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:

«Разработка информационной системы для частных поликлиник»

Студент группы **ИУ7-66Б** _____ **П. А. Калашков**
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ **О. В. Кузнецова**
(Подпись, дата) (И.О. Фамилия)

***2023** г.*

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В работе используются следующие термины с соответствующими определениями:

БД — база данных

СУБД — система управления базами данных

ВВЕДЕНИЕ

В XXI веке медицинское обслуживание является неотъемлемой частью жизни каждого человека. Несмотря на то, что большая часть населения России пользуется муниципальными учреждениями здравоохранения, лишь половина обращавшихся остаётся удоволетворена качеством и скоростью полученной медицинской помощи (по данным Всероссийского центра изучения общественного мнения) [1].

Треть россиян посещает частные поликлиники [1], что приводит к накоплению огромных объёмов данных, связанных с медицинским обслуживанием пациентов. Эффективность использования работниками поликлиник влияет на качество оказываемой медицинской помощи. Таким образом, создание информационной системы для частныхи поликлиник является актуальной задачей на 2023 год.

Целью работы: разработка информационной системы для частных поликлиник. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области;
2) определить функционал, реализуемый информационной системой;
3) изучить существующие на рынке решения;
4) спроектировать и разработать информационную систему в соответствии с поставленной задачей;
5) исследовать зависимость RPS от количества потоков нагрузочного тестирования.

1 Аналитическая часть

В данном разделе проводится анализ предметной области, анализ способов хранения данных в приложении и выбор оптимального, а также формализация требований к разрабатываемой программе.

1.1 Анализ предметной области

Поликлиникой будем называть многопрофильное лечебно-профилактическое учреждение, оказывающее медицинскую помощь населению на закрепленной территории на догоспитальном этапе.

Основными функциями и задачами поликлиники являются:

— оказание квалифицированной специализированной медицинской помощи населению непосредственно в поликлинике;
— оказание первой медицинской помощи при острых заболеваниях, травмах, отравлениях и других неотложных состояниях независимо от места проживания больного;
— своевременная госпитализация нуждающихся в стационарном лечении;
— экспертиза временной нетрудоспособности, освобождение больных от работы, направление на медико-социальную экспертизу лиц с признаками стойкой утраты трудоспособности;
— организация и проведение комплекса профилактических мероприятий, направленных на снижение заболеваемости, инвалидности и смертности среди населения, проживающего в районе обслуживания, а также среди работающих на прикрепленных предприятиях;
— организация и осуществление диспансеризации населения (здоровых и больных);
— организация и проведение мероприятий по санитарно-гигиеническому воспитанию населения, пропаганде здорового образа жизни.

Согласно рекомендации Минздрава России [2], в штате поликлиники долж-

но быть не менее 50 человек, однако, в случае частной поликлиники, это количество может быть уменьшено до следующих позиций:

1) главнй врач (1 должность);
2) врачи специалисты (минимум 2 должности);
3) уборщик (1 должность);
4) системный администратор (1 должность).

1.2 Анализ способов хранения данных в приложении

Рассмотрим понятия БД и СУБД, модели данных, а также

1.2.1 БД и СУБД

База данных, сокращённо БД (*англ.* DB — database) [3] — совокупность структурированных взаимосвязанных данных, относящихся к определённой предметной области и организованных таким образом, что эти данные могут быть использованы для решения многих задач многими пользователями,

Система управления базами данных, сокращённо СУБД (*англ.* DBMS — database management system) [3] — программная система, предназначенная для создания на ЭВМ общей базы данных для множества приложений, поддержания её в актуальном состоянии и обеспечения эффективного доступа пользователей к содержащимся в ней данным в рамках предоставленных им полномочий.

Основными функциями СУБД являются:

1) управление данными во внешней памяти (обеспечение абстракции пользователя от организации данных);
2) управление данными во внутренней памяти (обеспечение работы с буферами оперативной памяти);
3) управление транзакциями (поддержка транзакций по принципу ”всё или ничего обеспечение их целостности);
4) журнализация (возможность восстановить состояние после сбоя);
5) поддержка языков БД (например, поддержка SQL).

1.2.2 Модели данных и способы классификации СУБД

Моделью данных (*англ.* data model) [4] называют абстракцию, которая, будучип риложима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними. На рисунке 1 приведена классификация моделей данных.

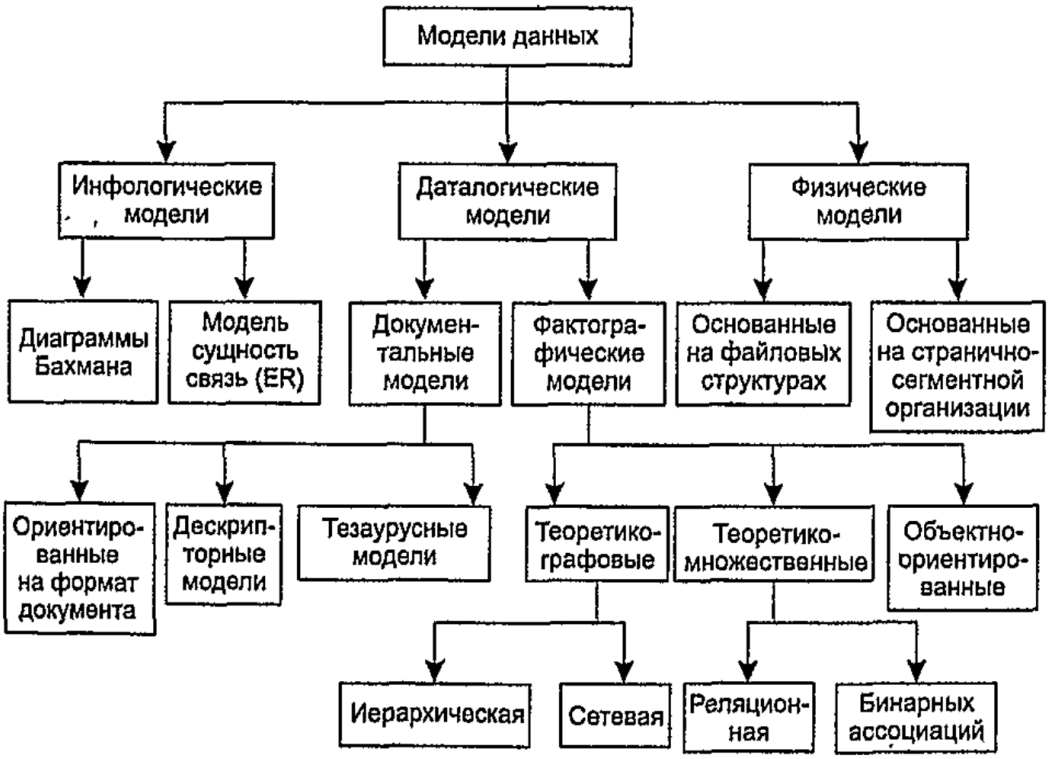

Рисунок 1 – Классификация моделей данных

Рассмотрим, как можно классифицировать существующие СУБД по двум харатеристикам: модели хранения данных и архитектуре орагизации хранения данных.

1.2.3 Модели хранения данных

По модели хранения данных выделяют следующие СУБД:

1) дореляционные (инвертированые списки, иерархические и сетевые СУБД)
2) реляционные;
3) постреляционные.

Системы, основанные на инвертированных списках (например, Datacom/DB

компании Applied Data Research, Inc.), поддерживают два типа операторов: прямые поисковые операторы (например, найти первую запись таблицы по некоторому пути доступа), а также операторы, находящие запись в терминах относительной позиции от предыдущей записи. В таких СУБД отсутствуют общие правила определения целостности данных, а хранимые таблицы и пути доступа к ним видны всем пользователям,

Иерархические СУБД содержат упорядоченный набор деревьев, причём каждый тип дерева состоит из одной корневой записи и упорядоченного набора типов поддеревьев. Для базы данных определяется полный порядок обхода деревьев — например, сверху вниз и слева направо. Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмы IBM.

Сетевые СУБД содержат наборы записей и связей между этими записями. Сети являются естественным способом представления отношений между объектами и всемозможными взаимосвязями между ними. Сетевая модель опирается на математическую структуру направленного графа — структуры, состоящей из узлов, соединённых направленными рёбрами (узлы представляют собой объекты в виде типов записей данных, а рёбра — связи между объектами). К известным сетевым системам управления базами данных относится: DBMS, IDMS, TOTAL, VISTA.

Реляционная модель данных (РМД) была предложена математиком Э.Ф. Коддом в 1970 г. РМД является наиболее широко распространенной моделью данных и единственной из трех основных моделей данных, для которой разработан теоретический базис с использованием теории множеств.

Основной структурой в РМД является отношение, вследствие чего и была называна модель (от *англ.* relation — отношение). Представление отношения может быть осуществлено в виде таблицы, а управление данными, которые связывают отношения, обеспечивается при помощи SQL — декларативного языка структурированнных запросов (*англ.* Structured Query Language), который осно-

ван на реляционной алгебре. Несмотря на наличие таких достоинств, как простота и популярность использования, реляционные СУБД имеют и недостатки, такие, как отсутствие специальных средств для отображения различных типов связей (например, ”многие ко многим”).

Примерами реляционных СУБД являются Oracle, Microsoft SQL Server, а также PostgreSQL.

1.2.4 Архитектуры организации хранения данных

По архитектуре организации хранения данных выделяют следующие СУБД:

1) файл-серверные;
2) клиент-серверные;
3) встраиваемые;
4) сервисно-ориентированные;
5) прочие (временные, пространственные, пространственно-временные).

При файл-серверной организации хранения данных система не имеет сетевого разделения компонентов диалога и использует файловую систему компьютера для отображения. Это облегчает нагрузку на центральный процессор (поскольку файл-сервер лишь извлекает данные из файлов), но добавляет нагрузку на сеть, что является существенным недостатком таких систем. Примерами файл-серверных СУБД являются Microsoft Access и MySQL (до версии 5.0).

Клиент-серверные СУБД состоят из двух частей: клиентской части (которая включена в прикладную программу) и непосредственно сервера, являющегося внешней по отношению к клиенту программой. Преимущества данной системы:

— возможность по мере надобности заменить один сервер другим;

— обеспечивание разграничения доступа между пользователями;

— снижение нагрузки на сеть и на клиентские машины.

Недостатком клиент-серверных СУБД является само существование сервера (это плохо для локальных программ) и количество вычислительных ресур-

сов, потребляемых сервером. Клиент-серверными СУБД являются, например, Oracle, Microsoft MySQL Server, PostgreSQL, а также MySQL (версии 5.0 и выше).

Встраиваемой СУБД называют СУБД, тесно связанную с прикладной программой и работающую на том же компьютере, не требуя профессионального администрирования. Благодаря непосредственной близости данных к прикладной области, встраиваемые СУБД применяются в программах, которые хранят большие объёмы даннных, но в которых не требуется доступ с многих компьютеров. Встраиваемыми системами являются SQLite, Firebird Embedded, Microsoft SQL Server Compact.

1.2.5 Выбор типа СУБД

По модели хранения данных будет выбрана реляционная модель, поскольку её популярность и простота, а также наличие формального математического аппарата сделали наиболее подходящими для использования в системах представления знаний, чем другие системы.

По архитектуре организации хранения данных будет выбрана клиент-серверная СУБД, поскольку из предметной области следует требование к доступу к данным с многих компьютеров, а также достаточная скорость работы системы при малой нагрузке сети.

Из перечисенных выше СУБД под выбранные требования подходит Oracle и PostgreSQL. В данной работе в качестве СУБД будет выбрана PostgreSQL, поскольку она является системой с открытым исходным кодом, одновременно имея большую пользовательскую базу, в то время как Oracle не только является проприетарной системой, но и не действует на территории РФ.

1.3 Группы пользователей

Выделим группы пользователей разрабатываемой системы, исходя из предметной области поставленной задачи.

Выделяется 4 типа пользователей: системный администратор, главный врач, врач и работник регистратуры. Предусмотрим набор возможностей для

каждого типа пользователей.

Набор возможностей системного администратора:

— вход в систему;

— регистрация новых пользователей;

— настройка уровней доступа.

Набор возможностей главного врача:

— вход в систему;

— учёт пациента;

— учёт сотрудника;

— просмотр расписания врачей и кабинетов.

Набор возможностей врача:

— вход в систему;

— приём пациента;

— управление электронной медицинской картой пациента;

— просмотр расписания врачей и кабинетов.

Набор возможностей работника регистратуры:

— вход в систему;

— регистрация нового пациента;

— учёт пациента;

— просмотр расписания врачей и кабинетов.

Данные наборы возможностей сотрудников поликлиники можно объединить в диаграмму, приведённую на рисунке 2.

1.4 Структура базы данных

Для определения структуры базы данных необходимо на основе заданной предметной области определить категории данных, с которыми будет осуществлятья работа, определить информацию, которую будет содержать каждая категория, а также построить ER-диаграмму на основе полученных данных.

Рисунок 2 – Use-Case диаграмма для поставленной предметной области

1.4.1 Категории данных

Основываясь на анализе предметной области, можно выделить следующие категории данных:

— пользователь системы;

— пациент поликлиники;

— сотрудник поликлиники;

— должность в поликлинике;

— приём пациента;

— расписание;

— медицинская карта пациента;

— состояние пациента;

— договор с пациентом;

— паспорт;

— адрес.

Информация, соответствующая каждой категории, содержится в таблице 1

1.4.2 Диаграмма сущность-связь

При помощи диаграммы сущность-связь (*англ.* entity-relation diagram) можно выделить ключевые сущности, их атрибуты, а также типы и род связей между этими сущностями. Диаграмма сущность-связь для поставленной предметной области приведена на рисунке 3.

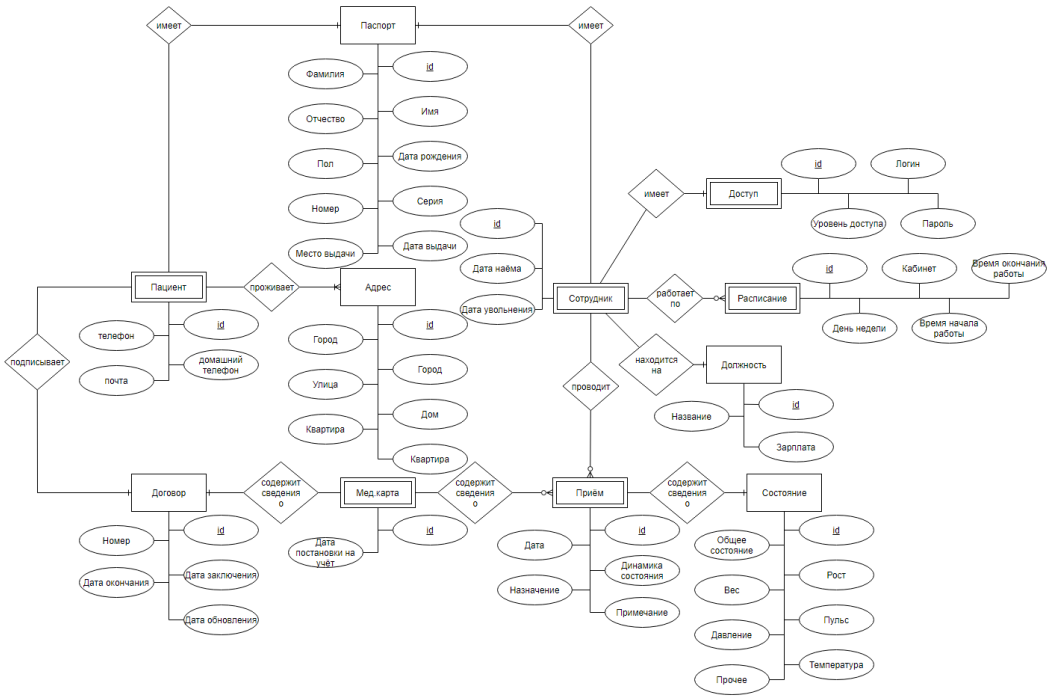

Рисунок 3 – ER-диаграмма для поставленной предметной области

Таблица 1 – Информация для выделенных категорий данных

Категория данных	Информация
Пользователь системы	логин, пароль, сотрудник, уровень доступа
Пациент поликлиники	паспорт, адрес, телефон, домашний телефон, почта
Сотрудник поликлиники	паспорт, должность, дата наёма, дата увольнения
Должность в поликлиники	название должности, зарплата
Приём пациента	врач, пациент, дата, динамика состояния, назначение, примечание, состояние
Расписание	сотрудник, день недели, кабинет, время начала работы, время окончания работы
Медицинская карта пациента	паспорт, договор, дата постановки на учёт
Состояние пациента	общее состояние, рост, вес, пульс, давление, температура, прочее
Договор с пациентом	номер договора, пациент, дата заключения, дата окончания, дата обновления
Паспорт	фамилия, имя, отчество, дата рождения, пол, серия, номер, дата выдачи, место выдачи
Адрес	страна, город, улица, дом, этаж, квартира

1.5 Общий вид архитектуры

Выбрав СУБД, которая будет использована для разработки, на основе анализа поставленной задачи опишем общий вид реализуемой архитектуры.

Архитектурой программного обеспечения (*англ.* software architecture) [5] — это форма, которая придаётся системе с целью упрощения разработки, развёртывания и сопровождения.

Для выбранной системы организации данных подходит клиент-серверная двузвенная архитектура (*англ.* client-server tw-tier architecture) [6], согласно которой сервер (процесс, реализующий некоторые сервисы, например, базу даных) предоставляет ресурсы набору соединённых с ним клиентов (процессы, запрашивающие сервисы у серверов путём посылки запроса и последующего получения ответа). Таким образом, совокупность клиентов и серверов вместе с промеждуточным программным обеспечение образуют единую систему, которая позволяет выполнять анализ и использование представленных данных.

В двузвенной архитектуре клиент и сервер могут как находиться на одной машине, так и на разных. Это обеспечивает сохранность и целостность данных, а также упрощает поддержку многопользовательского режима работы с общими данными.

Недостатком данной архитектуры является большой расход на поддержание работы приложений на рабочих станциях клиентов, что может существенно усложнить администрирование в больших системах. Тем не менее, для выбранной предметной области количество клиентов будет небольшим: рекомендованное Минздравом [2] количество должностей в поликлинике (55) является небольшим относительно того, какое количество клиентов обслуживают, например, современные социальные сети (согласно отчёту социальной сети ВКонтакте [7], количество российскийх пользователей в месяц составляет 73.4 млн. человек).

Диаграмма работы двухзвенной клиент-серверной архитектуры с серве-

ром СУБД приведён на рисунке 4.

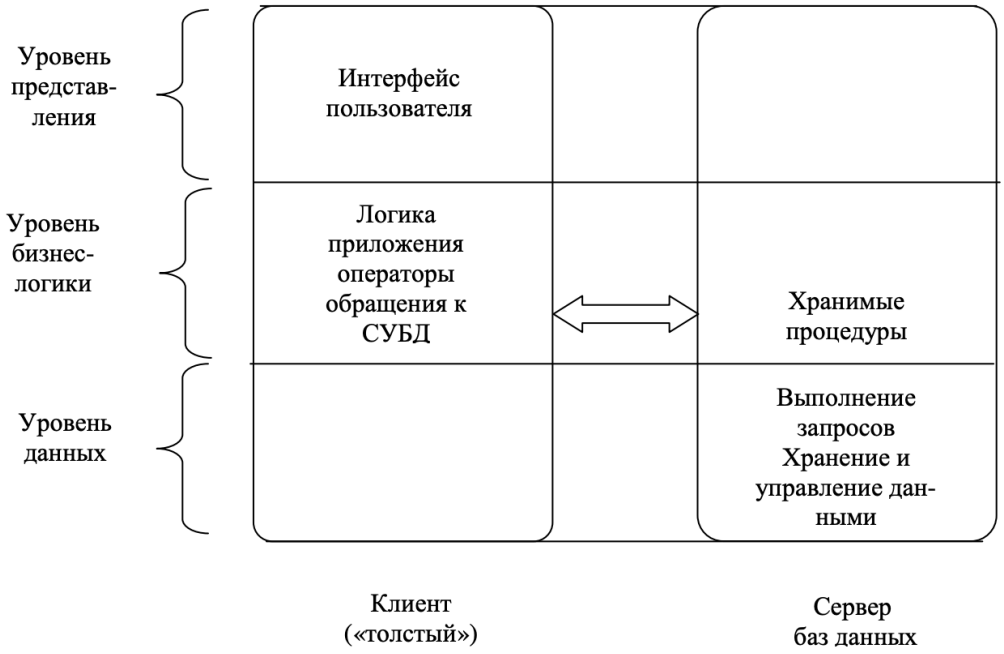

Рисунок 4 – ER-диаграмма для поставленной предметной области

Вывод

В данном разделе был проведён анализ предметной области (информационные системы для частных поликлиник), а также провдён анализ существующих СУБД. Как СУБД, при помощи которой будет решаться поставленная задача, была выбрана реляционная клиент-серверная СУБД PostgreSQL Как архитектуру разрабатываемой информационной системы была выбрана двузвенная клиент-серверная архитектура. Были приведены диаграммы сценариев использования информационной системы, проанализированы категории даных, приведена диаграмма сущность-связь, поясняющая суть связей между выделенными сущностями, соответствующими категориям данных.

2 Конструкторская часть

В данном разделе рассматривается спроектированная база данных, соответствующая выбранной в аналитической части СУБД, приводится диаграмма базы данных, рассматриваются сценарии создания таблиц БД и основные используемые паттерны проектирования.

2.1 Таблицы БД

Для выделенных в аналитической части категорий данных и соответствующей им информации, приведённой в таблице 1, база данных должна содержать следующие таблицы:

— пациенты поликлиники — patients;

— сотрудники поликлиники — staff;

— должности в поликлинике — posts;

— приёмы пациентов — sessions;

— расписания — schedules;

— медицинские карты пациентов — records;

— состояния пациентов — states;

— договоры с пациентами — agreements;

— паспорты — passports;

— доступы — accesses;

— адреса — addresses.

Используя знания о выбранной СУБД (PostgreSQL) и приведённую на рисунке 3 диаграмму сущность-связь, определим для каждой из перечисленных выше таблиц столбцы, их типы и ограничения, в таблицах 2 – 10.

Определим пользоательские типы данных для дня недели, пола и уровня доступа в листинге 1.

Таблица 2 – Информация о столбцах таблицы пациентов поликлиники

Столбец	Тип данных	Ограничения	Значение
patient_id	SERIAL	NOT NULL, PRIMARY KEY	ID пациента
address_id	INT	NOT NULL, FOREIGH KEY	ID адреса
passport	INT	NOT NULL, FOREIGH KEY	ID паспорта
phone	VARCHAR(10)	NOT NULL	Номер телефона
home_phone	VARCHAR(10)	—	Домаший номер телефона
email	VARCHAR(20)	—	Адрес электронной почты

Таблица 3 – Информация о столбцах таблицы сотрудников поликлиники

Столбец	Тип данных	Ограничения	Значение
staff_id	SERIAL	NOT NULL, PRIMARY KEY	ID сотрудника
passport	INT	NOT NULL, FOREIGH KEY	ID паспорта
post	INT	NOT NULL, FOREIGH KEY	ID должности
employment_date	DATE	NOT NULL	Дата наёма
dismissal_date	DATE	—	Дата увольнения

Таблица 4 – Информация о столбцах таблицы должностей в поликлинике

Столбец	Тип данных	Ограничения	Значение
post_id	SERIAL	NOT NULL, PRIMARY KEY	ID должности
title	VARCHAR(30)	NOT NULL	Название должности
salary	INT	NOT NULL	Зарплата

Таблица 5 – Информация о столбцах таблицы приёмов пациентов

Столбец	Тип данных	Ограничения	Значение
session_id	SERIAL	NOT NULL, PRIMARY KEY	ID приёма
doctor_id	INT	NOT NULL, FOREIGH KEY	ID принимающего врача
patient_id	INT	NOT NULL, FOREIGH KEY	ID пациента
record_id	INT	NOT NULL, FOREIGH KEY	ID медицинской карты
state_id	INT	NOT NULL, FOREIGH KEY	ID состояния
session_date	DATE	—	Дата приёма
dynamics	VARCHAR(256)	—	Динамика состояния
prescription	VARCHAR(256)	—	Назначение
note	VARCHAR(256)	—	Примечание

Таблица 6 – Информация о столбцах таблицы расписаний

Столбец	Тип данных	Ограничения	Значение
schedule_id	SERIAL	NOT NULL, PRIMARY KEY	ID расписания
staff_id	INT	NOT NULL, FOREIGH KEY	ID сотрудника
workstart	TIMESTAMP	NOT NULL	Дата начала работы
workend	TIMESTAMP	NOT NULL	Дата окончания работы
week_day	week_day_t	NOT NULL	День недели
office	INT	—	Кабинет

Таблица 7 – Информация о столбцах таблицы медицинских карт пациентов

Столбец	Тип данных	Ограничения	Значение
record_id	SERIAL	NOT NULL, PRIMARY KEY	ID мед. карты
agreement_id	INT	NOT NULL	ID договора
registration_date	DATE	NOT NULL	Дата постановки на учёт

Таблица 8 – Информация о столбцах таблицы состояний пациентов

Столбец	Тип данных	Ограничения	Значение
state_id	SERIAL	NOT NULL, PRIMARY KEY	ID состояния
general_condition	VARCHAR(256)	—	Общее состояние
height	INT	—	Рост пациента, см
patient_weight	INT	—	Вес пациента, кг
pulse	INT	—	Частота пульса в минуту
pressure	VARCHAR(50)	—	Артериальное давление
temperature	REAL	—	Температура, градусы Цельсия
other	VARCHAR(256)	—	Прочее

Таблица 9 – Информация о столбцах таблицы договоров

Столбец	Тип данных	Ограничения	Значение
agreement_id	SERIAL	NOT NULL, PRIMARY KEY	ID договора
patient_id	SERIAL	NOT NULL, FOREIGN KEY	ID пациента
code	VARCHAR(50)	NOT NULL	Номер договора
conclusion_date	DATE	NOT NULL	Дата заключения договора
expiration_date	DATE	NOT NULL	Дата окончания договора
renewal_date	DATE	NOT NULL	Дата обновления договора

Таблица 10 – Информация о столбцах таблицы адресов

Столбец	Тип данных	Ограничения	Значение
address_id	SERIAL	NOT NULL, PRIMARY KEY	ID адреса
country	VARCHAR(50)	NOT NULL	Страна
city	VARCHAR(50)	NOT NULL	Город
street	VARCHAR(50)	NOT NULL	Улица
house	VARCHAR(50)	NOT NULL	Дом
flat	VARCHAR(50)	—	Квартира

Таблица 11 – Информация о столбцах таблицы паспортов

Столбец	Тип данных	Ограничения	Значение
passport_id	SERIAL	NOT NULL, PRIMARY KEY	ID паспорта
surname	VARCHAR(50)	NOT NULL	Фамилия
middlename	VARCHAR(50)	NOT NULL	Имя
lastname	VARCHAR(50)	—	Отчество
birth_date	DATE	NOT NULL	Дата рождения
gender	gender_t	NOT NULL	Пол
series	VARCHAR(10)	NOT NULL	Серия
num	VARCHAR(10)	NOT NULL	Номер
issue_date	DATE	NOT NULL	Дата выдачи
issue_location	VARCHAR(128)	NOT NULL	Место выдачи

Таблица 12 – Информация о столбцах таблицы доступов

Столбец	Тип данных	Ограничения	Значение
access_id	SERIAL	NOT NULL, PRIMARY KEY	ID доступов
staff_id	INT	NOT NULL, FOREIGN KEY	ID сотрудника
username	VARCHAR(50)	NOT NULL	Логин
passwordhash	BYTEA	NOT NULL	Хэш пароля
access_level	access_level_t	NOT NULL	Уровень доступа

Листинг 1 – Определение пользовательских типов данных

```
CREATE TYPE week_day_t as ENUM('Sunday', 'Monday', 'Tuesday',
    'Wednesday', 'Thursday', 'Friday', 'Saturday');

CREATE TYPE gender_t as ENUM('Male', 'Female', 'Other');

CREATE TYPE access_level_t as ENUM('admin', 'chief', 'doctor',
    'registry');
```

2.1.1 Создание таблиц и ограничений

Результат переноса приведённых выше таблиц в скрипты создания страниц, написанных на SQL, можно увидеть на листингах 2 – 6.

Листинг 2 – Создание таблиц (часть 1)

```
CREATE TABLE patients (
    patient_id SERIAL NOT NULL PRIMARY KEY,
    address_id INT NOT NULL,
    passport INT NOT NULL,
    phone VARCHAR(10) NOT NULL,
    home_phone VARCHAR(10),
    email VARCHAR(20)
);
```

Листинг 3 – Создание таблиц (часть 2)

```
CREATE TABLE staff (
    staff_id SERIAL NOT NULL PRIMARY KEY,
    passport INT NOT NULL,
    post INT NOT NULL,
    employment_date DATE NOT NULL,
    dismissal_date DATE
);

CREATE TABLE posts (
    post_id SERIAL NOT NULL PRIMARY KEY,
    title VARCHAR(30) NOT NULL,
    salary INT NOT NULL
);

CREATE TABLE sessions (
    session_id SERIAL NOT NULL PRIMARY KEY,
    doctor_id INT NOT NULL,
    patient_id INT NOT NULL,
    record_id INT NOT NULL,
    state_id INT NOT NULL,
    session_date DATE NOT NULL,
    dynamics VARCHAR(256),
    prescription VARCHAR(256),
    note VARCHAR(256)
);
```

Листинг 4 – Создание таблиц (часть 3)

```
CREATE TABLE schedules (
    schedule_id SERIAL NOT NULL PRIMARY KEY,
    staff_id INT NOT NULL,
    workstart TIMESTAMP NOT NULL,
    workend TIMESTAMP NOT NULL,
    week_day week_day_t NOT NULL,
    office INT
);

CREATE TABLE records (
    record_id SERIAL NOT NULL PRIMARY KEY,
    agreement_id INT NOT NULL,
    registration_date DATE NOT NULL
);

CREATE TABLE states (
    state_id SERIAL NOT NULL PRIMARY KEY,
    general_condition VARCHAR(256),
    height INT,
    patient_weight INT,
    pulse INT,
    pressure VARCHAR(50),
    temperature REAL,
    other VARCHAR(256)
);
```

Листинг 5 – Создание таблиц (часть 4)

```
CREATE TABLE agreements (
    agreement_id SERIAL NOT NULL PRIMARY KEY,
    patient_id INT NOT NULL,
    code VARCHAR(50) NOT NULL,
    conclusion_date DATE NOT NULL,
    expiration_date DATE NOT NULL,
    renewal_date DATE NOT NULL
);

CREATE TABLE addresses (
    address_id SERIAL NOT NULL PRIMARY KEY,
    country VARCHAR(50) NOT NULL,
    city VARCHAR(50) NOT NULL,
    street VARCHAR(50) NOT NULL,
    house VARCHAR(50) NOT NULL,
    flat VARCHAR(50)
);

CREATE TABLE passports (
    passport_id SERIAL NOT NULL PRIMARY KEY,
    surname VARCHAR(50) NOT NULL,
    middlename VARCHAR(50) NOT NULL,
    lastname VARCHAR(50),
    birth_date DATE NOT NULL,
    gender gender_t NOT NULL,
    series VARCHAR(10) NOT NULL,
    num VARCHAR(10) NOT NULL,
    issue_date DATE NOT NULL,
    issue_location VARCHAR(128) NOT NULL
);
```

Листинг 6 – Создание таблиц (часть 5)

```
CREATE TABLE accesses (
    access_id SERIAL NOT NULL PRIMARY KEY,
    staff_id INT NOT NULL,
    username VARCHAR(50) NOT NULL,
    passwordhash BYTEA NOT NULL,
    access_level access_level_t NOT NULL
);
```

Также в созданные с применением перечисленных скриптов таблицы необходимо добавить ограничения. Это можно сделать при помощи скрипта, приведённого на листингах 7 – 9.

Листинг 7 – Создание ограничений для созданных таблиц (часть 1)

```
ALTER TABLE patients ADD CONSTRAINT addr_constraint
    FOREIGN KEY (address_id) REFERENCES addresses(address_id)
    ON DELETE CASCADE;

ALTER TABLE accesses ADD CONSTRAINT agreement_patient_constraint
    FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
    ON DELETE CASCADE;

ALTER TABLE patients ADD CONSTRAINT pat_passport_constraint
    FOREIGN KEY (passport) REFERENCES passports(passport_id)
    ON DELETE CASCADE;
```

Листинг 8 – Создание ограничений для созданных таблиц (часть 2)

```
ALTER TABLE staff ADD CONSTRAINT staff_passport_constraint
    FOREIGN KEY (passport) REFERENCES passports(passport_id)
    ON DELETE CASCADE;

ALTER TABLE staff ADD CONSTRAINT staff_post_constraint
    FOREIGN KEY (post) REFERENCES posts(post_id)
    ON DELETE CASCADE;

ALTER TABLE sessions ADD CONSTRAINT session_doctor_contraint
    FOREIGN KEY (doctor_id) REFERENCES staff(staff_id)
    ON DELETE CASCADE;

ALTER TABLE sessions ADD CONSTRAINT session_patient_contraint
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
    ON DELETE CASCADE;

ALTER TABLE sessions ADD CONSTRAINT session_record_constraint
    FOREIGN KEY (record_id) REFERENCES records(record_id)
    ON DELETE CASCADE;

ALTER TABLE sessions ADD CONSTRAINT session_state_constraint
    FOREIGN KEY (state_id) REFERENCES states(state_id)
    ON DELETE CASCADE;

ALTER TABLE schedules ADD CONSTRAINT schedule_staff_constraint
    FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
    ON DELETE CASCADE;
```

Листинг 9 – Создание ограничений для созданных таблиц (часть 3)

```
ALTER TABLE agreements ADD CONSTRAINT agr_patient_constraint
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
    ON DELETE CASCADE;
```

2.1.2 Создание ролевой модели и управление доступами

На основе приведённой выше Use-Case диаграммы создадим роли и распределим доступы к таблицам.

Листинг 10 – Создание ролей

```
CREATE ROLE admin;
CREATE ROLE chief;
CREATE ROLE doctor;
CREATE ROLE registry;
```

Листинг 11 – Распределение доступов между ролями (часть 1)

```
GRANT all ON all tables IN SCHEMA public TO admin;

GRANT all ON sessions TO chief;
GRANT all ON staff TO chief;
GRANT all ON states TO chief;
GRANT all ON posts TO chief;
GRANT all ON patients TO chief;
GRANT all ON passports TO chief;
GRANT all ON agreements TO chief;
```

Листинг 12 – Распределение доступов между ролями (часть 2)

```
GRANT all ON addresses TO chief;
GRANT all ON records TO chief;
GRANT all ON schedules TO chief;

GRANT all ON sessions TO doctor;
GRANT all ON records TO doctor;
GRANT all ON states TO doctor;
GRANT select ON patients TO doctor;
GRANT select ON staff TO doctor;
GRANT select, update ON schedules TO doctor;

GRANT all ON records TO registry;
GRANT all ON schedules TO registry;
GRANT all ON patients TO registry;
GRANT all ON passports TO registry;
GRANT select ON posts TO registry;
GRANT select ON staff TO registry;
```

Для управления доступами для конкретных соединений была использована технология политики RLS (*англ.* Row Level Security Polices) и процедура, приведённая на листингах 13 – 15.

Листинг 13 – Управление доступами при помощи RLS (часть 1)

```
ALTER TABLE accesses
    ENABLE ROW LEVEL SECURITY;
```

Листинг 14 – Управление доступами при помощи RLS (часть 2)

```
CREATE VIEW session AS
SELECT current_setting('app.user_uuid')::int AS user_uuid;

GRANT all ON session TO admin, chief, doctor, registry;

CREATE POLICY users_select
    ON accesses
    FOR SELECT
    USING (true);

CREATE POLICY users_update
    ON accesses
    FOR UPDATE
    TO doctor, chief, registry
    USING (access_id = (select user_uuid
                   from session));

CREATE POLICY users_update_admin
    ON accesses
    FOR UPDATE
    TO admin
    USING (true);

CREATE POLICY users_delete_admin
    ON accesses
    FOR DELETE
    TO admin
    USING (true);
```

Листинг 15 – Управление доступами при помощи RLS (часть 3)

```
CREATE OR REPLACE PROCEDURE before_each_query(IN user_uuid int)
    LANGUAGE plpgsql
AS
$$
DECLARE
    role_name regrole;
BEGIN
    role_name = (SELECT role FROM users WHERE
        accesses.access_id = user_uuid)::regrole;
    execute format('SET role %I', role_name);
    execute format('SET app.user_uuid = %L', user_uuid);
END;
$$;
```

2.1.3 Диаграмма базы данных

По полученным выше таблицам, отображающим свойства столбцов таблиц базы данных, построим диаграмму базы данных (*англ. database schema*) при помощи языка разметки баз данных (*англ.* Database Markup Language, DML), предназначенного для представления данных и информации о таблицах, взятых из реляционной базы данных,

Диаграмма базы данных для поставленной задачи отображена на рисунке 5.

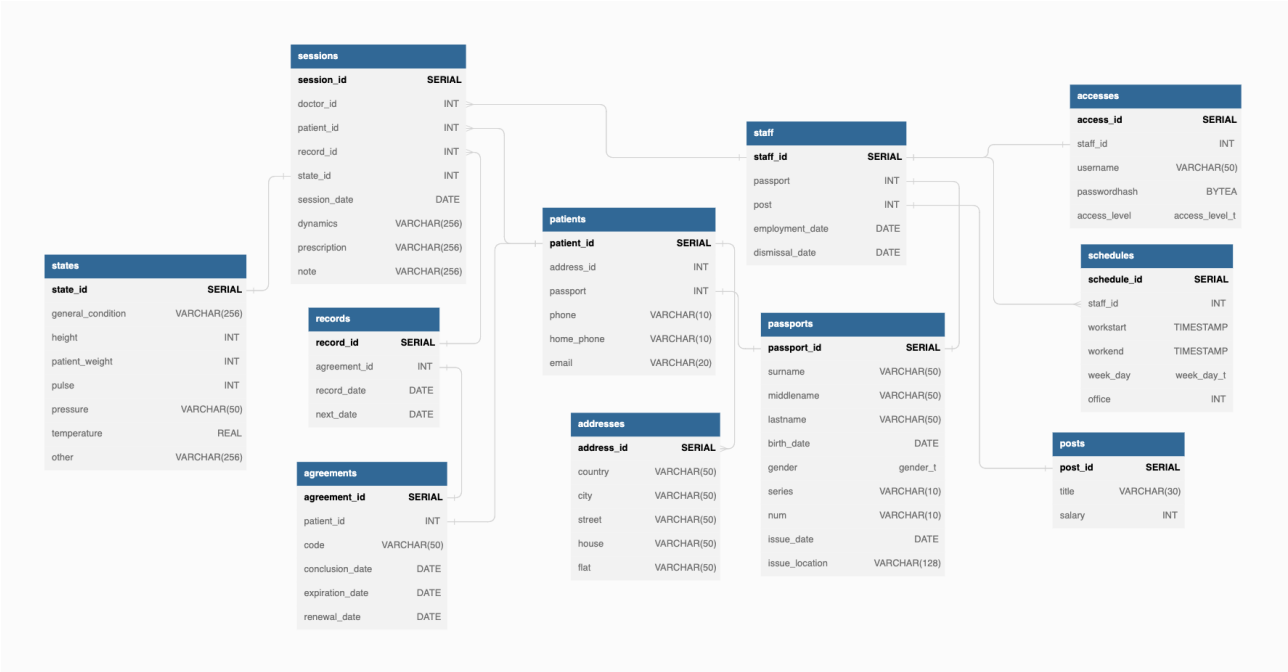

Рисунок 5 – Диаграмма базы данных для поставленной задачи

2.2 Основные паттерны проектирования

В данной работе будут использованы два паттерна, рассмотренные ниже — архитектурный паттерн MVC и паттерн проектирования REST.

2.2.1 MVC

MVC (*англ.* Model View Controller) [8] — архитектурный паттерн, в основе которого лежат три понятия: модель, представление и контроллер.

Модель содержит всё информацию состояния, данные и логику приложения, необходимую для ведения базы данных. Она представляет собой интерфейс для получения и изменения состояния, а также может отправлять оповещения об изменениях состояния наблюдателем,

Представление определяет собой отображение модели, и, как правило, получает состояние и данные для отображения непосредственно от модели.

Контроллер получает данные, вводимые пользователем, определяет их смысл для модели, выполняет операции с моделью и управляет её бизнес-логикой, связывая с представлением.

Упрощённо можно привести следующие этапы работы приложения, к которым может привести взаимодействие с пользователем:

1) пользователь взаимодействует с моделью: представление сообщает контроллеру о том, какая была выполнена операция, контроллер должен обработать это действие;
2) контроллер обращается к модели с запросами об изменении состояния: контроллер должен интерпретировать действия пользователя и выполнить необходимые операции с моделью;
3) котроллер обращается к представлению с запросом об изменении: когда контроллер получает действие от представления, он может в результате его обработки обратиться к представлению с некоторым запросом на изменение (например, заблокировать некоторые кнопки в интерфейсе);
4) модель оповещает представление об изменении состояния (вследствие действий пользователя состояние модели может измениться, о чём модель может уведомить представление);
5) представление заправшивает у модели информацию состояния: для корректного отображения данных представление может заправшивать текущее состояние (за которое отвечает модель).

2.2.2 REST

REST (*англ.* Representational State Transfer, передача репрезентативного состояния) — паттерн проектирования, в основе которого лежит понятие ресурса и его состояния. Каждый ресурс имеет URI (unified resourse identifier), универсальный идентификатор, который позволяет совершать различные действия над данным ресурсом, например, операции создания, чтения, обновления и удаления (CRUD — create, read, update, delete).

Архитектура REST поддерживает выбранную модель клиент-сервер, причём не требует от сервера хранения какой-либо информации о состоянии клиента в периодах между запросами (всю необходимую для выполения операции информацию клиент передаёт во время запроса). Поддерживается возможность

кэширования ответов сервера на клиенте с целью ускорения быстродействия программы клиента, что помогает уменьшить нагрузку на клиентских машинах.

При этом программный интерфейс приложения (*англ.* API, application programming interface), которым пользуется клиент для осуществления запросов на сервер, может быть унифицирован. В этом случае стараются придерживаться трёх основных принципов:

1) однозначное идентифицирование ресурсов в запросе;
2) использование представления для взаимодействия с ресурсами (представление определяет текущее или желаемое состояние ресурса);
3) достаточное количество информации в каждом запросе: каждый отдельно взятый запрос содержит информацию, однозрачно определяющее то, как необходимо обрабатывать и идентифицировать данный запрос.

2.3 Диаграмма классов приложения

Приложение строится согласно диаграмме классов, приведённой на рисунке 6.

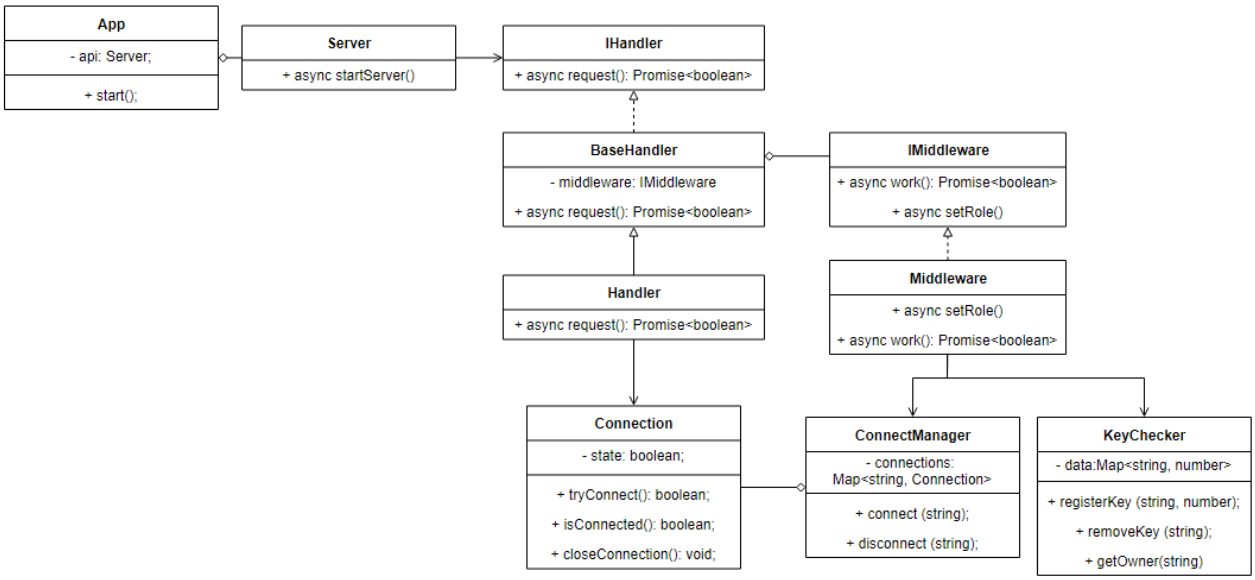

Рисунок 6 – Диаграмма классов

Вывод

В данном разделе была рассмотрена спроектированная база данных, соответствующая выбранной в аналитической части СУБД, приведена диаграмма базы данных, рассмотрены сценарии создания таблиц БД и ролевой модели для управления доступом к таблицам, а также приведена диаграмма классов приложения и разобраны основные используемые паттерны проектирования: MVC и REST.

3 Технологическая часть

В данном разделе рассмотрены средства разработки программного обеспечения, приведены детали реализации и листинги исходных кодов программы, а также приведён пример результата работы программы.

3.1 Средства реализации

Как основное средство реализации и разработки ПО был выбран язык программирования TypeScript [10]. Причиной выбора данного языка является тот факт, что он имеет компилируется в JavaScript [11], что делает его также кроссплатформенным и позволяет использовать библиотеки, написанные для JavaScript. Наличие типизации позволяет траспилятору получать более эффективный код, чем для JavaScript, вследствие чего производительность TypeScript является предпочтительнее при написании сервера [12].

Для запуска написанного сервера была использована библиотека Node.js [13] для языка JavaScript, которая предоставляет асинхронное окружение, что позволяет избавиться от синхронности и разрабатывать более производительные приложения.

Для обработки HTTP запросов была использована библиотека Fastify [14], которая была выбрана из-за своего превосходства в скорости обработки соединений по сравнению с другими библиотеками (Koa, Express, Restify, Hapi) [15]. Для доступа к СУБД, реализованной при помощи PostgreSQL [16], была использована библиотека pg-promise [17], предоставляющая интерфейс для взаимодействия с базой данных. Средой разработки послужил графический редактор Visual Studio Code [18], который известен содержанием большого количество плагинов, ускоряющих процесс разработки программы.

3.2 Реализация сервера

В расположенных ниже листингах 16 – 20 приведены реализации менеджера соединений и класса соединения, объекты которого используются в обработчиках запросов.

Листинг 16 – Реализация менеджера соединений (часть 1)

```
import dbConfig from '../configs/db.config.json';
import Connection from './Connection';

class ConnectManager {
    private connections: Map<string, any>;
    private connInfo: any;

    constructor() {
        this.connections = new Map();
        this.connInfo = dbConfig;
    }

    connect(connectionName: string) {
        const newConnection = new Connection(this.connInfo);
        let check = false;
        let i = 0;
        while (i < this.connInfo.repeat && !check) {
            check = newConnection.tryConnect();
            i++;
        }
        if (!check) {
            return null;
        }
        this.connections.set(connectionName, newConnection);
        return this.connections.get(connectionName);
    }
```

Листинг 17 – Реализация менеджера соединений (часть 2)

```
1
2   disconnect(connectionName: string) {
3       this.connections.get(connectionName).closeConnection();
4       this.connections.delete(connectionName);
5   }
6   }
7
8   export const connectManager = new ConnectManager();
```

Листинг 18 – Реализация класса соединения (часть 1)

```
1   import logger from '../logger';
2   import pgPromise from 'pg-promise';
3
4   export default class Connection {
5       private state: boolean;
6       private connectionInfo: any;
7       private connection: any;
8
9
10      constructor(connInfo: any) {
11          this.connectionInfo = connInfo;
12          this.state = false;
13      }
14
15      tryConnect(): boolean {
16          const promise = pgPromise();
17          const connectionURL =
```

Листинг 19 – Реализация класса соединения (часть 2)

```
            `${this.connectionInfo.dbName}://` +
            `${this.connectionInfo.user}:` +
            `${this.connectionInfo.password}@` +
            `${this.connectionInfo.url}:` +
            `${this.connectionInfo.port}/` +
            `${this.connectionInfo.db}`;
        try {
            this.connection = promise(connectionURL);
            this.state = true;
        } catch (e) {
            logger.error(e);
            return false;
        }
        return true;
    }

    isConnected(): boolean {
        return this.state;
    }

    closeConnection(): void {
        this.connection.$pool.end();
    }

    async one(params: any) {
        if (this.state) {
            return await this.connection.one(params);
        }
        return null;
    }
```

Листинг 20 – Реализация класса соединения (часть 3)

```
            `${this.connectionInfo.dbName}://` +
            `${this.connectionInfo.user}:` +
            `${this.connectionInfo.password}@` +
            `${this.connectionInfo.url}:` +
            `${this.connectionInfo.port}/` +
            `${this.connectionInfo.db}`;
        try {
            this.connection = promise(connectionURL);
            this.state = true;
        } catch (e) {
            logger.error(e);
            return false;
        }
        return true;
    }

    isConnected(): boolean {
        return this.state;
    }

    closeConnection(): void {
        this.connection.$pool.end();
    }

    async one(params: any) {
        if (this.state) {
            return await this.connection.one(params);
        }
        return null;
    }
```

Вывод

В данном разделе были рассмотрены средства реализации ПО, приведены листинги исходного кода программы, разработанной на основе алгоритмов, выбранных в аналитическом разделе и соответствующих приведённой в конструкторской части диаграмме классов приложения.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы разработанной программы и поставлен эксперимент по сравнению производительности в зависимости от сложности моделируемой сцены. Сложность оценивается количеством используемых на сцене поверхностей.

4.1 Результаты разработки

4.1.1 Описание интерфейса

В верхнем правом углу программы располагается панель управления, позволяющая контролировать созданные объекты, например, вращать их, масштабировать или перемещать (пример такой панели для куба, сферы и цилиндра привелён на рисунке 7).

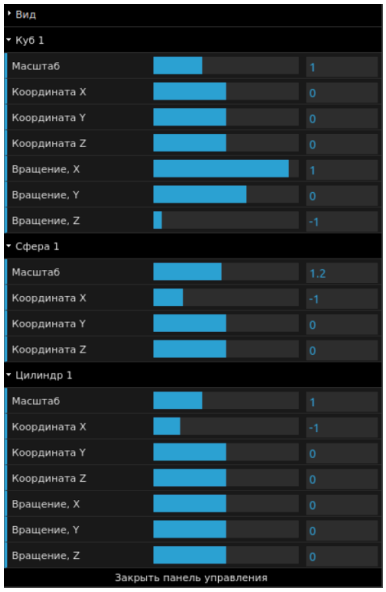

Рисунок 7 – Пример панели управления для куба, сферы и цилиндра

В нижнем правом углу располагаются кнопки, позволяющие пользователю перейти к созданию нового объекта или к удалению последнего созданного (рисунок 8).

Разработанная программа позволяет пользователю создавать твердотельные модели на основе примитивов (куб, цилиндр, сфера и тор) и использовать логические операции (объединение, пересечение, вычитание). Присутствует воз-

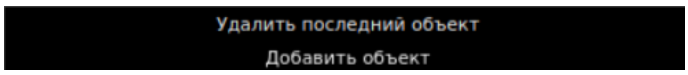

Рисунок 8 – Кнопки добавления нового объекта и удаления последнего созданного

можность добавлять на сцену объекты, удалять их и преобразовывать, а также менять позицию камеры.

4.1.2 Демонстрация работы программы

На рисунке 9 продемонстрирована рабоота программы на примере вычитания куба из сферы и объединения с цилиндром.

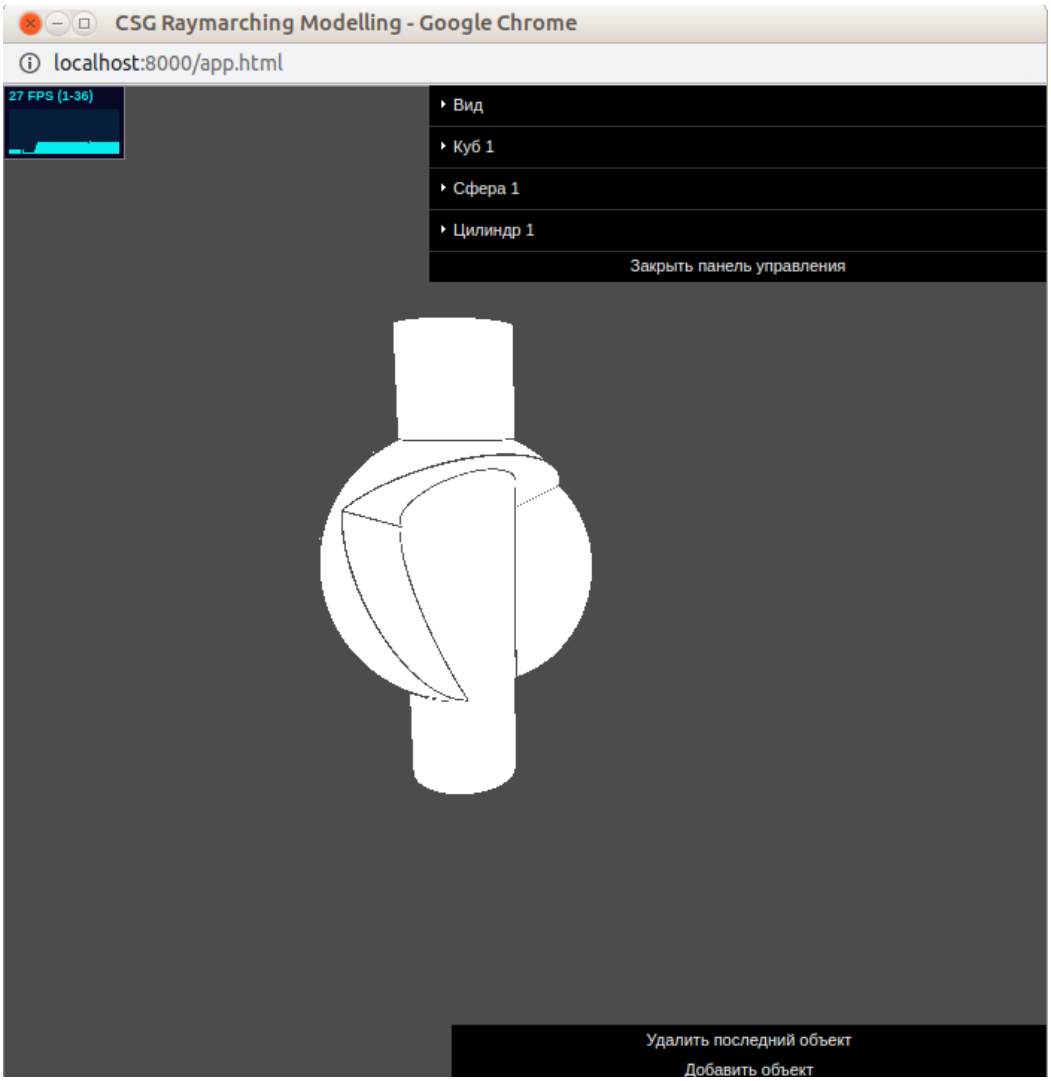

Рисунок 9 – Демонстрация работы программы (композиция моделей куба, сферы и цилиндра)

На рисунке 10 продемонстрирована работа программы на примере пересечения куба с тором и объединения с цилиндром.

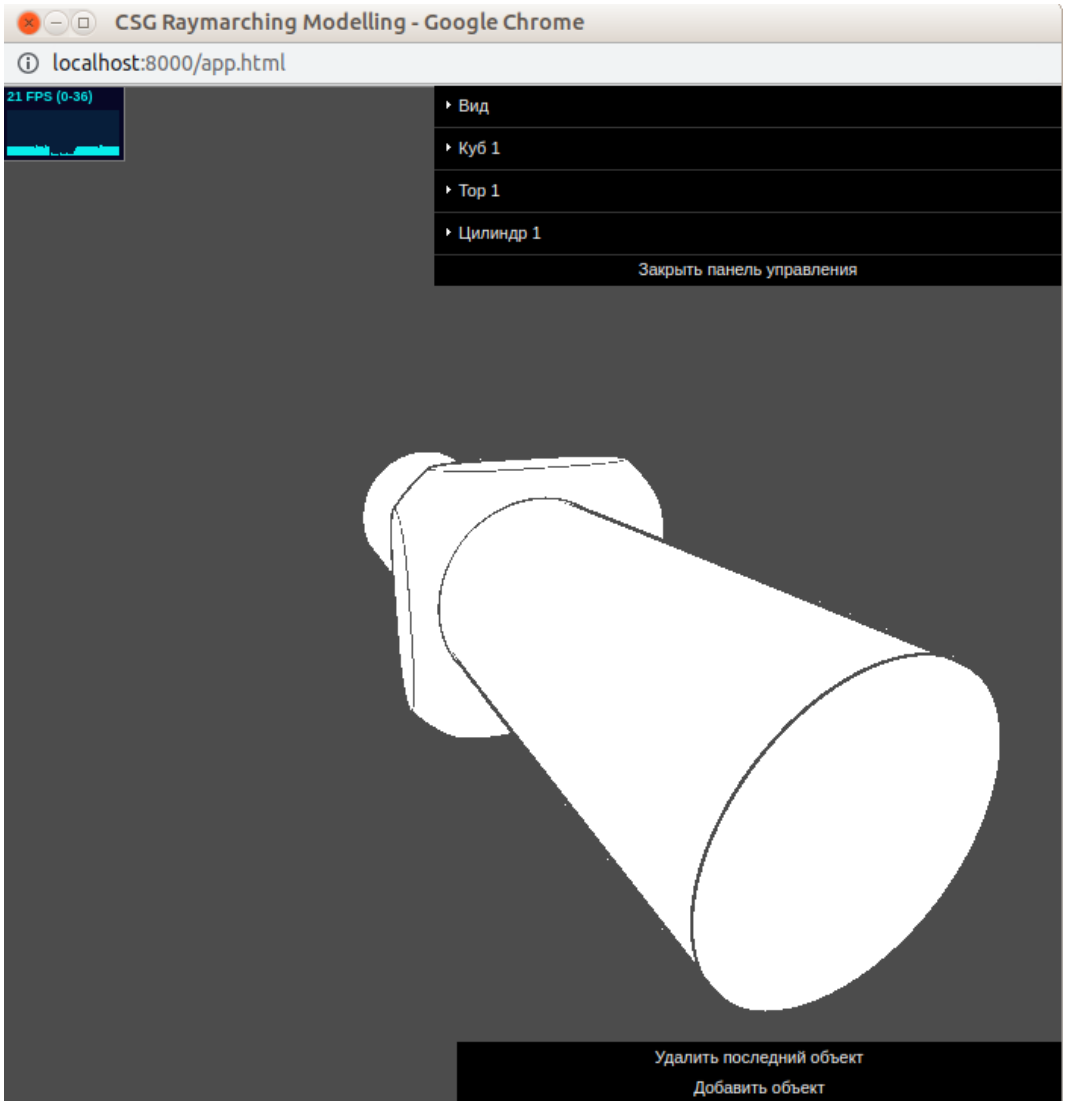

Рисунок 10 – Демонстрация работы программы (композиция моделей куба, тора и цилиндра)

4.2 Постановка эксперимента

Целью эксперимента является сравнение производительности приложения при использовании встроенной и дискретной видеокарт. Производительность будет оцениваться с помощью измерения количества кадров в секунду (FPS), с которами работает приложение. Нагрузка будет меняться в зависимости от количества объектов, расположенных на сцене.

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система: Ubuntu 21.04 [**?**] 64-bit;

— Память: 16 Гб;

— Процессор: Intel Core™ i5-8300H [**?**] с тактовой частотой 2.30 ГГц;

— Видеокарты:

  — AMD Radeon (TM) VEGA 8 Graphics (встроенная) [**?**];

  — NVIDIA GeForce GTX 1080 (дискретная) [**?**].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и системным окружением операционной системы.

4.3 Сравнение производительности

Результаты сравнения производительности сцен разной нагруженности приведены в таблице 13.

Таблица 13 – Сравнение FPS при запуске приложения на встроенной и дискретной видеокартах.

Количество объектов	FPS-NVIDIA	FPS-AMD
1	60	24
2	55	18
3	48	10
4	36	8
5	32	6

Результаты тестирования приводят к следующим выводам:

1) с ростом количества объектов, количество FPS уменьшается (в лучшем случае 60 кадров (24 на встроенной), в худшем 32 кадра (6 на встроенной));
2) дискретная видеокарта позволяет получить большее количество кадров, которое приемлемо для использования приложения в режиме реального времени (30+ FPS), это объясняется наличием встроенной памяти (2 ГБ) и большей мощностью в сравнении с встроенной, которая менее производительна, а также использует разделяемую память (часть оперативной);
3) встроенной видеокарты недостаточно для выполнения программы в режиме реального времени - приемлемое FPS должно быть около 30 и более, встроенная (в лучшем случае - 1 объект на сцене) позволяет получить лишь 24 FPS — для режима реального времени недостаточно;
4) в среднем дискретная видеокарты производительнее встроенной в 4 раза.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Государственная медицина: в погоне за качеством [Электронный ресурс]. — Режим доступа: https://wciom.ru/analytical-reviews/analiticheskii-obzor/gosudarstvennaja-medicina-v-pogone-za-kachestvom, свободный (дата обращения: 14.04.2023).

2. Рекомендуемые штатные нормативы поликлиники [Электронный ресурс]. — Режим доступа: https://base.garant.ru/70195856/f52b32b623103013c77c8c319c288f45/, свободный (дата обращения: 14.04.2023).

3. Карпова И. П., Введение в базы данных. Учебное пособие,. — Москва: Московский государственный институт электроники и математики (Технический университет), 2009.

4. Карпова Т. С., Базы данных: модели, разработка, реализация, — СПб.: Питер, 2001. –304 с..

5. Мартин Р., Чистая архитектура. Искусство разработки программного обеспечения — СПб.: Питер, 2018. – 352 с..

6. Рыбальченко М. В., Архитектура информационных систем: учебное пособие. Ч.1. — Таганрог: Издательство ЮФУ, 2015. – 92 с.

7. ВКонтакте: итоги первого квартала 2022 года [Электронный ресурс]. — Режим доступа: https://vk.com/press/q1-2022-results, свободный (дата обращения: 15.04.2023).

8. Фримен Э., Робсон Э., Сьерра К., Бейтс Б., Паттерны проектирования. Обновлённое юбилейное издание. — СПб.: Питер, 2018. – 656 с..

9. Дергачев А. М., Кореньков Ю. Д., Логинов И. П., Сафронов А. Г., Технологии веб-сервисов. — СПб.: Университет ИТМО, 2021. – 100с..

10. TypeScript — official site [Электронный ресурс]. — Режим доступа: https://www.typescriptlang.org/ (дата обращения: 17.05.2023).

11. JavaScript — official site [Электронный ресурс]. — Режим доступа: https://www.javascript.com/ (дата обращения: 17.05.2023).

12. Typescript VS Javascript benchmarks [Электронный ресурс]. — Режим доступа: https://programming-language-benchmarks.vercel.app/typescript-vs-javascript (дата обращения: 17.05.2023).

13. Node.js [Электронный ресурс]. — Режим доступа: https://nodejs.org/ (дата обращения: 17.05.2023).

14. Fastify — official site [Электронный ресурс]. — Режим доступа: https://www.fastify.io/, свободный (дата обращения: 17.05.2023).

15. Fastify Benchmarks — official site [Электронный ресурс]. — Режим доступа: https://www.fastify.io/benchmarks/ (дата обращения: 17.05.2023).

16. PostgreSQL — official site [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/, свободный (дата обращения: 17.05.2023).

17. pg-promise — Source Code [Электронный ресурс]. — Режим доступа: https://github.com/vitaly-t/pg-promise, свободный (дата обращения: 17.05.2023).

18. Visual Studio Code — Code Editing. Redefined [Электронный ресурс]. — Режим доступа: https://code.visualstudio.com, свободный (дата обращения: 17.05.2023).