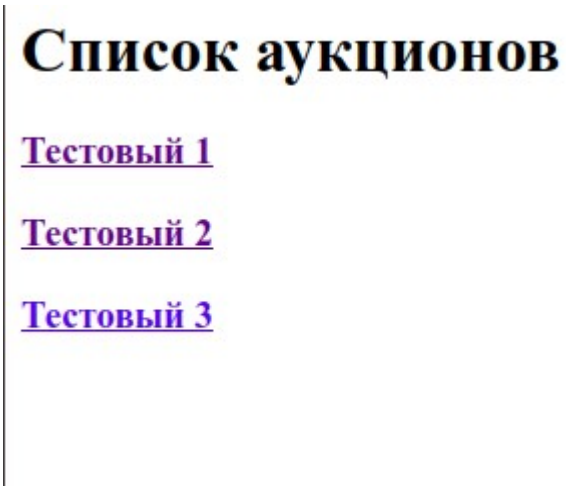

Рис. 3.5 — список аукционов

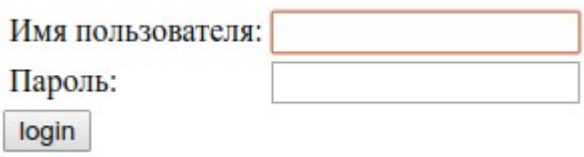

Рис. 3.6 — форма для авторизации

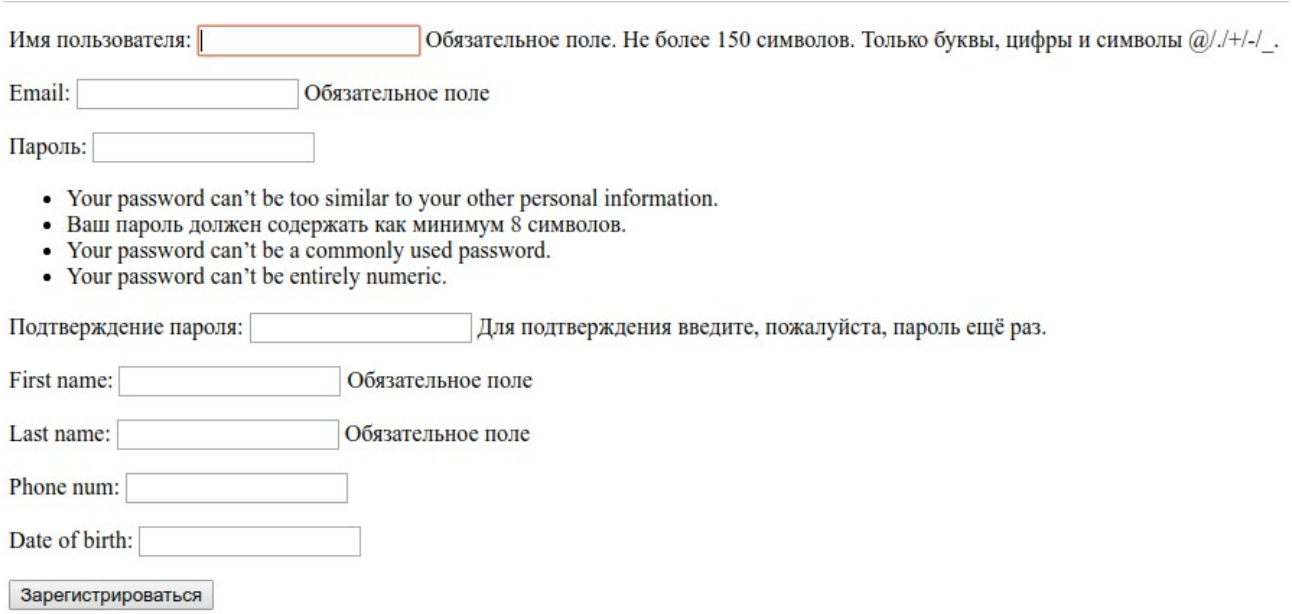

Рис. 3.7 — форма для регистрации пользователя

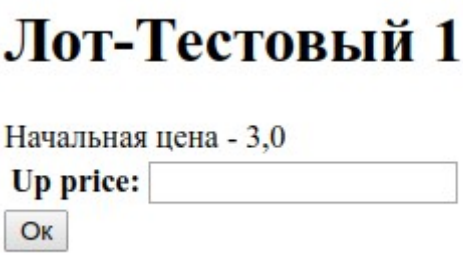

Рис. 3.8 — страница с участием в торгах

Вывод

В данном разделе были представлены примеры реализации модели, представления и контроллера, описана и показана реализация обратного отсчета, а также предоставлены скриншоты элементов интерфейса.

Заключение

В данной работе были выполнены следующие задачи:

- формализована задача по разработке интернет-аукциона;
- проведен анализ видов СУБД, языков программирования, фреймворков для определения технологического стека курсовой работы;
- спроектирована и реализована база данных для хранения и структурирования информации о лотах и о пользователях
- спроектировано и реализовано веб-приложение для взаимодействия пользователей с базой данных.

В качестве языка программирования был выбран python, фреймворк — django, СУБД - объектно-реляционная PostgreSQL. База данных состоит из четырех таблиц, среди которых: таблица пользователей, лотов, профилей и категорий. Веб-приложение спроектировано по паттерну модель-представление-контроллер, который на данный момент является самым используемым при разработке сайтов.

Разработанное приложение позволяет пользователю выставить лот и участвовать в торгах. На данный момент предусмотрены торги на повышение цены со скрытым от пользователя обратным отсчетом. Пользователь может зарегистрироваться, авторизироваться и выйти из аккаунта. Для просмотра результатов торгов в приложение был добавлен личный кабинет, где пользователь может узнать о победителе лота, выставленного им.

Исходя из анализа предметной области система в ходе дальнейшей разработки должна быть расширена для проведения онлайн-торгов различного характера, а не только классических аукционов. Для этого потребуется добавление новой сущности — тип электронных торгов, в которой будут

перечислены основные характеристики розыгрыша: торги идут на повышение или понижение, знают ли участники торгов о других претендентах на лот, видят ли пользователи обратный отсчет. Таким образом получится покрыть большинство видов интернет-аукционов. Однако эта система не может быть использована для биржевых торгов, так как в этом типе торгов самый важный механизм — механизм ценообразования, который не предусмотрен в настоящей работе. На данный момент не введена система верификации пользователей, которая бы позволила выставлять какой-либо лот только подтвержденным продавцам. С точки зрения интерфейса приложение не обладает поиском и фильтрацией лотов на главной странице, что может усложнить навигацию пользователя при выборе интересующего лота.

Тем не менее, все поставленные в техническом задании задачи были выполнены, а архитектура готова к возможному масштабированию.

Список использованных источников

1. База данных [Электронный ресурс]. -

   Режим доступа:

   https://www.oracle.com/ru/database/what-is-database.html (дата обращения - 20.05.2020)

2. Аукционы и аукционные торги. Формы проведения аукционов [Электронный ресурс]. -

   Режим доступа: http://www.grandars.ru/college/biznes/aukcion.html (дата обращения - 23.05.2020)

3. Электронные аукционы [Электронный ресурс]. -

   Режим доступа: - http://www.urdc.ru/elektronnye-aukciony (дата обращения - 20.05.2020)

4. Паттерн MVC [Электронный ресурс]. -

   Режим доступа: - http://design-pattern.ru/patterns/mvc.html (дата обращения - 20.05.2020)

5. ORM [Электронный ресурс]. -

   Режим доступа: - https://flexberry.github.io/ru/gbt_orm.html (дата обращения - 25.05.2020)

6. Python documentation [Электронный ресурс]. -

   Режим доступа: https://www.python.org/doc/ (дата обращения - 10.05.2020)

7. Django documentation [Электронный ресурс]. -

   Режим доступа https://docs.djangoproject.com/en/3.0/ (дата обращения - 20.05.2020)