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 курсовой работе на тему: «Многопользовательская игра «Квадраты»»

Студент: Левушкин И. К.
Группа: ИУ7-72Б
Научный руководитель: Рогозин Н.О.

Москва, 2020 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7___
(Индекс)
_____ __И.В. Рудаков
(И.О. Фамилия)
«_____» _____ 2020 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Компьютерные сети_____

Студент группы ____ИУ7-72б____

_____Левушкин Илья Кириллович_____
(Фамилия, имя, отчество)

Тема курсового проекта _Создание многопользовательской игры “Квадраты”_____
_____
_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный и др.)
___учебный_____

Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

1. Техническое задание

Реализовать многопользовательскую игру с определенными правилами. Пользователь должен иметь возможность зарегистрироваться, авторизовываться, добавлять и удалять друзей из списка друзей, приглашать друзей в игру, начинать игру с другом, пригласившим пользователя, а также с пользователем, находящимся в общем списке ждущих игроков. Соответственно, пользователь также должен иметь возможность просматривать этот список и добавляться в него.

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть,

экспериментально-исследовательский раздел, заключение, список литературы, приложения.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.). На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ___ » _____ 2020 г.

Руководитель курсового проекта _____ ___Н.О. Рогозин___
(Подпись, дата) (И.О. Фамилия)

Студент _____ ___И.К. Левушкин__
(Подпись, дата) (И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

Введение

Сетевые многопользовательские игры занимают значительную нишу в современной игровой индустрии. Число игроков и суммы денег, вращающиеся в этой области, поражают воображение. В 2014 году в «League of Legends» ежемесячно заходили 67 миллионов игроков. В 2015-м на чемпионате мира по игре в «Dota 2» призовой фонд составил более 16 миллионов долларов [1].

Целью данного проекта является создание сетевой многопользовательской настольной игры «Квадраты».

1 Аналитический раздел

1.1 Формализация задачи

В соответствии с техническим заданием необходимо разработать систему, позволяющую первоначально осуществлять регистрацию учетной записи и вход в нее.

Далее авторизованный пользователь может проводить стандартные CRUD-операции (создание, просмотр, редактирование, удаление) над определенными сущностями.

А именно:

- просмотр и редактирование сущности списка друзей;
- просмотр и редактирование сущности пула игроков;
- просмотр, создание и редактирование сущности игры;

Авторизованный пользователь с правами администратора будет получать дополнительные возможности:

- просмотр, редактирование сущности пользователя;
- редактирование и удаление сущности игры;

Для достижения цели необходим анализ существующих принципов построения веб-приложений.

1.2 Протоколы прикладного уровня

1.2.1 HTTP

HTTP — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных [2].

HTTP явялется наиболее распространенным протоколом всемирной паутины (World Wide Web) [3]. Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier – уникальный идентификатор ресурса) в запросе клиента. Основными ресурсами являются хранящиеся на сервере файлы, но ими могут быть и другие логические или абстрактные объекты. Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: MIME-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Существует несколько версий этого протокола: HTTP 1.0, HTTP 1.1, HTTP/2 и HTTP/3. Для транспортировки HTTP-сообщений служит протокол TCP (обычно 80 порт).

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей, которые передаются в следующем порядке:

1. стартовая строка — определяет тип сообщения;
2. заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. тело сообщения — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

1.2.2 HTTPS

HTTPS — расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивается защита этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Эта система была разработана компанией Netscape Communications Corporation, чтобы обеспечить аутентификацию и защищённое соединение. HTTPS широко используется во всемирной паутине для приложений, в кото-

рых важна безопасность соединения, например, в платежных системах.

SSL/TLS-сертификат — это цифровая подпись сайта. С помощью сертификата подтверждается подлинность сайта. Перед тем как установить защищённое соединение, браузер запрашивает этот документ и обращается к центру сертификации, чтобы подтвердить легальность документа. Если он действителен, то браузер считает этот сайт безопасным и начинает обмен данными.

В настоящее время HTTPS поддерживается всеми популярными браузерами.

1.3 Архитектура

1.3.1 Клиент-серверная архитектура

Клиент-серверная архитектура — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине [4].

Рисунок 1: Клиент-серверная архитектура.

1.3.2 REST архитектура

REST — архитектурный стиль взаимодействия компонентов распределённого приложения в сети [5]. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

Ограничения REST:

- модель клиент-сервер;
- отсутствие состояния;
- кэширование;
- единообразие интерфейса:
  - идентификация ресурсов;
  - манипуляция ресурсами через представление;
  - «самоописываемые» сообщения;
  - гипермедиа как средство изменения состояния приложения;
- слои.

Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «RESTful».

Рой Филдинг — один из главных авторов спецификации протокола HTTP, описывает влияние архитектуры REST на масштабируемость следующим образом:

- простота унифицированного интерфейса;
- открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей (даже при работающем приложении);
- прозрачность связей между компонентами системы для сервисных служб;
- переносимость компонентов системы путем перемещения программного кода вместе с данными;
- надёжность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений или данных.

1.3.3 Монолитная архитектура

В программной инженерии монолитная модель относится к единой неделимой единице. Концепция монолитного программного обеспечения заключается в том, что различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте [6].

Большим преимуществом монолита является то, что его легко реализовать. В монолитной архитектуре можно быстро начать реализовывать бизнес-логику вместо того, чтобы тратить время на размышления о межпроцессном взаимодействие.

Однако в монолите практически нет изоляции. Проблема или ошибка в модуле может замедлить или разрушить все приложение.

1.3.4 Микросервисная архитектура

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов.

Микросервисы легче держать модульными. Технически это обеспечивается жесткими границами между отдельными сервисами.

В больших компаниях разные сервисы могут принадлежать разным командам. Услуги могут быть повторно использованы всей компанией. Это также позволяет командам работать над услугами в основном самостоятельно. Нет необходимости координировать развертывание между командами. Развивать сервисы лучше с увеличением количества команд.

На рисунке 2 показаны различия между монолитной архитектурой и микросервисной.

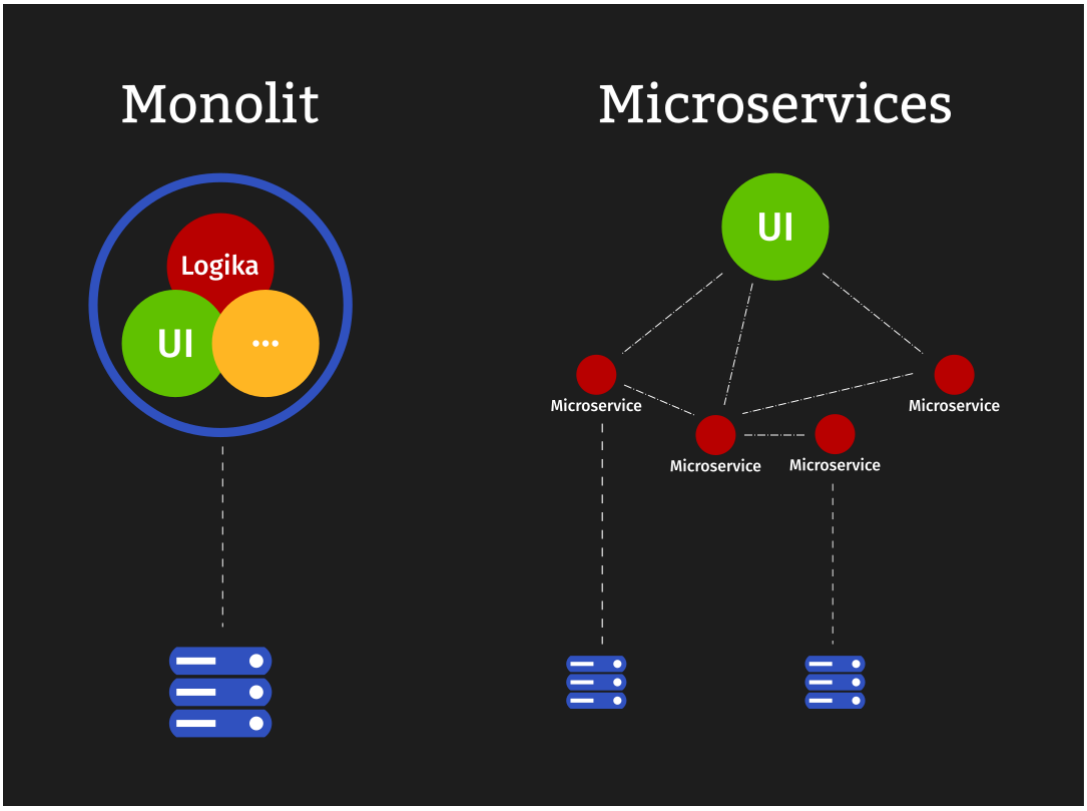

Рисунок 2: Монолитная и микросервисная архитектуры.

1.4 Базы данных

1.4.1 Общие сведения

Очевидно, клиентские данные нужно каким-то образом хранить, поэтому требуется анализ существующих типов баз данных. Рассмотрим популярные и зарекомендовавшие себя решения – реляционную и нереляционную. Помимо этих видов существуют и другие (иерархическая, сетевая) [7], но выбранные БД являются наиболее популярными в мире веба.

1.4.2 Реляционная модель

Реляционная модель данных использует организацию данных в виде двумерных таблиц. Каждая такая таблица, называемая реляционной таблицей или отношением, представляет собой двумерный массив и обладает следующими свойствами: все столбцы в таблице однородные, т.е. все элементы в одном столбце имеют одинаковый тип и максимально допустимый размер; каждый столбец имеет уникальное имя; одинаковые строки в таблице отсутствуют; порядок следования строк и столбцов в таблице не имеет значения. Основными структурными элементами реляционной таблицы являются поле и запись.

Поле в таблице, называемое внешним ключом, может содержать ссылки на столбцы в других таблицах, что позволяет их соединять.

Ниже приведен пример организации данных курьерской службы приведен на рисунке 3.

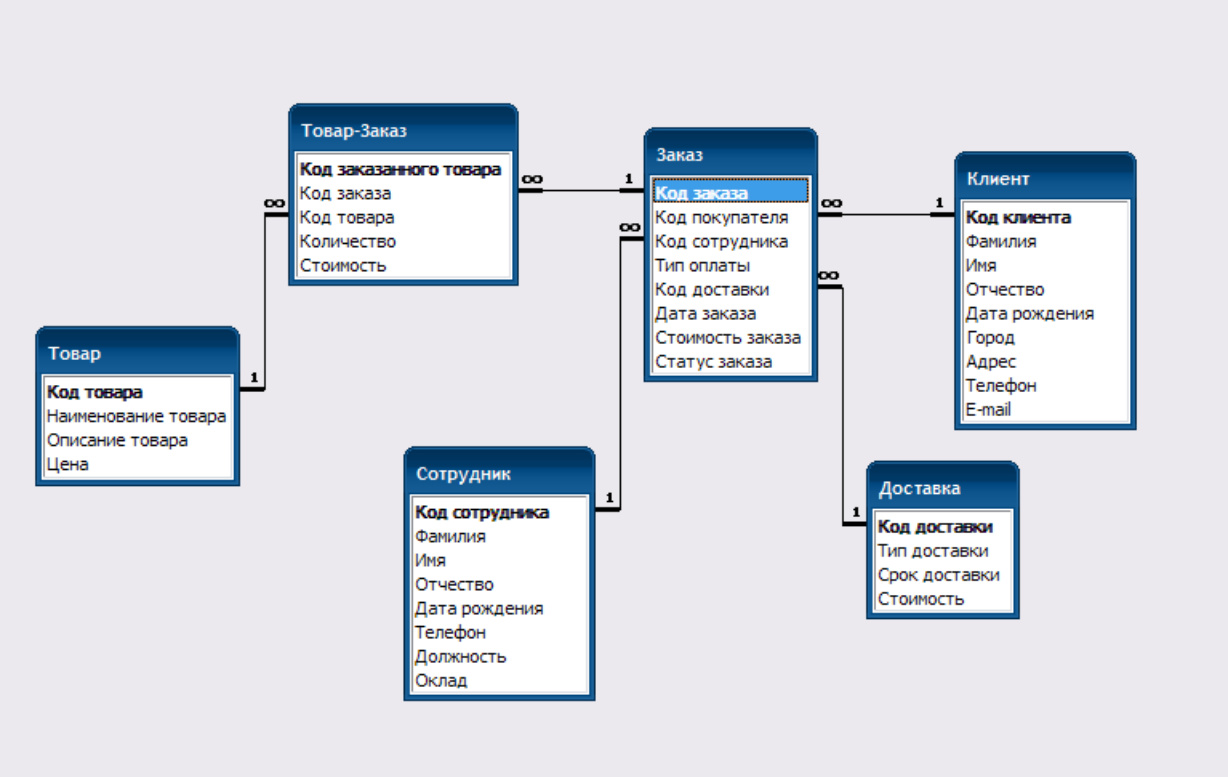

Рисунок 3: Пример реляционной модели.

Для доступа к данным используется язык структурированных запросов (SQL) [8]. Реляционная модель – надежный выбор для многих приложений, один из старейших способов организации информации и самый популярный.

Сложности состоят в изначальном проектировании такой БД, ведь при плохо спроектированной системе возможно значительное снижение скорости доступа к данным, стремительное разрастание занимаего пространства и другие проблемы, связанные с поддержкой и модификацией программы.

1.4.3 Нереляционная модель

NoSQL – группа типов БД, предлагающих подходы, отличные от стандартного реляционного шаблона [?]. Говоря NoSQL, подразумевают либо «не-SQL», либо «не только SQL», чтобы уточнить, что иногда допускается SQL-подобный запрос.

Описание схемы данных в случае использования NoSQL-решений может осуществляться через использование различных структур

данных: хеш-таблиц, деревьев и других.

Вместе с достоинствами в виде гибкости системы имеются и недостатки, связанные с отсутствием жесткой схемы отношения между данными. Кроме того, зачастую нереляционные БД привязаны к конкретной СУБД и требуют специализированных знаний для работы с ними, в то время как реляционная модель точно имеет поддержку общего стандарта написания запросов.

Выводы по аналитическому разделу

В рамках данной работы было решено использовать реляционную модель организации данных, поскольку она остается надежным выбором для многих систем и основана на строгом математическом аппарате, что позволяет лаконично описывать необходимые операции над данными.

В результате проведенного анализа было решено разрабатывать веб-приложение в клиент-серверной архитектуре, используя методологию REST, что позволит сделать ПО надежным и легко масштабируемым.

Сервер будет представлять собой монолитную архитектуру, поскольку является наилучшим решением для небольших проектов.

2 Конструкторский раздел

2.1 Правила игры

В данной работе реализуется логическая настольная игра с полной информацией [17], что означает, что игра игрокам известны функция полезности, правила игры, а также ходы других игроков.

2.1.1 Инвентарь

В игру «Квадраты» играют на квадратной доске, расчерченной вертикальными и горизонтальными линиями. Стандартная доска имеет размер поля 11x11, но также возможны любые другие вариации. У каждого из игроков имеется неограниченное количество фишек его цвета, который присваивается ему в начале игры.

2.1.2 Основные правила

Играют в игру два игрока, один из которых получает красные фишки, другой - синие. Цель игры - набрать количество очков большее чем у противника.

2.1.3 Начальная позиция

Перед началом игры доска пуста. Первыми ходят синие. Затем красные. Далее ходы делаются по очереди.

Игрок не может отказываться от хода.

2.1.4 Порядок игры

Делая ход, игрок выставляет одну фишку на доску в любую незанятую клетку. Фишки, однажды поставленные на доску, не убираются. За каждый собранный «квадрат» на поле игрок получает определенное количество очков в зависимости от величины

стороны квадрата.

Квадрат представляет собой конструкцию, соединенную в углах четырьмя фишками одного цвета. Соединение квадратов по горизонтали-вертикали дает $1, 3, 5, 7, 9...$ очков в зависимости от длины стороны квадрата. Соединение квадратов по диагонали дает $2, 4, 6, 8, 10, ...$ очков в зависимости от длины стороны квадрата.

Конструкция, состоящая из 4 фишек одного цвета, у которой все ребра равны, и **имеющая** или нет внутри себя или между фишек другие фишки, называется **квадратом**.

2.1.5 Завершение партии

Партия заканчивается, как только поле будет полностью заполнено фишками. Победителем определяется игрок, набравший наиболее количество очков.

2.2 Описание функционала

Формулировка требования к возможностям программного обеспечения из раздела 1.1 требует большей конкретики. Диаграмма вариантов использования (Use-Case) описывает взаимоотношения и зависимости между вариантами использования и действующими лицами, участвующими в процессе.

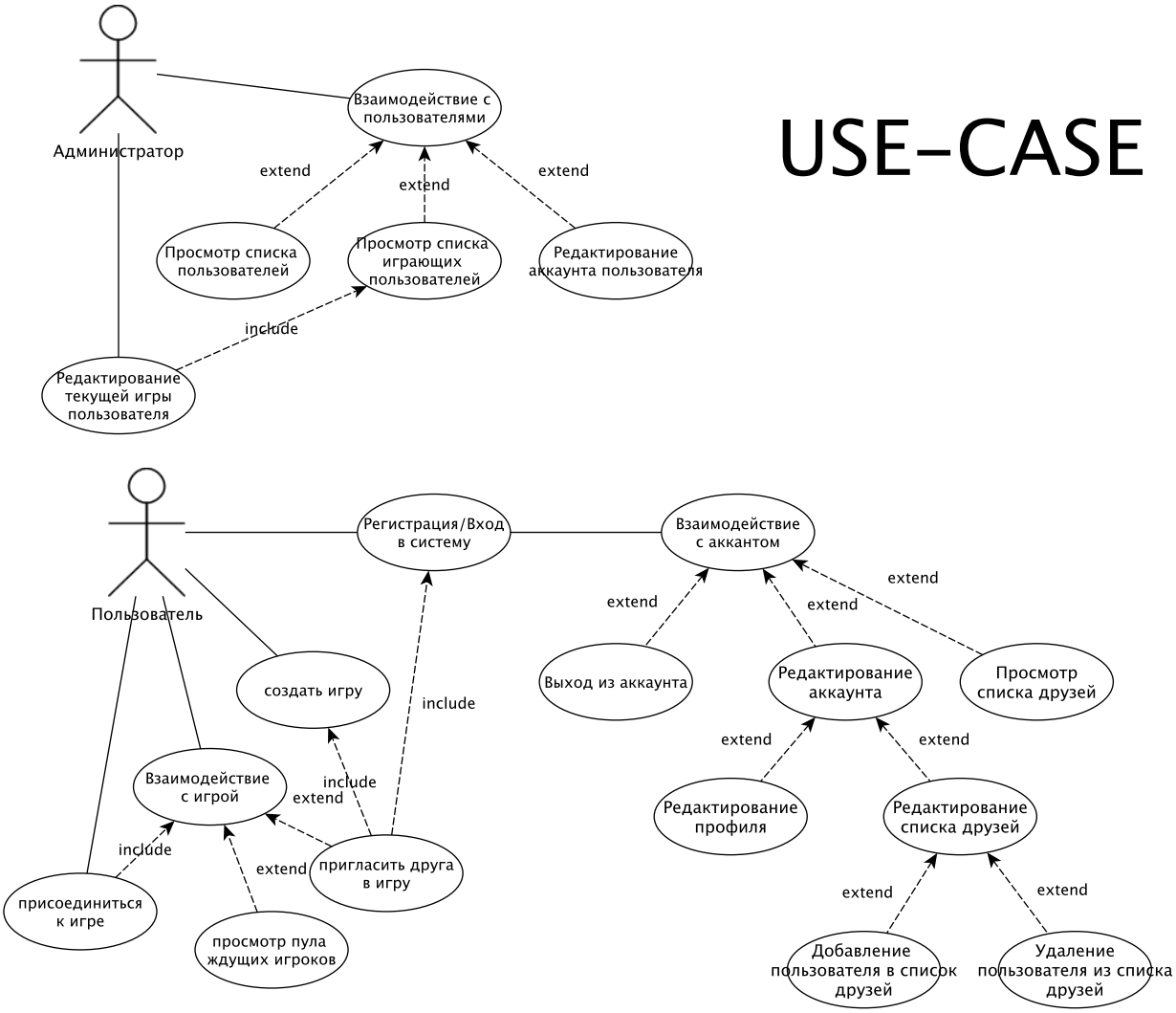

Рисунок 4: Use-Case диаграмма.

2.3 Структура базы данных

Ниже приведены диаграмма «сущность-связь» (ER), описывающая концептуальную схему предметной области, и диаграмма сущностей спроектированной базы данных, графическое изображение которой дает четкое представление о действующих сущностях и их атрибутах.

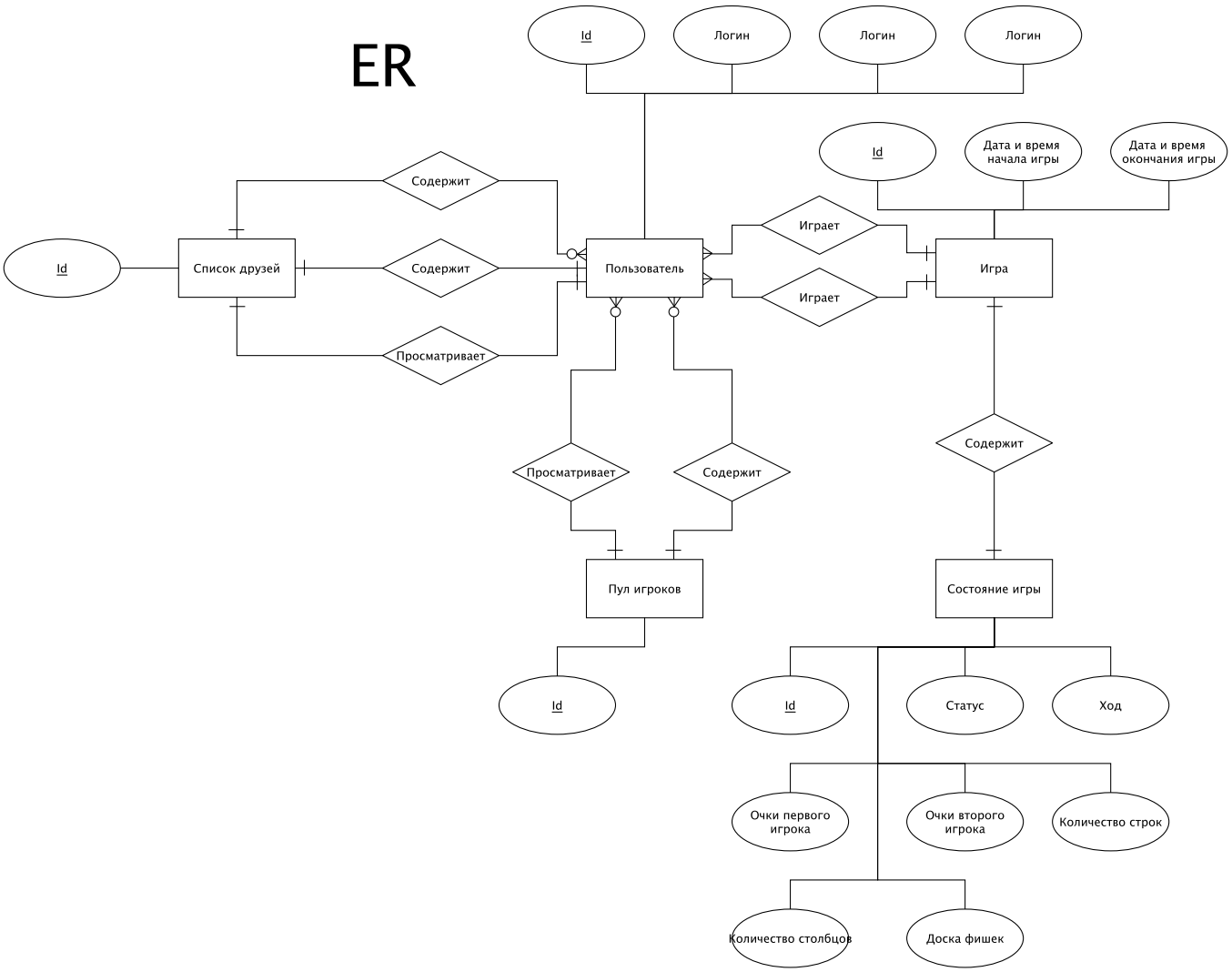

Рисунок 5: ER диаграмма.

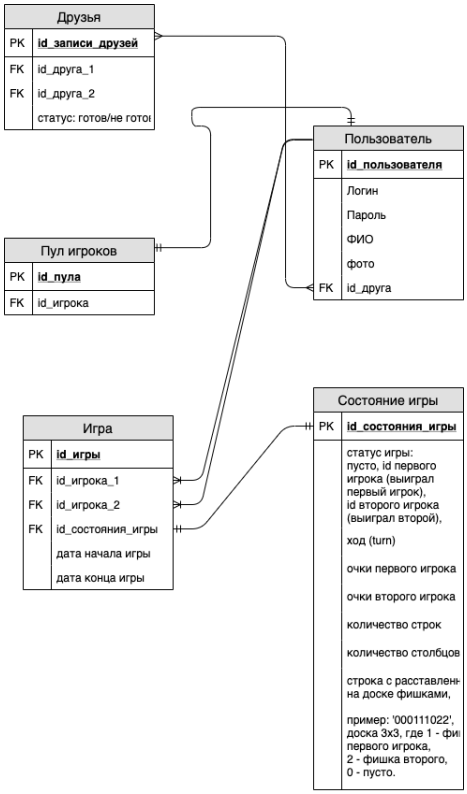

Рисунок 6: Диаграмма базы данных.

2.4 API системы в идеологии REST

Грамотным решением является отделение логики программы для последующего удобного взаимодействия с ней и возможности переиспользования (построение API).

Ниже приведено краткое описание спроектированного API системы в идеологии REST в формате Swagger:

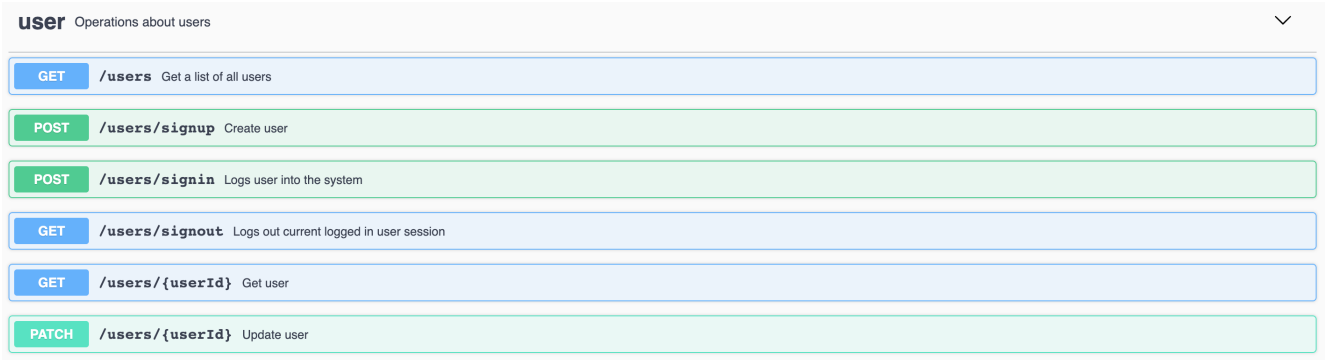

Рисунок 7: API сущности пользователя (user).

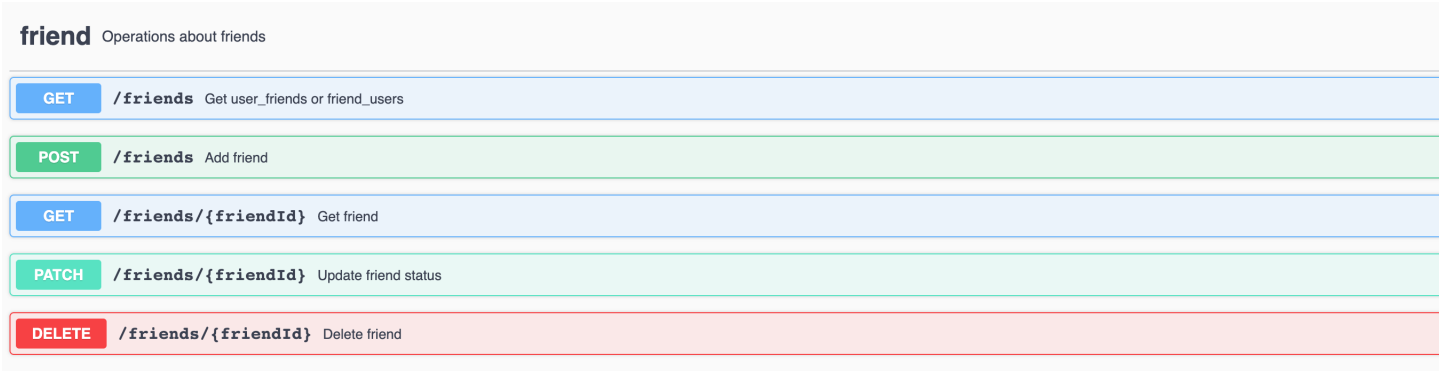

Рисунок 8: API сущности друга (friend).

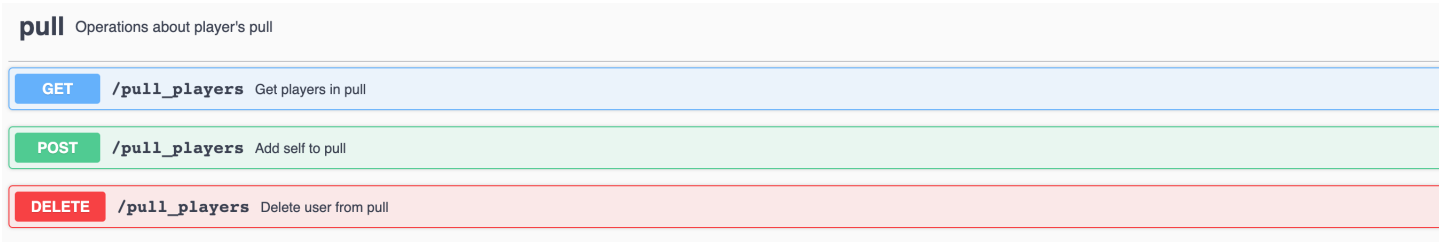

Рисунок 9: API сущности пула игроков (pull).

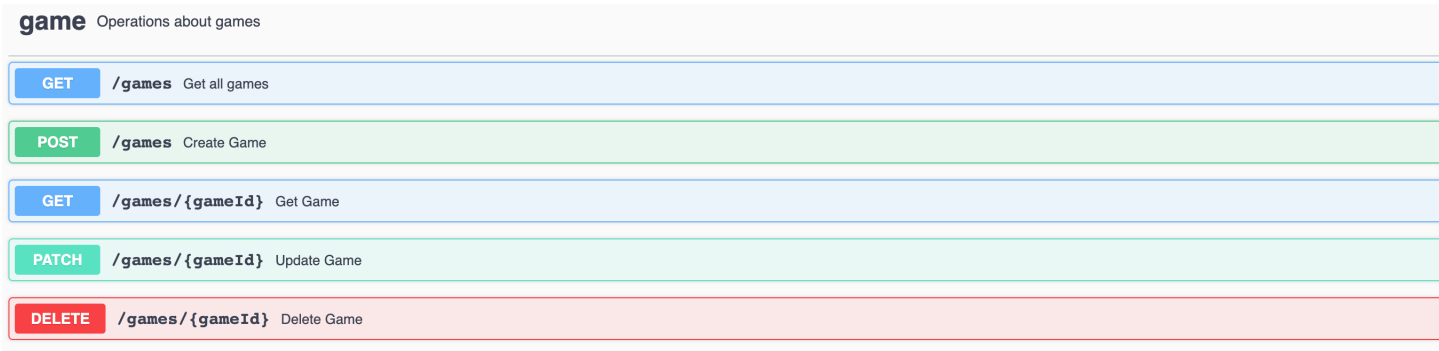

Рисунок 10: API сущности игры (game).

2.5 Описание классов и компонентов

Общее представление компонентов программы содержится на рисунке 11.

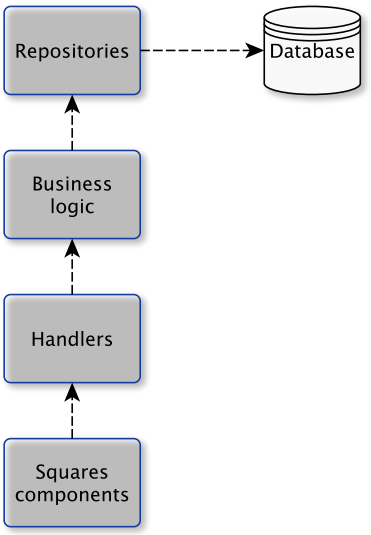

Рисунок 11: Диаграмма компонентов.

Ниже приведена диаграмма классов бизнес-логики.

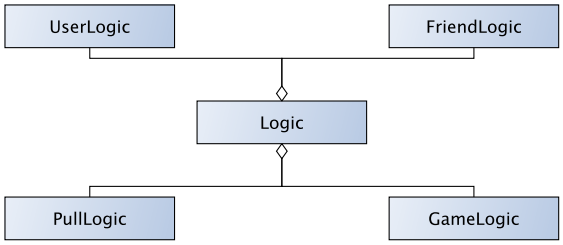

Рисунок 12: Диаграмма классов бизнес-логики.

Аналогично слою бизнес-логики строятся компоненты обработки запросов и репозитория. Такая структура, на первый взгляд, выглядит избыточной, но позволяет абстрагировать уровни приложения и легче его тестировать.

Ниже представлена функциональная модель системы.

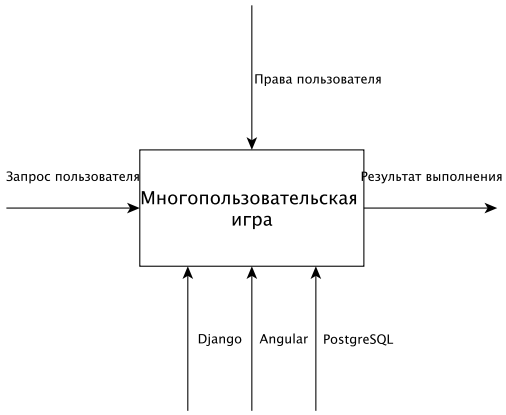

Рисунок 13: Функциональная модель системы.

Выводы по конструкторскому разделу

Были спроектированы и представлены описание функционала программного обеспечения в виде Use-Case диаграммы, структура базы данных с помощью ER-диаграммы и диаграммы базы данных, API системы в идеологии REST, а также дано описание классов и компонентов системы.

3 Технологический раздел

3.1 Использованные технологии

Основной используемой технологией для построения серверной части приложения выбран веб-фреймворк Django [10]. Django – это веб- фреймворк языка Python [11], использующий шаблон проектирования MVC.

Для построения клиентской части приложения выбран фреймворк Angular [14], использующий язык TypeScript [15].

В качестве системы управления базами данных используется PostgreSQL [9]. Фундаментальная характеристика представленной СУБД – это поддержка пользовательских объектов и их поведения. Это делает PostgreSQL невероятно гибким и надежным. Среди прочего, здесь можно создавать, хранить и извлекать сложные структуры данных.

Ниже представлен основной технологический стек проекта.

Front-end:

- Язык гипертекстовой разметки HTML5[12]
- Таблицы стилей CSS3[13]
- Язык программирования TypeScript[15]

Back-end:

- СУБД PostgreSQL[9]
- Фреймворк Django[10]
- Язык программирования Python3[11]

Листинг 1: Модель доступа к данным сущности pull

```
from django.db import models
from .users import Users
from ..managers import PullPlayersManager

from datetime import datetime


class PullPlayers(models.Model):
        player = models.OneToOneField(Users, on_delete=models.PROTECT)

        date_time_appear = models.DateTimeField(default=datetime.now)

        objects = PullPlayersManager()

        def __str__(self):
                return self.player.user.username
```

3.2 Примеры кода ПО

Организацию серверной и клиентской частей можно проследить на примере одной из сущностей. Для демонстрации возьмем структуру проекта, показанную в листинге 1.

Принципы чистой архитектуры указывают на разделение кода на независимые слои. На примере метода добавления пользователя в пул игроков можно просмотреть, как это соблюдается в текущей реализации.

Листинг 2: Post-метод pull-обработчика HTTP-запросов

```
from rest_framework.response import Response
from rest_framework import status
from rest_framework import authentication, permissions
from rest_framework.views import APIView

from ..repository import PullPlayersCRUD
from ..serializers import PullPlayersSerializer
from ..business_logic import PullPlayersLogic


class PullPlayersView(APIView):
        manager = PullPlayersCRUD()
        logic = PullPlayersLogic()

        authentication_classes = [authentication.TokenAuthentication]
        permission_classes = [permissions.IsAuthenticated]

        # добавить себя в пул
        def post(self, request):
                if not self.logic.user_in_pull(request.user.pk):
                        self.manager.create_note(request.user.pk)
                else:
                 return Response(f'User is already in pull', status=status.HTTP_403_FORBIDDEN)

                 return Response({"success": f'User was added in pull successfully'},
                 status=status.HTTP_201_CREATED)
```

Листинг 2 содержит post-метод обработчика HTTP-запросов, который выполняет проверку JWT-токена (процедура аутентификации), передает право классу бизнес-логики пула решать, можно ли создавать новую запись в таблице пула игроков, и если ответ положительный, то передает управление классу менеджеру пула, чтоб тот создал новую запись в таблице пула.

Листинг 3: Pull-бизнес-логика

```
from ..models import PullPlayers
from .user import UsersLogic

from ..repository import UsersCRUD

class PullPlayersLogic:

        def __init__(self):
                self.objects = PullPlayers.objects
                self.users_logic = UsersLogic()
                self.users_manager = UsersCRUD()


        def user_is_admin(self, user):
                return self.users_logic.user_is_admin(user)

        def user_in_pull(self, user_pk):
                users = self.users_manager.get_note_by_user_pk(user_pk)
                return self.objects.is_user_exists(users.pk)

        def user_in_pull2(self, users_pk):
                return self.objects.is_user_exists(users_pk)
```

Далее представлен уровень бизнес-логики (листинг 3), где происходит установление принадлежности пользователя к пулу и принятие решения в соответсвующии с результатом.

Листинг 4: Репозиторий пула

```
from ..models import Users, Games, PullPlayers, Friends


class PullPlayersCRUD:

        def __init__(self):
                self.objects = PullPlayers.objects
                self.user_objects = Users.objects

        def get_all(self):
                return self.objects.get_all()

        def get_note(self, user_pk):
                users = self.user_objects.get_note_by_user_pk(user_pk)
                return self.objects.get_note(users.pk)

        def delete_note(self, user_pk):
                users = self.user_objects.get_note_by_user_pk(user_pk)
                self.objects.delete_note(self.user_objects.get_note(users.pk))

        def delete_note2(self, users_pk):
                self.objects.delete_note(self.user_objects.get_note(users_pk))

        def create_note(self, user_pk):
                users = self.user_objects.get_note_by_user_pk(user_pk)
                return self.objects.create_note(self.user_objects.get_note(users.pk))
```

Ниже идет обращение к методу репозитория, в котором происходит непосредственное взаимодействие с базой данных (листинг 4).

Исходя из представленного кода видно, что слои независимы друг от друга.

3.3 Nginx

Nginx – это веб-сервер с обратным прокси-сервером, который используется для обслуживания динамического содержимого веб-сайта и контроля загрузки сервера [?]. Основная настройка nginx указана в листинге 5.

В секции events настраивается подведение nginx относительно сетевых соединений. Если multi_accept установлен в on, то рабочий процесс за один раз будет принимать сразу все новые соединения.

Значение worker_connections определяет максимально возможное количество одновременных соединений к серверу.

Секция http объединяет все настройки относительно работы протокола http. Параметр sendfile включает или отключает использование сервером одноимённого системного вызова. Включение этого параметра увеличивает производительность nginx за счёт использования более эффективного способа ввода-вывода.

Включение опции tcp_nopush заставляет nginx пытаться отправлять HTTP-заголовки в одном пакете.

Параметр tcp_nodelay влияет разрешает или запрещает использование сервером опции сокета TCP_NODELAY при работе с постоянными соединениями.

Параметр gzip управляет включением/отключением сжатия сервером данных, передаваемых клиенту. Включение этой опции увеличивает нагрузку на центральный процессор системы, однако позволяет существенно сократить объём передаваемых данных.

Параметр proxy_cache_path позволяет настроить параметры хранения кэша. nginx кэш позволяет значительно сократить количество запросов на бэкенд. Это достигается путем сохранения HTTP ответа, на определенное время, а при повторном обращении к ресурсу, отдачи его из кэша без проксирования запроса на бекенд.

При помощи опции default_type задаётся MIME-тип по умолчанию, который будет передавать сервер клиенту, если этот тип не удалось определить.

Параметр access_log определяет местоположение и формат лог-файла доступа к содержимому.

Веб-сервер по умолчанию распределяет запросы равномерно между бэкендами (с учетом весов). Этот стандартный метод в Nginx, так что директива включения отсутствует. Веса выбраны так, что из каждых 4 запросов бэкенд *localhost:8000* будет обрабатывать 2, *localhost:8001* 1, а *localhost:8002* 1 (листинг 6).

Листинг 5: Конфигурация nginx

```
events {
multi_accept on;
worker_connections 1024;
}

http {
charset utf-8;
sendfile on;
tcp_nopush on;
tcp_nodelay on;
server_tokens off;
log_not_found off;
types_hash_max_size 4096;
client_max_body_size 16M;
gzip on;
gzip_comp_level 5;
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=all:32m max_size=1g;

# MIME
include mime.types;
default_type application/octet-stream;

# logging
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" "$http_x_forwarded_for"';

access_log  /Users/ilalevuskin/github/GitHub/computer_networks/project/nginx/access.log  main;
error_log /Users/ilalevuskin/github/GitHub/computer_networks/project/nginx/error.log warn;

server {
listen 82;
add_header Server Squares always;

location / {
set $do_not_cache 0;
if ($http_cookie ~* ".+" ) {
set $do_not_cache 1;
}
proxy_cache_bypass $do_not_cache;
proxy_pass http://127.0.0.1:81;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_cache all;
proxy_cache_methods GET HEAD;
proxy_cache_valid any 1m;
}
}

upstream backend_balanced {
server localhost:8000 weight=2;
server localhost:8001 weight=1;
server localhost:8002 weight=1;
}
}
```

Листинг 6: Балансировка нагрузки

```
upstream backend_main {
server localhost:8000;
}

map $request_method $backend
{
GET backend_balanced;
HEAD backend_balanced;
default backend_main;
}
```

3.4 Примеры работы серверной части приложения

Ниже приведены примеры запросов и ответов сервера. Тестирование проводилось с помощью утилиты Postman [16]

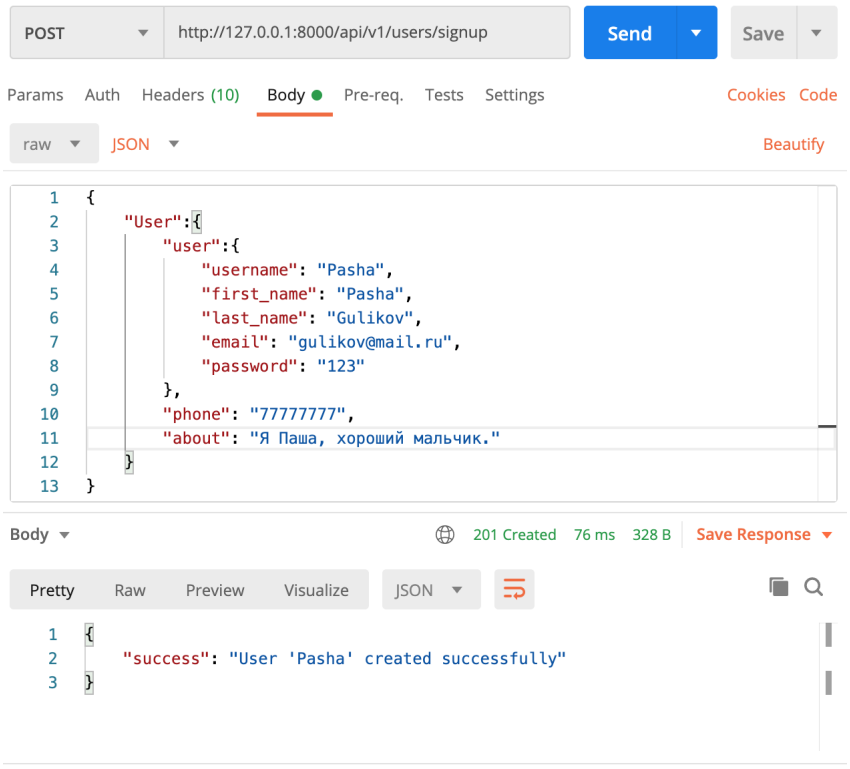

Рисунок 14: Регистрация пользователя.

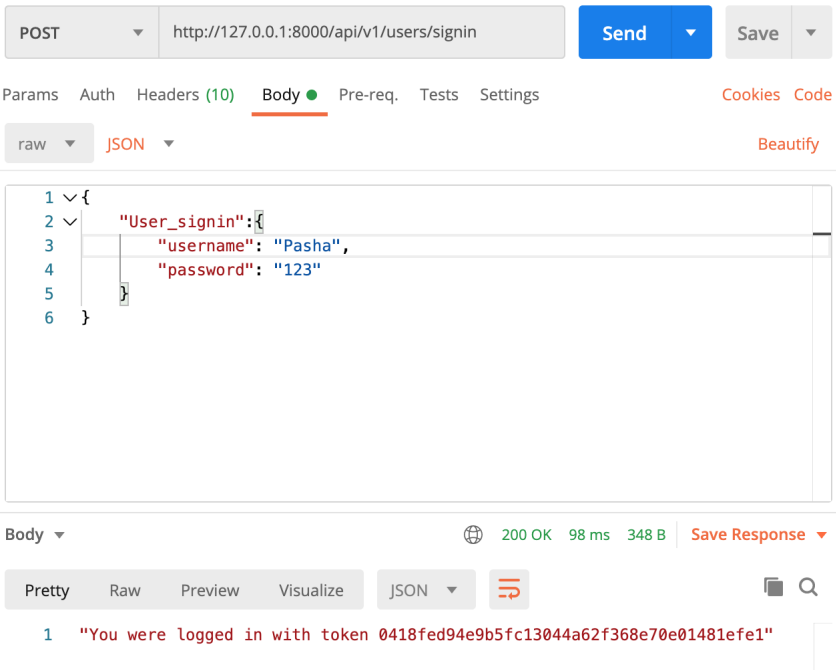

Рисунок 15: Вход пользователя.

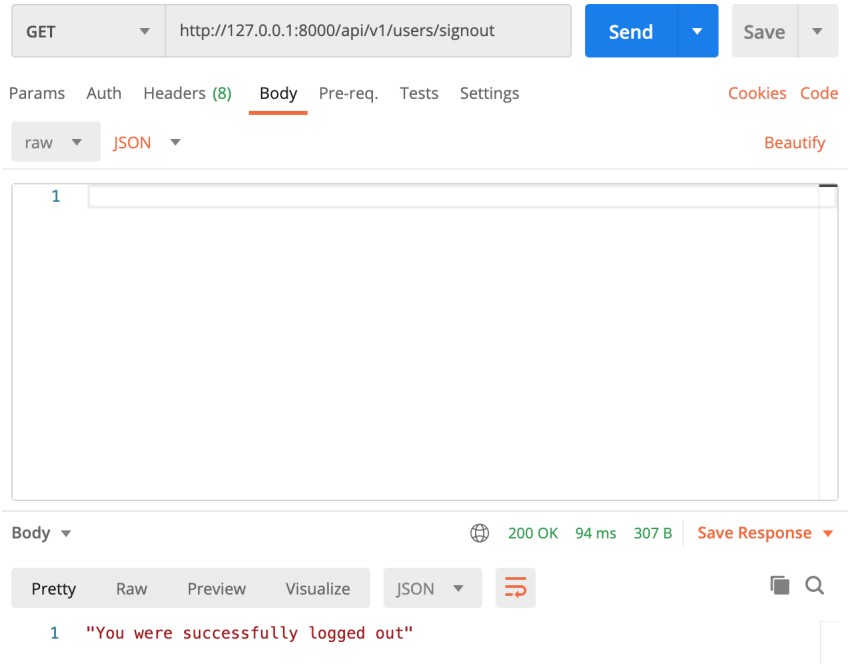

Рисунок 16: Выход пользователя.

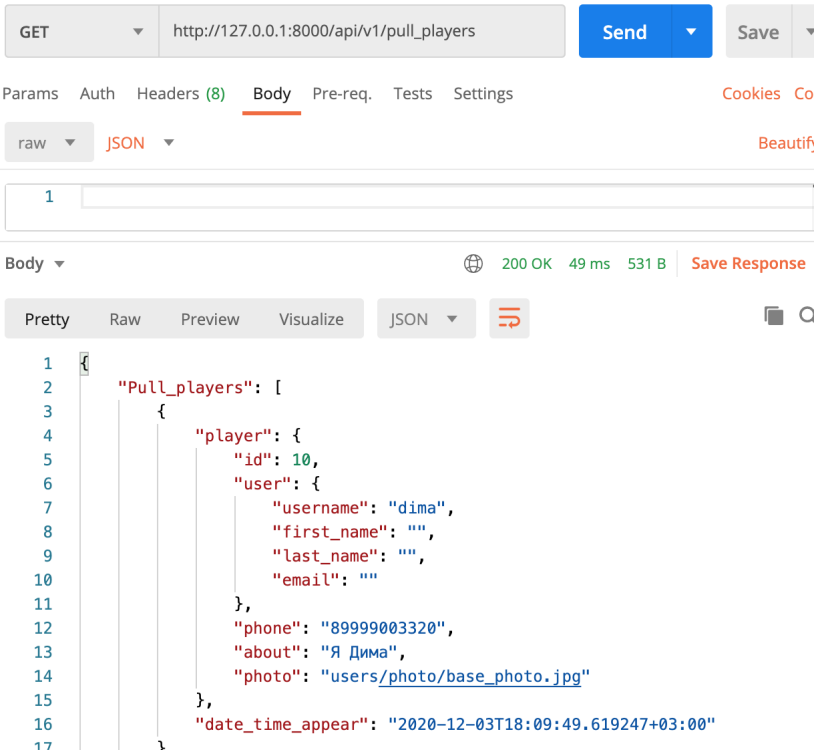

Рисунок 17: Просмотр пула игроков.

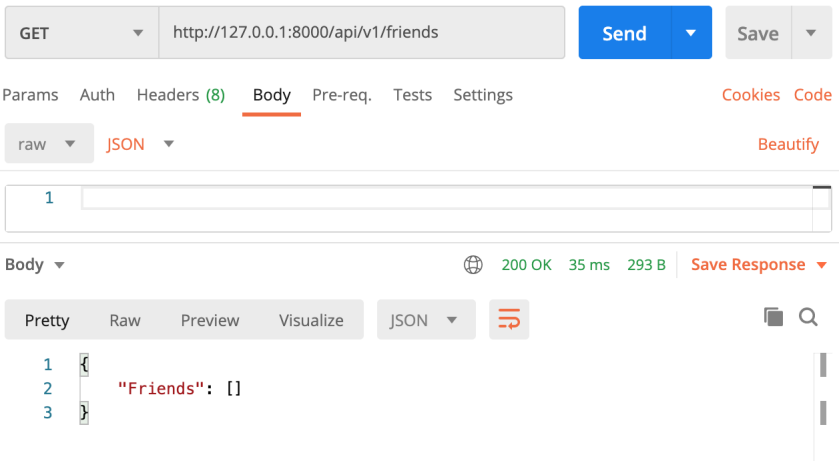

Рисунок 18: Просмотр списка друзей.

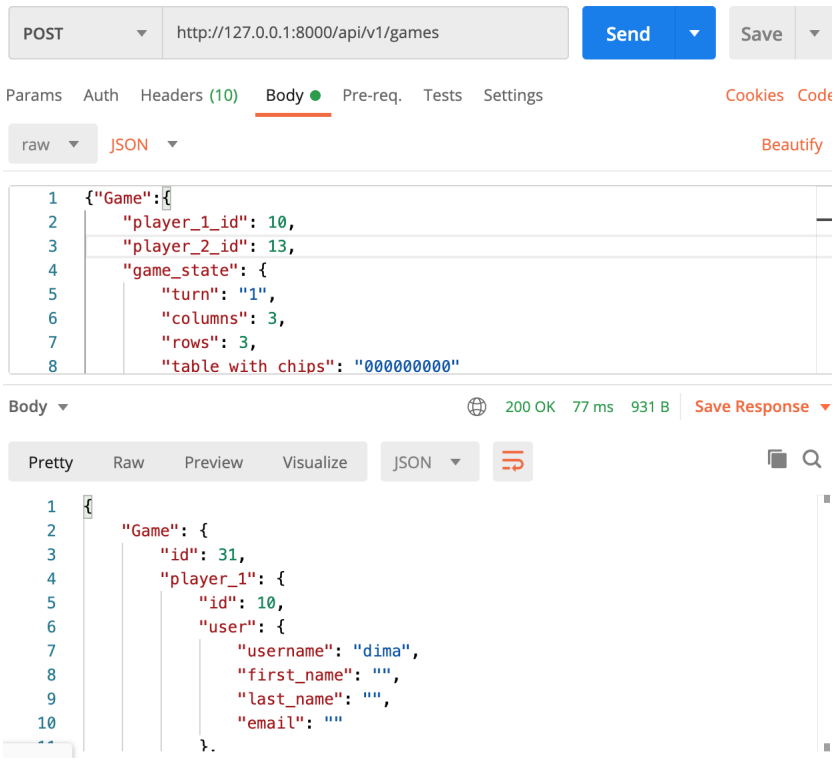

Рисунок 19: Создание игры.

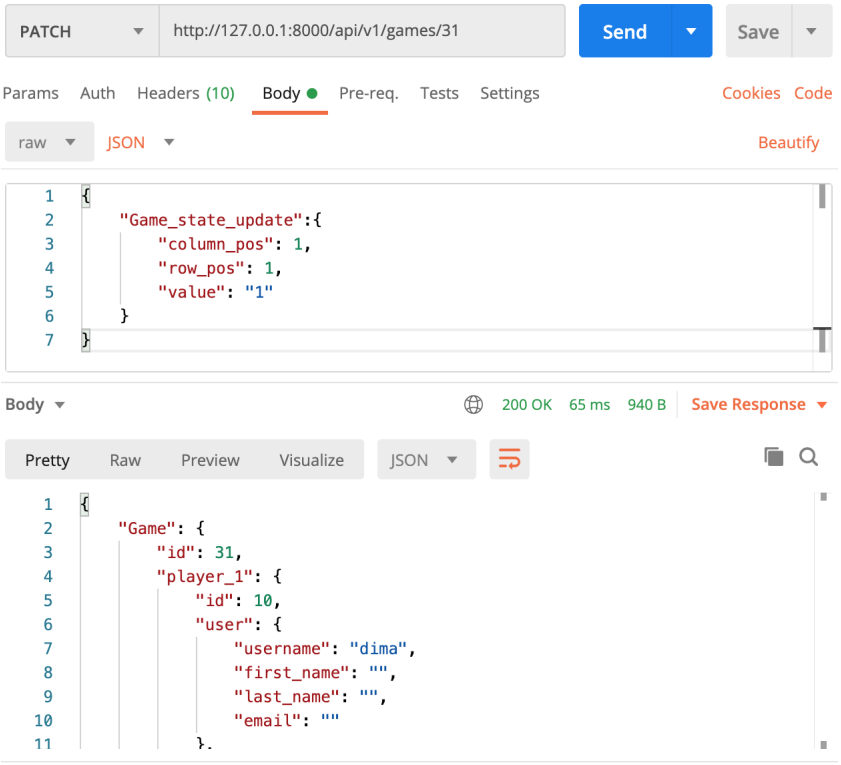

Рисунок 20: Обновление игры.

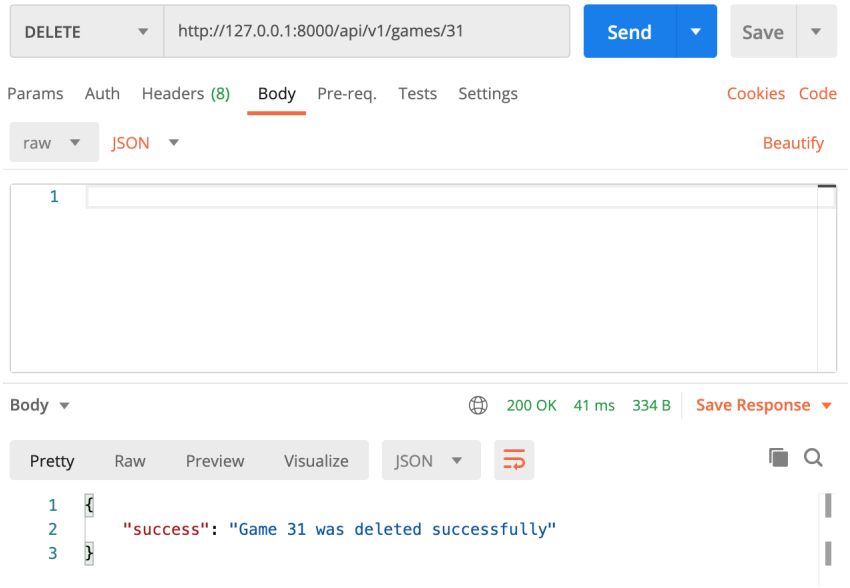

Рисунок 21: Удаление игры администратором.

Выводы по технологическому разделу

В результате проделанной работы были выбраны технологии для разработки ПО.

Были приведены листинги кода ПО и примеры работы серверной части приложения.

Заключение

В ходе проделанной работы был выполнен анализ принципов построения современных веб-приложений.

Спроектирована архитектура программы на основе шаблона MVC со стороны сервера. Была использована реляционная модель организации данных. Веб-приложение разрабатывалось, используя методологию REST.

Таким образом, с помощью СУБД PostgreSQL и фреймворков Django и Angular было реализовано веб-приложение SPA - многопользовательская игра «Квадраты».

В планах дальнейшего развития приложения - расширение функционала, в рамках которого будет реализован список лучших игроков, обмен сообщений между игроками в процессе игры, возможность регистрации/авторизации через платформы ВК, Facebook и Twitter.

Также, планируется поддержка протокола HTTPS и развертывание приложения на одном из хостингов.

Список литературы

[1] Многопользовательские игры [Электронный ресурс]. – Режим доступа: http://sd.blackball.lv/library/Mnogopoljzovateljskie_igry__Razrabotka свободный – (02.12.2020)

[2] Обзор протокола HTTP [Электрон- ный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview, свободный – (02.12.2020)

[3] About The World Wide Web [Электронный ресурс]. – Режим доступа: https://www.w3.org/WWW/, свободный – (02.12.2020)

[4] Клиент-серверная архитектура в картинках [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/495698/, свободный – (02.12.2020)

[5] Руководство по созданию RESTful сервиса [Электронный ресурс]. – Режим доступа: http://www.restapitutorial.ru/, свободный – (02.12.2020)

[6] Лучшая архитектура для MVP [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/otus/blog/476024/, свободный – (02.12.2020)

[7] Основные понятия баз данных [Электронный ресурс]. – Режим доступа: http://inf.susu.ac.ru/Klinachev/lc_sga_26.htm, свободный – (02.12.2020)

[8] Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: полное руководство – М.: Вильямс, 2014

[9] PostgreSQL [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/, свободный – (02.07.2020)

[10] Django [Электронный ресурс]. – Режим доступа: https://www.djangoproject.com/, свободный – (13.07.2020)

[11] Python [Электронный ресурс]. – Режим доступа: https://www.python.org/, свободный – (12.07.2020)

[12] HTML5 [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/HTML/HTML5, свободный – (12.07.2020)

[13] CSS3 [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS/Reference, свободный – (12.07.2020)

[14] Angular Documentation [Электронный ресурс]. – Режим доступа: https://angular.io/, свободный – (02.12.2020)

[15] TypeScript Documentation [Электронный ресурс]. – Режим доступа: https://www.typescriptlang.org/, свободный – (02.12.2020)

[16] Postman [Электронный ресурс]. – Режим доступа: https://www.postman.com/, свободный – (02.12.2020)

[17] Оуэн Г. Теория игр. — М.: Вузовская книга, 2004. — 216 с.: ил. — 500 экз.