

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту  
на тему

«Каталог игровых услуг»

Студент гр. 753501

Киселёв А.А.

Руководитель

Удовин И.А.

Минск 2020

# Содержание

<b>Введение</b>	<b>3</b>
<b>Анализ предметной области</b>	<b>4</b>
Теоретические сведения	4
Технологические средства разработки	6
Модули программ и их описание	11
<b>Реализация базы данных в приложении</b>	<b>13</b>
Подключение БД	13
Описание и использование моделей для базы данных	14
Формы в Django	18
Авторизация/Регистарция	20
Создание заявки на услугу	23
<b>Заключение</b>	<b>26</b>
<b>Список использованной литературы</b>	<b>27</b>

# Введение

Согласно отчётам мировых маркетинговых агентств, в частности Global Games Market Report 2020, рост мировой игровой индустрии продолжится как минимум до 2023 года. Выводы исследовательского анализа сообщают, что к концу текущего года объём игровой аудитории по всему миру вырастет до 2,7 млрд человек, а к 2023-му перевалит за отметку 3 млрд игроков.

Учитывая актуальность этого вопроса, а также особый интерес для меня лично, темой своей работы я выбрал создание сервиса, который будет предоставлять каталог игровых услуг. Это своего рода площадка для встречи продавцов и покупателей услуг в определённой игровой тематике.

# 1. Анализ предметной области

## 1.1. Теоретические сведения

**База данных** — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. 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.

Для базы данных обычно требуется комплексное программное обеспечение, которое называется системой управления базами данных (СУБД). СУБД служит интерфейсом между базой данных и пользователями или программами, предоставляя пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией. СУБД также упрощает контроль и управление базами данных, позволяя выполнять различные административные операции, такие как мониторинг производительности, настройка, а также резервное копирование и восстановление. В качестве примеров популярного программного обеспечения для управления базами данных, или СУБД, можно назвать MySQL, Microsoft Access, Microsoft SQL Server, Oracle Database и dBASE.

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).

**SQL** — это язык программирования, используемый в большинстве реляционных баз данных для запроса, обработки и определения данных, а также контроля доступа. SQL был впервые разработан в IBM в 1970-х годах,

и Oracle выступил в качестве основного участника, что привело к внедрению стандарта SQL ANSI. SQL дал толчок выпуску многочисленных расширений от таких компаний, как IBM, Oracle и Microsoft. Хотя в настоящее время SQL все еще широко используется, начали появляться новые языки программирования.

Существует множество различных типов баз данных. Выбор наилучшей базы данных для конкретной организации зависит от того, как эта организация намеревается использовать данные.

**Реляционные базы данных** – старейший тип до сих пор широко используемых БД общего назначения. Данные и связи между данными организованы с помощью таблиц. Каждый столбец в таблице имеет имя и тип. Каждая строка представляет отдельную запись или элемент данных в таблице, который содержит значения для каждого из столбцов.

**NoSQL база данных** — это база данных, в которой в отличие от большинства традиционных систем баз данных не используется табличная схема строк и столбцов. В этих базах данных применяется модель хранения, оптимизированная под конкретные требования типа хранимых данных. Например, данные могут храниться как простые пары "ключ — значение", документы JSON или граф, состоящий из ребер и вершин.

## 1.2. Технологические средства разработки

**Visual Studio Code** — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией. Visual Studio Code основан на Electron и реализуется через веб-редактор Monaco, разработанный для Visual Studio Online (см. рис. 1).



Рис. 1 - Логотип VSCode

**Python** – это универсальный современный ЯП высокого уровня, к преимуществам которого относят высокую производительность программных решений и структурированный, хорошо читаемый код. Синтаксис Python максимально облегчен, что позволяет выучить его за сравнительно короткое время. Ядро имеет очень удобную структуру, а широкий перечень встроенных библиотек позволяет применять внушительный набор полезных функций и возможностей. ЯП может использоваться для написания прикладных приложений, а также разработки WEB-сервисов. Python может поддерживать широкий перечень стилей

разработки приложений, в том числе, очень удобен для работы с ООП и функционального программирования (см. рис. 2).



Рис. 2 - Логотип Python

**Django** — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation(см. рис. 3).

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других (например, Ruby on Rails). Один из основных принципов фреймворка — DRY (англ. *Don't repeat yourself*). Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.



Рис. 3 - Логотип Django

**HTML5** (HyperText Markup Language) — язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта (англ. HTML Living Standard). Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров (см. рис. 4).



Рис. 4 - Логотип HTML5

**CSS (Cascading Style Sheets)** — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL. Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов (см. рис. 5).

CSS поддерживает таблицы стилей для конкретных носителей, поэтому авторы могут адаптировать представление своих документов к визуальным браузерам, слуховым устройствам, принтерам, брайлевским устройствам, карманным устройствам и т.д.





Рис. 5 - Логотип CSS3

**PostgreSQL** — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности. СУБД отличается высокой надежностью и хорошей производительностью. PostgreSQL поддерживает транзакции (ACID), репликация реализована встроенными механизмами. При этом система расширяемая — можно создавать свои типы данных и индексов, а также расширять поведение при помощи языков программирования.

Из дополнительных возможностей PostgreSQL хочется отдельно отметить тот факт, что эта СУБД позволяет работать не только со структурированными и нормализованными, но и со слабоструктурированными данными (в форматах json / jsonb), при этом эти данные индексируются и работа с ними действительно удобна(см. рис. 6).

### **Особенности PostgreSQL:**

**Функции** в PostgreSQL являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки собственно SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием различных языков программирования. PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса.

**Многоверсионность** поддерживается в PostgreSQL — возможна одновременная модификация БД несколькими пользователями с помощью механизма Multiversion Concurrency Control (MVCC). Благодаря этому соблюдаются требования ACID, и практически отпадает нужда в блокировках чтения.

**Расширение** PostgreSQL для собственных нужд возможно практически в любом аспекте. Есть возможность добавлять собственные преобразования типов, типы данных, домены (пользовательские типы с изначально наложенными ограничениями), функции (включая агрегатные), индексы, операторы (включая переопределение уже существующих) и процедурные языки.

**Наследование** в PostgreSQL реализовано на уровне таблиц. Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.



Рис. 6 - Логотип PostgreSQL

### 1.3. Модули программ и их описание

В рамках исследования эффективности различных подходов к созданию двух баз данных в данном курсовом проекте реализовано несколько модулей, каждый из которых выполняет определенную задачу:

- **Клиент**, который представлен блоками кода HTML, CSS, JS и необходим для выполнения задач отрисовки элементов в браузере пользователя. Также выполняет функцию взаимодействия с пользователем, включая запись определенных полей в БД.
- **Локальный Сервер**, который служит для решения задач, связанных с MySQL DB.

Кроме того, в состав проекта вошло множество файлов, которые отвечают за различные операции в программе. Рассмотрим основные из них:

- **settings.py** содержит в себе все настройки проекта. Здесь мы регистрируем приложения, задаём размещение статичных файлов, настройки базы данных и так далее.
- **urls.py** задаёт ассоциации url адресов с представлениями. Несмотря на то, что этот файл может содержать все настройки url, обычно его делят на части, по одной на приложение, как будет показано далее.
- **models.py** - файл для описания и хранения моделей (модель является единственным источником информации о ваших данных. Она содержит основные поля и поведение данных, которые вы храните. Как правило, каждая модель отображается в одну таблицу базы данных.)
- **views.py** - файл с представлениями проекта. Это то место, где размещается «логика» работы нашего приложения. Оно запрашивает информацию из моделей и передаст её в шаблоны.
- **templates** - папка с шаблонами для проекта (Django предоставляет стандартный API для загрузки и рендеринга шаблонов, независимо 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.)

- **static** - папка для хранения разного рода(изображения, CSS, Javascript и др.) статических файлов

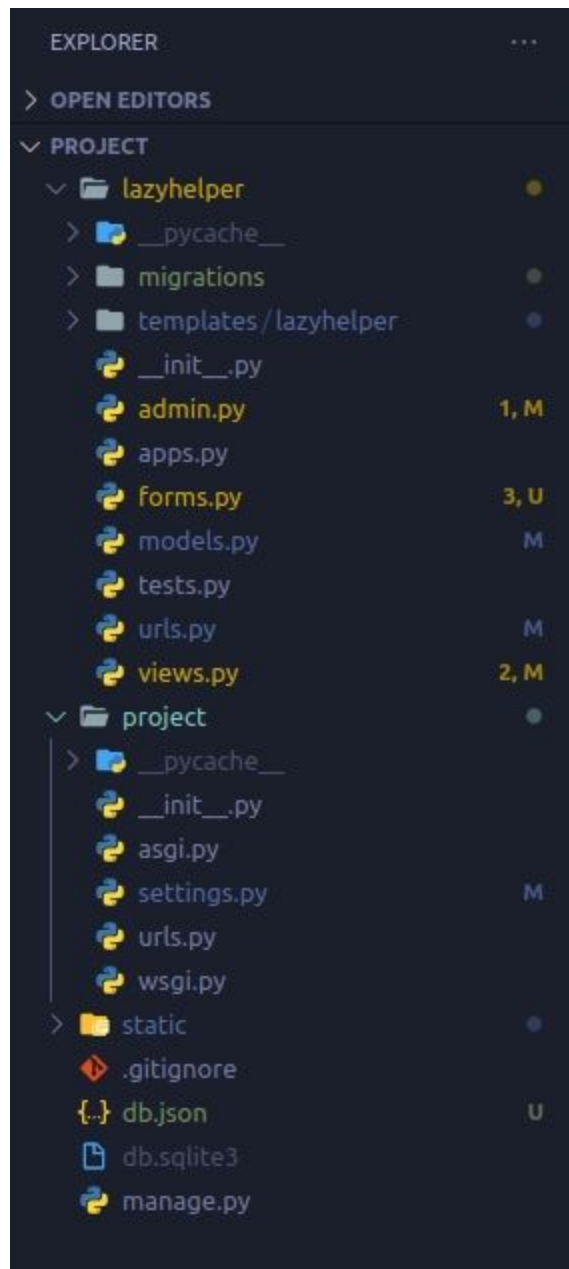


Рис. 7 - Каталог проекта

## 2. Реализация базы данных в приложении

### 2.1. Подключение БД

Подключение выполняется в файле `settings.py`. По умолчанию в настройках указано использование SQLite.

Для начала я поставил сам сервер PostgreSQL и библиотеку разработчика (она пригодится нам при установке бэкэнда). Далее всё по шагам.

1. Открываем консоль PostgreSQL
2. Создаем и настраиваем пользователя при помощи которого будем соединяться с базой данных из Django:

```
create user artem with password 'pr100artem';  
alter role user_name set client_encoding to 'utf8';  
alter role user_name set default_transaction_isolation to 'read committed';  
alter role user_name set timezone to 'UTC';
```

3. Создаем базу для нашего проекта:

```
create database lazyhelper;
```

4. Раздаем все права на пользование этой бд нашему созданному пользователю:

```
GRANT ALL PRIVILEGES ON DATABASE lazyhelper TO artem;
```

5. В окружении проекта устанавливаем бэкэнд для PostgreSQL:

```
pip install psycopg2
```

6. Настраиваем раздел DATABASES конфигурационного файла проекта(см. рис. 8):

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',  
        'NAME': 'lazyhelper',  
        'USER': 'artem',  
        'PASSWORD': 'pro100artem',  
        'HOST': '127.0.0.1',  
        'PORT': '5432',  
    }  
}
```

Рис. 8 - конфигурационный файл проекта

7. БД подключена

## 2.2. Описание и использование моделей для базы данных

Django ORM (Object Relational Mapping) является одной из самых мощных особенностей Django. Это позволяет нам взаимодействовать с базой данных, используя код Python, а не SQL.

Модели отображают информацию о данных, с которыми вы работаете. Они содержат поля и поведение ваших данных. Обычно одна модель представляет одну таблицу в базе данных.

Основы:

- Каждая модель это класс унаследованный от `django.db.models.Model`.
- Атрибут модели представляет поле в базе данных.

- Django предоставляет автоматически созданное API для доступа к данным.

Модели описываются в файле `models.py`. Самая важная часть модели – и единственная обязательная – это список полей таблицы базы данных которые она представляет. Поля определены атрибутами класса.

### Типы полей

Каждое поле в вашей модели должно быть экземпляром соответствующего `Field` класса. Django использует классы полей для определения такой информации как:

- Типа колонки в базе данных (например: `INTEGER`, `VARCHAR`).
- Виджет используемый при создании поля формы (например: `<input type="text">`, `<select>`).
- Минимальные правила проверки данных, используемые в интерфейсе администратора и для автоматического создания формы.

### Класс Meta

В каждую модель можно дописать класс `Meta`. Это просто контейнер класса с некоторыми параметрами (метаданными), прикрепленными к модели. Он определяет такие вещи, как доступные разрешения, связанное имя таблицы базы данных, является ли модель абстрактной или нет, единственной и множественной версиями имени и т.д.

В данной работе я использовал следующие модели:

- Модель `CustomUser` для таблице пользователей, унаследованная от класса `AbstractUser` для расширения стандартной модели `User`, которая уже присутствует в Django. В эту модель была добавлена связь один-к-многим с таблице ролей, что позволило мне при регистрации пользователя указывать его роль(см. рис. 9).

```

class CustomUser(AbstractUser):
    role = models.ForeignKey('Role', null=True, on_delete=models.CASCADE)

    class Meta:
        db_table = 'users'
        verbose_name = 'User'

    def __str__(self):
        return self.username

class Role(models.Model):
    role = models.CharField(max_length=11, null=True)

    class Meta:
        db_table = 'roles'

    def __str__(self):
        return self.role

```

Рис. 9 - модель для таблицы пользователей

- Модель Product для таблицы услуг, которые будут представлены на сайте. В параметрах поля можно указать choices и передать в него список возможных значений. Такое поле будет работать как Enum(см. рис. 10).

```

class Product(models.Model):
    CATEGORY = (
        ('dungeon', 'Dungeon'),
        ('raid', 'Raid'),
        ('pvp', 'PVP'),
        ('other', 'Other'),
    )

    name = models.CharField(max_length=80, null=True)
    price = models.FloatField(null=True)
    category = models.CharField(max_length=10, null=True, choices=CATEGORY, default=CATEGORY[3][0])
    description = models.TextField(max_length=600, null=True)

    class Meta:
        db_table = 'products'

    def __str__(self):
        return self.name

```

Рис. 10 - модель для таблицы услуг



- Модель Order для таблицы заказов, которые будут создаваться заказчиками и исполняться исполнителями. В данной модели у меня присутствуют два внешних ключа: один для заказчика и один для исполнителя(см. рис. 11).

```
class Order(models.Model):
    STATE = (
        ('Vacant', 'Vacant'),
        ('In progress', 'In progress'),
        ('Done', 'Done'),
    )

    product = models.ForeignKey('Product', null=True, on_delete=models.SET_NULL)
    comment = models.TextField(max_length=300, blank=True, null=True)
    customer = models.ForeignKey('CustomUser', null=True, on_delete=models.SET_NULL, related_name='customer')
    contractor = models.ForeignKey('CustomUser', null=True, on_delete=models.SET_NULL, related_name='contractor')
    creation_date = models.DateTimeField(auto_now_add=True, null=True)
    execution_date = models.DateTimeField(blank=True, null=True)
    price = models.FloatField(null=True)
    state = models.CharField(max_length=12, null=True, choices=STATE, default=STATE[0][0])

    class Meta:
        db_table = 'orders'

    def __str__(self):
        return self.product.name
```

Рис. 11 - модель для таблицы заказов.

## Использование моделей

После определения моделей необходимо указать Django что необходимо их использовать. Сделайте это отредактировав файл настроек и изменив INSTALLED\_APPS, добавив пакет, который содержит ваш models.py.

```
INSTALLED_APPS = [
    ...
    'lazyhelper',
]
```

После чего нам остается лишь выполнить миграцию базы данных.

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Первая команда отвечает за создание новых миграций на основе изменений, которые вы внесли в свои модели. Она создаёт файл миграции, по которому мы будем обновлять нашу текущую базу данных, и к которому мы потом можем откатиться назад, если что-то пойдет не так(см. рис. 12).

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [
        ('auth', '0012_alter_user_first_name_max_length'),
    ]

    operations = [
        migrations.CreateModel(
            name='CustomUser',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('password', models.CharField(max_length=128, verbose_name='password')),
                ('last_login', models.DateTimeField(blank=True, null=True, verbose_name='last login')),
                ('is_superuser', models.BooleanField(default=False, help_text='Designates that this user has all permissions without explicitly assigning them.', verbose_name='superuser status')),
                ('username', models.CharField(error_messages={'unique': 'A user with that username already exists.'}, help_text='Required. 150 characters or fewer. Letters, digits and @/./-/_. only.', max_length=150, unique=True, validators=[django.contrib.auth.validators.UnicodeUsernameValidator()], verbose_name='username')),
                ('first_name', models.CharField(blank=True, max_length=150, verbose_name='first name')),
                ('last_name', models.CharField(blank=True, max_length=150, verbose_name='last name')),
                ('email', models.EmailField(blank=True, max_length=254, verbose_name='email address')),
                ('is_staff', models.BooleanField(default=False, help_text='Designates whether the user can log into this admin site.', verbose_name='staff status')),
                ('is_active', models.BooleanField(default=True, help_text='Designates whether this user should be treated as active. Unselect this instead of deleting accounts.', verbose_name='active')),
                ('date_joined', models.DateTimeField(default=django.utils.timezone.now, verbose_name='date joined')),
                ('groups', models.ManyToManyField(blank=True, help_text='The groups this user belongs to. A user will get all permissions granted to each of their groups.', related_name='user_set', related_query_name='user', to='auth.Group', verbose_name='groups')),
            ],
            options={
                'verbose_name': 'User',
                'db_table': 'user',
            },
            managers=[
                ('objects', django.contrib.auth.models.UserManager()),
            ],
        ),
        migrations.CreateModel(
            name='Product',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=80, null=True)),
                ('price', models.FloatField(null=True)),
                ('category', models.CharField(choices=[('dungeon', 'Dungeon'), ('raid', 'Raid'), ('pvp', 'PVP'), ('other', 'Other')], default='other', max_length=10, null=True)),
                ('description', models.TextField(max_length=600, null=True)),
            ],
            options={
                'db_table': 'products',
            },
        ),
    ],
```

Рис. 12 - файл миграции БД.

Вторая же команда отвечает непосредственно за применение или отмену миграции.

## 2.3. Формы в Django

Класс Form является сердцем системы Django при работе с формами. Он определяет поля формы, их расположение, показ виджетов, текстовых

меток, начальных значений, валидацию значений и сообщения об ошибках для "неправильных" полей (если таковые имеются). Данный класс, кроме того, предоставляет методы для отрисовки самого себя в шаблоне при помощи predefined форматов (таблицы, списки и так далее), или для получения значения любого элемента (позволяя выполнять более точную отрисовку).

Для того, чтобы создать класс с функционалом базового класса `Form` мы должны импортировать библиотеку `forms`, наследовать наш класс от класса `Form`, а затем объявить поля формы.

Самые популярные аргументы для большинства полей перечислены ниже:

- `required`: Если `True`, то данное поле не может быть пустым, или иметь значение `None`. Данное значение установлено по умолчанию.
- `label`: Текстовая метка, используемая для рендеринга поля в HTML-код. Если `label` не определена, то Django попытается создать ее значение при помощи имени поля, переводя первый символ в верхний регистр, а также заменяя символы подчеркивания пробелами (например, для переменной с именем `renewal_date`, будет создан следующий текст метки: `Renewal date`).
- `initial`: Начальное значение для поля при показе формы.
- `help_text`: Дополнительный текст, который может быть показан на форме, для описания того, как использовать поле.
- `disabled`: Если установлено в `True`, то поле показывается, но его значение изменить нельзя. По умолчанию равно `False`.

Ниже представлен список используемых в работе форм(см. рис. 13):

- `UserRegistrationForm` - форма для регистрации пользователя
- `UserChangeForm` - форма для авторизации пользователя
- `OrderForm` - форма для добавления заказа

```

class UserRegistrationForm(UserCreationForm):
    class Meta(UserCreationForm):
        model = CustomUser
        fields = ['username', 'email', 'role', 'password1', 'password2']

class UserChangeForm(UserChangeForm):
    class Meta:
        model = CustomUser
        fields = UserChangeForm.Meta.fields

class OrderForm(ModelForm):
    class Meta:
        model = Order
        fields = ['customer', 'product', 'comment', 'price']

```

Рис. 13 - файл с формами Django.

## 2.4. Авторизация/Регистарция

Сперва рассмотрим систему авторизации и регистрации. Код находится в файле `views.py`. На странице регистрации имеются следующие поля(см. рис. 14):

HOME / CATALOG / GUARANTEES

### Sign up

Username

Email address

Role

Customer ▼

Password

Password confirmation

Complete

Рис. 14 - форма регистрации.

При нажатии на кнопку «Complete» срабатывает обработчик события, который проверяет данные на корректность и добавляет нового пользователя в базу данных, если поля прошли валидацию(см. рис. 15):

```
def registration(request):
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            messages.success(request, 'You finally registered')
            form.save()
            return redirect('/login')
        else:
            messages.error(request, 'Something go wrong')
    else:
        form = UserRegistrationForm()

    context = {'form': form}
    return render(request, 'lazyhelper/register.html', context)
```

Рис. 15 - функция для обработки регистрации.

За добавление нового пользователя в базу данных отвечает метод `form.save()` объекта `form`. Он забирает все значения с формы и сохраняет их в базу данных. До этого данные проходят валидацию методом `is_valid()`. Все валидаторы описаны в файле конфигурации проекта. Если в процессе создания пользователя данные не пройдут валидацию, мы увидим ошибку, которая сообщит нам об этом. Если же все данные пройдут проверку, то мы увидим сообщение об успешной регистрации.

С авторизацией дела обстоят немного иначе(см. рис. 16):

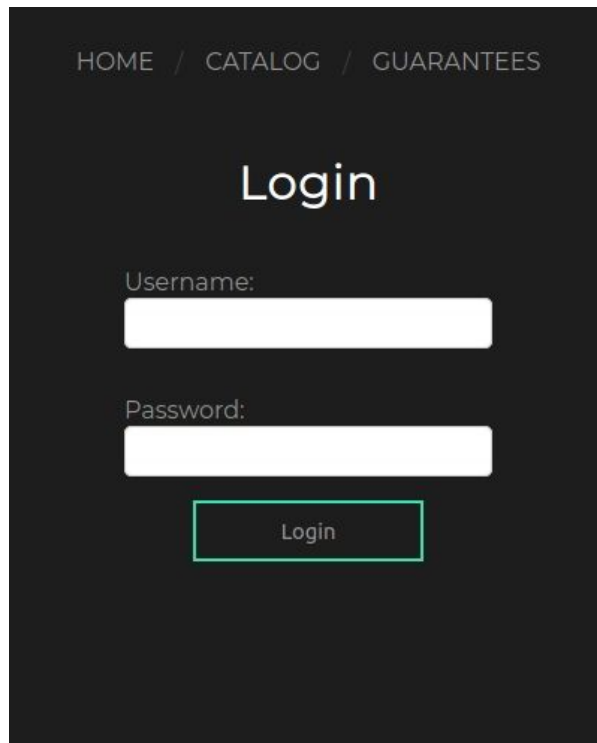


Рис. 16 - форма авторизации.

Для авторизации используется готовое решение от Django из модуля `django.contrib.auth`.

```
from django.contrib.auth import views as vws
```

```
path('login/', vws.LoginView.as_view(), name='login'),  
path('logout/', vws.LogoutView.as_view(), name='logout'),
```

При нажатии на кнопку Login срабатывает обработчик, который проверяет введенные данные на корректность и сверяет их с данными в базе данных. При совпадении контроллер возвращает пользователя и совершает редирект на домашнюю страницу.

## 2.5. Создание заявки на услугу

Создание заявки происходит с помощью ранее показанной формы OrderForm(см. рис. 17). Важно отметить, что создавать заявку могут только авторизованные пользователи, в случае же если пользователь не авторизован его перебросит на страницу авторизации.

The image shows a mobile application interface for creating a service request. At the top, there is a navigation bar with the text "HOME / CATALOG / GUARANTEES". Below this, the form is titled "Customer:" and features a dropdown menu with the selected value "alesha". The next section is titled "Product:" and has a dropdown menu with the selected value "CN boost – Castle Nathria raid carry". Below that is a section titled "Comment:" with a text input field containing the placeholder text "My comment". The final section is titled "Price:" and displays the value "180" next to a green horizontal line, with a small icon of two arrows pointing up and down to its right. At the bottom of the form is a green rectangular button labeled "Create".

Рис. 17 - форма для создания заказа.

За добавление заказа отвечает функция `create_order`, которая подставляет `id` пользователя в поле для `id` заказчика. После чего она проверяет тип входящего запроса, и если это запрос `POST`, то выполняет валидацию формы и если она проходит успешно, то сохраняет полученный объект в базу данных(см. рис. 18).

```
def create_order(request, pk):
    customer = CustomUser.objects.get(id=pk)
    form = OrderForm(initial={'customer': customer})
    if request.method == 'POST':
        form = OrderForm(request.POST)
        if form.is_valid():
            form.cleaned_data['price'] = str(Product.objects.get(name=form.cleaned_data['product']).price)
            print(form.cleaned_data)
            form.save(update_fields=['price'])
            return redirect('/')

    context = {'form': form}
    return render(request, 'lazyhelper/order_form.html', context)
```

Рис. 17 - функция для обработки метода `POST` при создании заказа.

После создания заказа, пользователь может увидеть его у себя на странице, в списке вакантных заказов(см. рис. 18).

Hello alesha!

You have 3 order(s) now.

PRODUCT	CATEGORY	PRICE	CREATION DATE	STATE	FINISHED
In progress orders:					
Twisting corridors – challenge mode Torghest	Other	234.0	Dec. 15, 2020, 9:51 a.m.	In progress	Done
Vacant orders:					
CN boost – Castle Nathria raid carry	Raid	None	Dec. 15, 2020, 9:37 a.m.	Vacant	
Mythic+ keystone boost	Dungeon	234.0	Dec. 15, 2020, 9:49 a.m.	Vacant	
Done orders:					

Рис. 18 - шаблон для страницы пользователя.



Важно отметить, что шаблон динамический и в зависимости от роли текущего пользователя он либо отображает заказы текущего пользователя(те, которые в ожидании, те, которые в процессе и завершённые), либо заказы исполнителя (те, которые доступны на выполнение, те, которые он сейчас выполняет и уже выполненные заказы). Функция для рендеринга шаблона представлена ниже(см. Рис. 19).

```
def user(request, pk):
    user = CustomUser.objects.get(id=pk)
    if user.role.role == 'Customer':
        orders = user.customer.all()
        order_count = orders.count()
        vacant = orders.filter(state='Vacant')
        inprogress = orders.filter(state='In progress')
        done = orders.filter(state='Done')
        context = {'user': user, 'orders': orders, 'order_count': order_count,
                  'vacant_orders': vacant, 'inprogress_orders': inprogress, 'done_orders': done}
    elif user.role.role == 'Contractor':
        all_orders = Order.objects.filter(state='Vacant')
        orders_count = all_orders.count()
        contractor_orders = user.contractor.all()
        inprogress = contractor_orders.filter(state='In progress')
        done = contractor_orders.filter(state='Done')
        context = {'user': user, 'contractor_orders': contractor_orders, 'orders_count': orders_count,
                  'inprogress_orders': inprogress, 'done_orders': done, 'all_orders': all_orders}
    return render(request, 'lazyhelper/user.html', context)
```

Рис. 19 - функция для рендеринга страницы пользователя

## Заключение

В рамках данной курсовой работы были рассмотрены основные виды баз данных существующих на момент выполнения работы. Также было уделено внимание различным видам СУБД. Для реализации поставленного задания была выбрана наиболее подходящая СУБД. В процессе выполнения была спроектирована база данных, которая впоследствии была создана при помощи Django ORM. По итогу выполнения курсовой работы было реализовано полностью рабочее Web-API представляющее собой каталог игровых услуг.

## Список использованной литературы

1. Базы данных – Что это такое? [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://hostiq.ua/wiki/database/>);
2. What is a Relational Database? [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://aws.amazon.com/relational-database/>);
3. Visual Studio Code [Электронный ресурс]. - Электронные данные. - Режим доступа: ([https://ru.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://ru.wikipedia.org/wiki/Visual_Studio_Code));
4. HTML5 [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.wikipedia.org/wiki/HTML5>);
5. CSS [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.wikipedia.org/wiki/CSS>);
6. David Maxwell - Tango with Django[Электронный ресурс] - Электронные данные. - Режим доступа: (<https://www.tangowithdjango.com/>)