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 статического сервера»

Студент группы **ИУ7-76Б** _____ **П. А. Калашков**

(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ _____

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 49 с., 9 рис., 12 табл., 22 ист.

TODO

БАЗЫ ДАННЫХ, PostgreSQL, РЕЛЯЦИОННАЯ МОДЕЛЬ, REST

Цель работы — разработка информационной системы для частных поликлиник.

Моделью базы данных является реляционная модель, архитектура приложения — двухзвенная клиент-серверная. Для представления API используется паттерн REST.

Результаты: разработана программа, предназначенная для использования сотрудниками частной поликлиники. Предусмотрены различные роли, соответствующие должностям минимального штата поликлиники, возможность администрирования приложения и работе с информацией о пациентах и приёмах.

Проведено исследование зависимости времени обработки запроса от количества полей таблицы. Из результатов следует, что зависимость времени обработки запроса от количества полей таблицы имеет линейный характер.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

TODO

В работе используются следующие термины с соответствующими определениями:

HTML — Hypertext Markdown Language, язык гипертексовой разметки

CSS — Cascading Style Sheet, каскадные таблицы стилей

JS — JavaScript

PNG — Portable Network Graphics

JPEG (JPG) — Joint Photographic Experts Group

SWF — Small Web Format

GIF — Graphics Interchange Format

HTTP — Hypertext Transfer Protocol, протокол передачи гипертекста

ВВЕДЕНИЕ

По данным DatePortal, к январю 2022 года в Росссийской Федерации насчитывалось 129.8 миллионов интернет-пользователей, примерно 89% популяции [1]. При этом количество доменов в области российского интернета к декабрю 2022 года составляет 4.93 миллиона, при этом на большинстве этих сайтов (73.8%) зарегистрированы физические лица и индивидуальные предприниматели [2]. При этом объём трафика в рунете к 2022 году составил 91 эксабайт (квинтиллион байтов), а к концу 2023 года превысит 100 экабайт [3].

В большинстве современных веб-приложений используются такие технологии, как HTML, JS и CSS, часто используются различные форматы файлов для отображения информации определённого типа (например, PNG, JPG или JPEG для изображений).

Целью работы является разработка сервера раздачи статческой информации на языке Си без использования сторонних библиотек, построенного при помощи паттерна *thread pool* и использующего системный вызов *pselect*. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области;
2) определить функционал, реализуемый сервером раздачи статической инфоормации;
3) провести анализ паттерна *thread pool* и системного вызова *pselect* ;
4) спроектировать и разработать сервер раздачи статическй информации;
5) провести сравнение результатов нагрузочного тестирования при помощи Apache Benchmarks с NGINX.

1 Аналитическая часть

В данном разделе проводится анализ предметной области, анализ паттерна *thread pool*, асинхронной блокирующей модели ввода-вывда и системного вызова *pselect*, а также протокол HTTP и формализация требований к разрабатываемой программе.

1.1 Статическая информация

Статическая информация — информация, которая редко меняется с течением времени. В данной работе под статической информацией будем подразумевать файлы определённых форматоов, которые, предположительно, редко меняются с течением времени.

Согласно поставленной задаче, разрабатываемый сервер разадчи статической инфоормации должен предоставлять доступ к файлам следующих форматов:

1) файлы HTML (имеющие расширение .html), содержащие гипертекстовые документы;
2) скрипты JS (имеющие расширение .js);
3) файлы стилей CSS (имеющие расширение .css) ;
4) файлы изображений форматов PNG и JPEG (имеющие расширения .png и .jpeg или .jpg соответственно);
5) файлы SWF для векторной анимации (имеющие расширение .swf);
6) файлы GIF, содержащие графические изображения или анимации (имеющие расширение .gif).

Также необходимо учесть, что рассмотренные файлы могут иметь размер до нескольких сотен мегабайт, и передача таких файлов должна осуществляться корректно.

1.2 Существующие решения

Поскольку проблема раздачи статической информации возникла вместе с появлением интернета, к 2023 году существует набор готвых решений, как уни-

версальных, так и направленных на соответствие определённым технологиям.

NGINX (от англ. *Engine X*) — веб-сервер и прокси-сервер, работающий как на Unix-подобных системах, так и на системах Miscrosoft Windows (с версии 0.7.72). Разработан в 2004 году российским программистом Игорем Сысоевым, выпускником МГТУ им. Н. Э. Баумана. NGINX позиционируется производителем как простой, быстрый и надёжный сервер, использование корого целесообразно в том числе для раздачи статической информации. Помимо раздачи статической информации, NGINX предоставляет возможности кэширования запросов, сжатия при помощи технологии gzip, балансировки нагрузки между серверами, а также многие другие функции [4]. По данным Netcrafts, к ноябрю 2023 года NGINX используется для поддержки 249 миллионов сайтов, что составляет большую часть (22.83%) опрошенных сайтов [5].

Apache (от англ. *a patchy server*) — ближайший конкурент NGINX в области веб-серверов, является веб-сервером с открытым исходным кодом, поддержкивает такие операционные системы, как Linux, BSD, macOS и Microsoft Windows. С апреля 1996 года и по июль 2016 года являлся самым популярным веб-сервером в мире, поскольку с момента появления интернета являлся самым надёжным сервером, сейчас же используется 22.74% сайтов, лишь немнго уступая NGINX [5].

Cloudflare — американская компания, предоставляющая услуги по раздаче статической информации, защите от атак, предоставлению серверов DNS и проксированию сайтов. Услугами данной компании пользуются 10.62% опрошенных (115 миллионов сайтов), при этом раздача статической информации не стоит у компании на первом месте, отдаётся приоритет защите от DDoS атак: так, в 2014 году Cloudflare выдержала атаку мощностью 400 Гбит / с.

1.3 Паттерн пула потоков

Пул потоков (англ. *thread pool*) — архитектурный паттерн, предназначенный для многопоточной обработи информации [6]. В то время как создавать новый поток под новый запрос является неэффективно, предлагается создать

набор потоков (пул) определённй величины, и поддерживать его на протяжении работы программы. Когда на сервер приходит запрос, предлагается использовать свободный поток из пула потоков. В данном потоке будет произведена обработка пришедшего запроса, после чего поток будет возвращён в пул потоков, готовый к обработке дальнейших запросов.

Исследования [6] показали, что использование пула потоков может значительно улучшить производительность системы и уменьшить время обработки запросов. Тем не менее, существуют следующие проблемы, возникающие при использовании пула потоков:

— определение числа потоков (размера пула);
— контроль целостности очереди и предотвращения обработки одного запроса двумя потоками;
— оптимальное использование вычислительных ресурсов и предотвращение ”простаивания”потоков.

В данной работе предлагается оставить возможность указывать количество потоков при запуске сервера. Это позволит использовать разработанную программу наиболее гибко.

Последние две проблемы предлагается решить при помощи средств взаимоисключения (например, mutex-ов), а также использования каждым потом своего сокета, обеспечив неблокирующий ввод-вывод посредством сокетов. На рисунке 1 представлена концептуальная модель пула потоков: очередь запросов, ожидающих обработку, пул потоков и обработанные запросы.

1.4 Асинхронный блокирующий ввод-вывод

Прежде, чем перейти к описанию системного вызова *pselect*, рассмотрим модель асинхронного блокирующего ввоода-вывода, в котой этот вызов используется.

Модель асинхронного блокирующего ввода-вывода основана на опросе набора источников ввода-вывода на предмет готовности. Блокировка происходит на моменте опроса: главный процесс блокируется в ожидании готовности

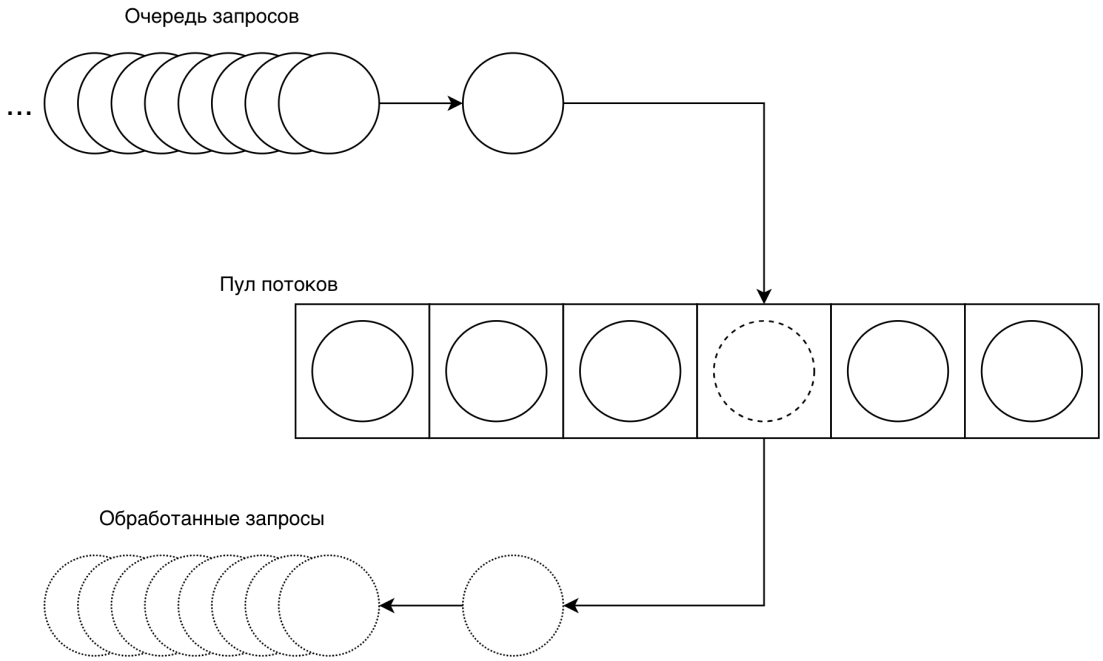

Рисунок 1 – Концептуальная модель пула потоков

соединения. Как только соединение установлено, можно осуществлять передачу и дальнейшую обработку данных. Асинхронным данный метод называется потому, что возможен одновременный опрос нескольких источников ввода-вывода и блокировка лишь до того момента, когда первый из источников сообщит о готовности.

На рисунке 2 представлена модель асинхронного блокирующего ввода-вывода с ипользованием select: запрос на чтение, блокировка в ожидании результата чтения, передача данных из пространства ядра в пространство пользователя.

1.5 Системный вызов pselect

Системный вызов *pselect* — функция, существующая в Unix-подобных и POSIX системах и предназначенная для опроса файловых дескрипторов открытых каналов ввода-вывода. Подключение данной функции происходит при помощи заголовочного файла sys/select.h на языке программирования Си.

Функция pselect принимает на вход 6 аргументов:

1) nfds, число типа int, значение котого вычисляется как $n+1$, где n — макси-

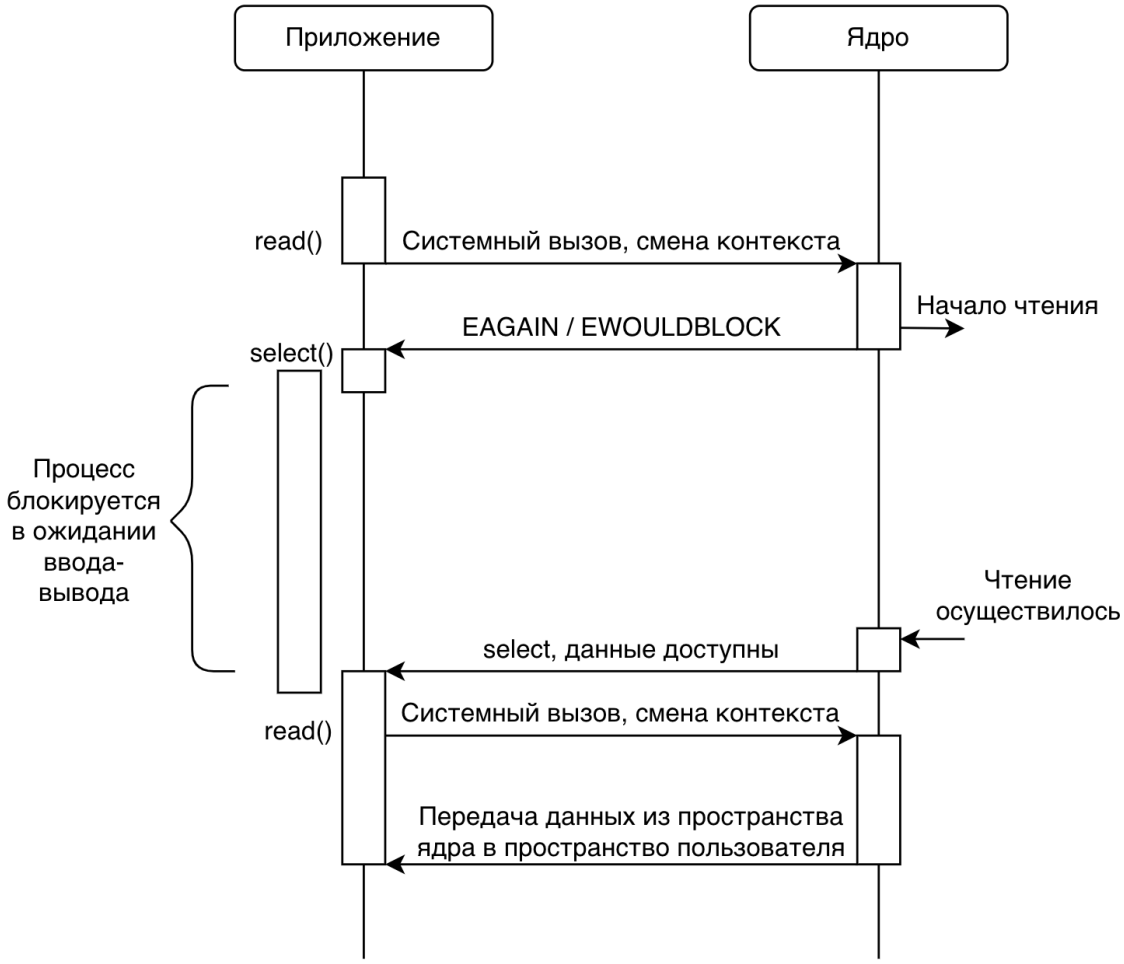

Рисунок 2 – Модель асинхронного блокирующего ввода-вывода с ипользованием select

мальное значение файлового дескриптора из наборов файловых дескрипторов, опрос которых осуществляется;

2) readfds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет чтения из них;
3) writefds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет записи в них;
4) exceptfds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет появления исключительных ситуаций;
5) timeout, структура типа struct timespec (содержит миллисекунды и наносекунды), определяет максимальный интервал времени, в течение которого будет осуществлено ожидание;

6) sigmask, маска типа sigset_t сигналов, позволяющая изменить маску сигналов на время работы функции select.

Возвращает функция количество файловых дескрипторов, готовых к вводу-выводу, 0 в случае, если за интервал времени timeout таких дескрипторов нет и -1 в случае ошибки.

При использовании в разрабатываемой программе необходимыми будут лишь параметры nfds, readfds и timeout, поскольку при раздаче статической информации требуется чтение с диска в определённый промежуток времени, а не запись на диск или мониторинг исключительных ситуаций на сокетах.

1.6 HTTP

HTTP (англ. *Hypertext Transfer Protocol*) — протокол прикладного уровня OSI ISO, изначально предназначенный для передачи гипертекстовых документов и сейчас являющийся универсальным средством взаимодействия между узлами в интернете.

HTTP сообщение состоит из трёх частей: стартовой строки, определяющей тип соообщения, заголовков, характеризующих тело сообщения, параметры передачи и прочие сведения, а также тела сообщения.

Одним из обязательных заголовков в HTTP сообщении является метод запроса: OPTIONS, GET, HEAD, PUT, POST, PATCH, DELETE, TRACE или CONNECT. В данной работе будут рассмотрены лишь два метода: GET и HEAD, поскольку именно они имеют смысл при реализации сервера раздачи статической инфрмации.

Метод GET используется для запроса содержимого указанного ресурса. Запросы GET считаются идемпотентными, т. е. результат запроса всегда будет одним и тем же в случае статической информации. В тело запроса при этом отсутствует, а в теле ответа будет находиться содержимое запрошенного ресурса, в случае разрабатываемого сервера — содержимое запрошенного файла.

Метод HEAD аналогичен методу GET за исключением того, что в ответе HEAD отсутствует тело сообщения. Обычно HEAD-запросы применяются для

извлечения метаданных или проверки существования ресурса.

Ещё один заголовок, который необходимо рассмотреть для правильной передачи данных, это Content-Type. Он предназначен для описания типа передаваемых данных для правильной интерпретации на стороне получателя. Для каждого из рассмотренных выше типов значение заколовка будет своим:

1) файлы HTML — text/html
2) скрипты JS — text/javascript;
3) файлы стилей CSS — text/css;
4) файлы изображений форматов PNG и JPEG — image/png и image/jpeg;
5) файлы SWF для векторной анимации — application/x-shockwave-flash;
6) файлы GIF — image/gif.

1.7 Формализация требований к разрабатываемой программе

На основе рассмотренной выше информации и задания к работе разрабатываемая программа должна соответстовать следующим требваниям:

1) поддержка запросов GET и HEAD (поддержка статусов 200, 403, 404);
2) ответ на неподдерживаемые запросы статусом 405;
3) выставление content type в зависимости от типа файла (поддержка .html, .css, .js, .png, .jpg, .jpeg, .swf, .gif);
4) корректная передача файлов размером в 100мб;
5) сервер по умолчанию должен возвращать html-страницу на выбранную тему с css-стилем;
6) учесть минимальные требования к безопасности статик-серверов (предусмотреть ошибку в случае если адрес будет выходить за root директорию сервера);
7) реализовать логгер;
8) использовать язык Си без сторонних библиотек;
9) реализовать архитектуру при помощи пула потоков с использованием системного вызова pselect;
10) обеспечить стабильную работу сервера.

Вывод

В данном разделе проводится анализ предметной области, анализ паттерна *thread pool*, асинхронной блокирующей модели ввода-вывда и системного вызова *pselect*, а также протокол HTTP и формализация требований к разрабатываемой программе. Необходимо будет провести нагрузочное тестирование при помощи Apache Benchmarks и сравнить результаты тестирования разработанной программы с результатами тестирования NGINX-сервера. Как архитектурный паттерн будет использоваться пул потоков, как модель ввода-вывода асинхронный блокирующий ввод-вывод, осуществляемый при помощи системного вызова *pselect*.

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.

Для таблиц необходимо описать информацию о стоблцах (типы данных, ограничения, значения), а также описать связь между описанными таблицами при помощи диаграммы базы данных.

На основе используемых паттернов проектирования и информации о типе базы данных необходимо построить диаграмму классов приложения, на которой будут выделены основные классы и интерфейсы, а также их методы.

2.1 Таблицы БД

Для выделенных в аналитической части категорий данных и соответствующей им информации, приведённой в таблице **??**, 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 **??** диаграмму сущность-связь, определим для каждой из перечисленных

выше таблиц столбцы, их типы и ограничения, в таблицах 1 – 9.

Таблица 1 – 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 2 – 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 3 – 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 4 – 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 5 – 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 6 – 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 7 – 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 8 – 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 9 – 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 10 – 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 11 – 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

Создание пользовательских типов данных, а также таблиц и ограничений для них будет приведено в технологической части.

2.1.1 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 3.

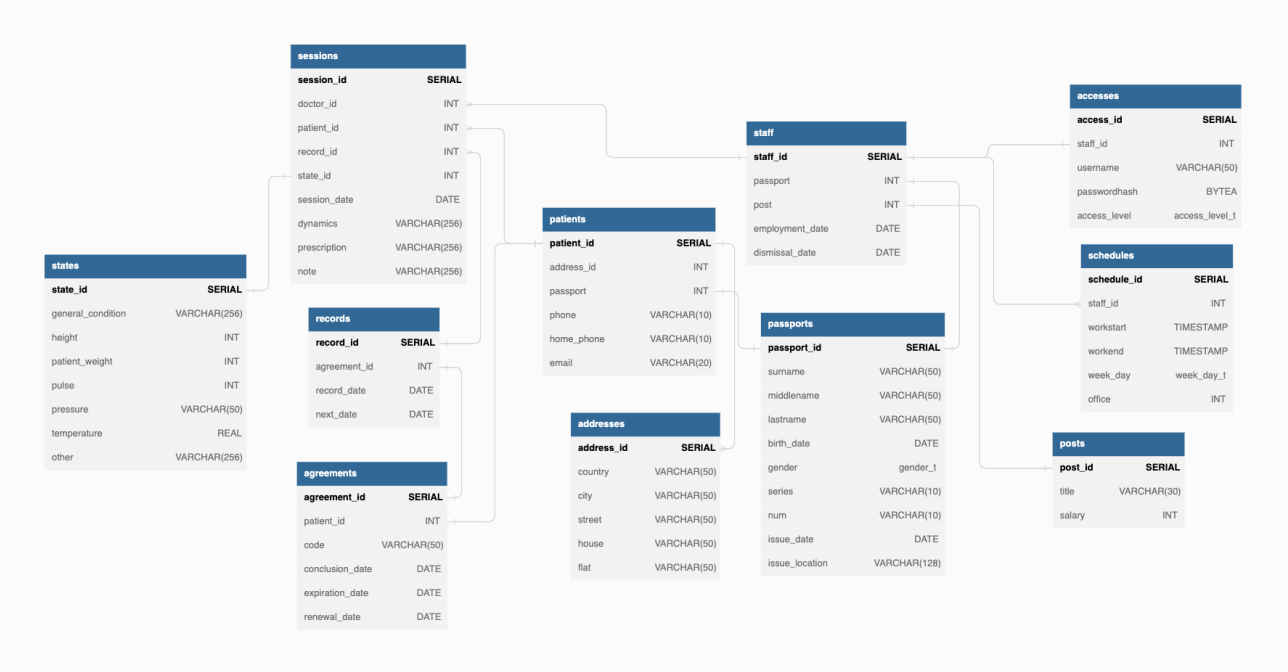

Рисунок 3 – Диаграмма базы данных для поставленной задачи

2.2 REST

REST (*англ.* Representational State Transfer, передача репрезентативного состояния) [8] — 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 4.

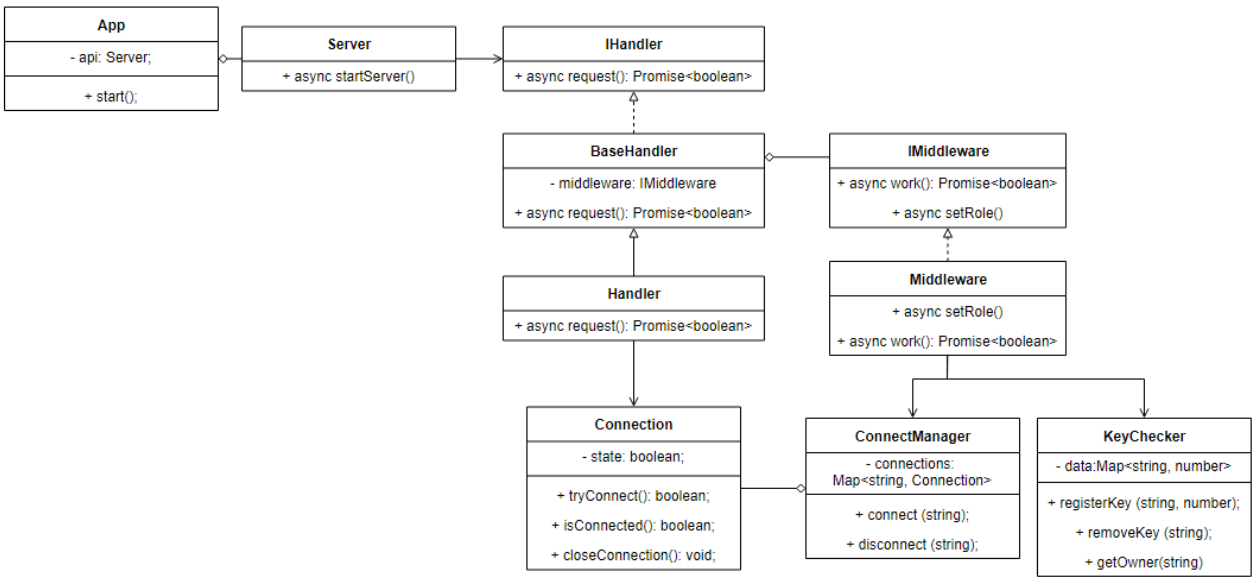

Рисунок 4 – Диаграмма классов

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 разобран основной используемый паттерн проектирования REST.

3 Технологическая часть

В данном разделе рассмотрены средства разработки программного обеспечения, приведены детали реализации и листинги исходных кодов.

Приведённые листинги исходных кодов включают в себя создание таблиц и ограничений, создание ролевой модели и функции управления доступами (на основе приведённой выше Use-Case диаграммы), реализацию сервера (менеджер соединений, класс соединений).

3.1 Средства реализации

Как основное средство реализации и разработки ПО был выбран язык программирования TypeScript [9]. Причиной выбора данного языка является тот факт, что он имеет компилируется в JavaScript [10], 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 [11].

Для запуска написанного сервера была использована библиотека Node.js [12] 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 [13], которая была выбрана из-за своего превосходства в скорости обработки соединений по сравнению с другими библиотеками (Koa, Express, Restify, Hapi) [14]. Для доступа к СУБД, реализованной при помощи PostgreSQL [15], была использована библиотека pg-promise [16], предоставляющая интерфейс для взаимодействия с базой данных.

Для реализации графического интерфейса API была использована библиотека FastAPI [17], позволяющая быстро создать рабочий интерфейс для созданного API (вместе с документацией). Средой разработки послужил графиче-

ский редактор Visual Studio Code [18], который известен содержанием большого количество плагинов, ускоряющих процесс разработки программы.

3.1.1 Создание таблиц и ограничений

Результат переноса приведённых выше таблиц в скрипты создания пользовательских типов данных и страниц, написанных на SQL, можно увидеть на листингах 1 – 6.

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');
```

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 agr_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;
```

3.1.2 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
$$
DECLARE
    role_name regrole;
BEGIN
    role_name = (SELECT role FROM users WHERE
        accesses.access_id = user_uuid)::regrole;
    execute format('SET role %I', role_name);
    execute format('SET app.user_uuid = %L', user_uuid);
END;
$$;
```

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 скриптов создания таблиц и ограничений, а также листинги исходного кода программы, разработанной на основе алгоритмов, выбранных в аналитическом разделе и соответствующих приведённой в конструкторской части диаграмме классов приложения.

4 Исследовательская часть

В данном разделе будут приведены примеры работы разработанной программы и поставлен эксперимент по исследованию зависимости времени обработки запроса в зависимости от количества полей таблиц. Результаты иссследования позволят точнее определять основные требования к базе данных при её выборе для проектирования системы, тем самым облегчая процесс выбора наиболее подходящей для конкретной задачи СУБД.

4.1 Описание интерфейса

В основной части страницы находятся методы API, а также их особенности (тип метода, адрес методы, структура, которую метод использует). Часть таких методов приведена на рисунке 5).

Рисунок 5 – Интерфейс программы: методы API

По нажатию на элемент каждого метода появляется подробное описание: какие данные метод принимает на вход, какие параметры являются обязательными, а какие опциональными, структуры какого вида возвращает и в каком виде, что возвращает в случае ошибки (рисунок 6).

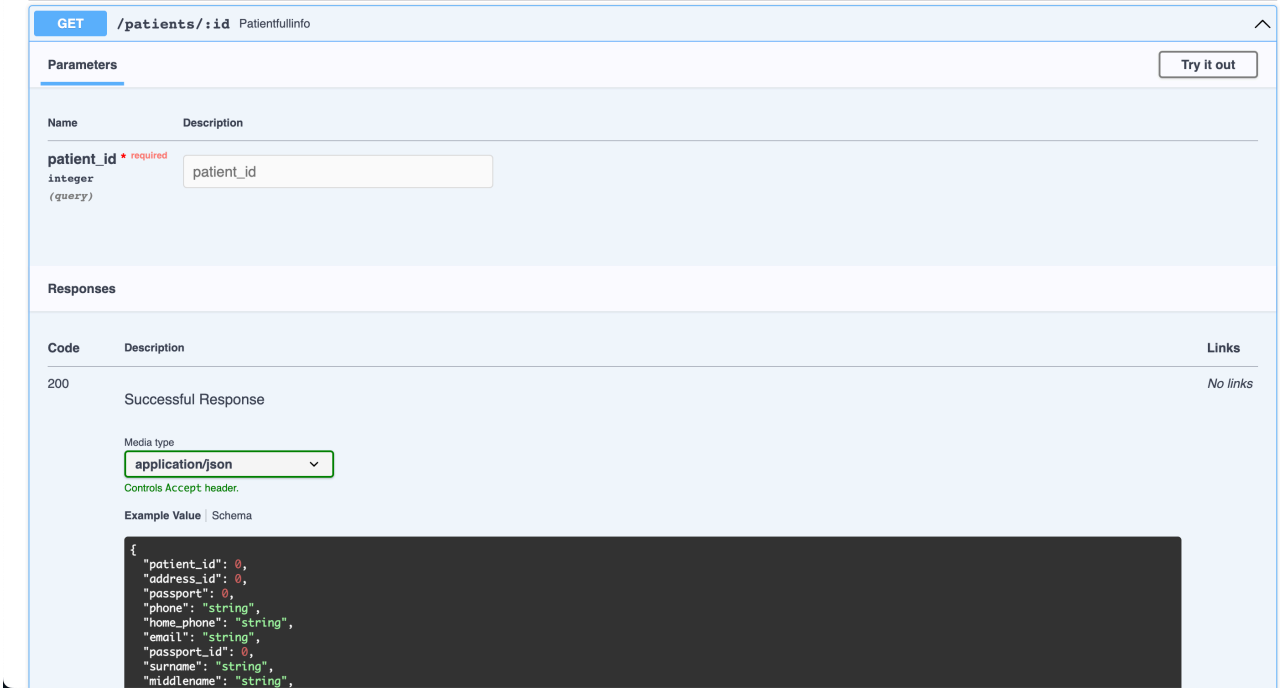

Рисунок 6 – Интерфейс программы: подробное описание метода API

4.2 Демонстрация работы программы

На рисунках 7 – 8 продемонстрирована работа программы на примере осуществления запроса для получения списка пациентов. Созданный интерфейс позволяет не толькоо просматривать методы API, как было показано выше, но и осуществлять отправку конкретных запросов. При этом имеется возможность посмотреть результат запроса (его статус и заголовки, присланные данные), его адрес, а также получить команды для отправления эквивалентного запроса из командной строки.

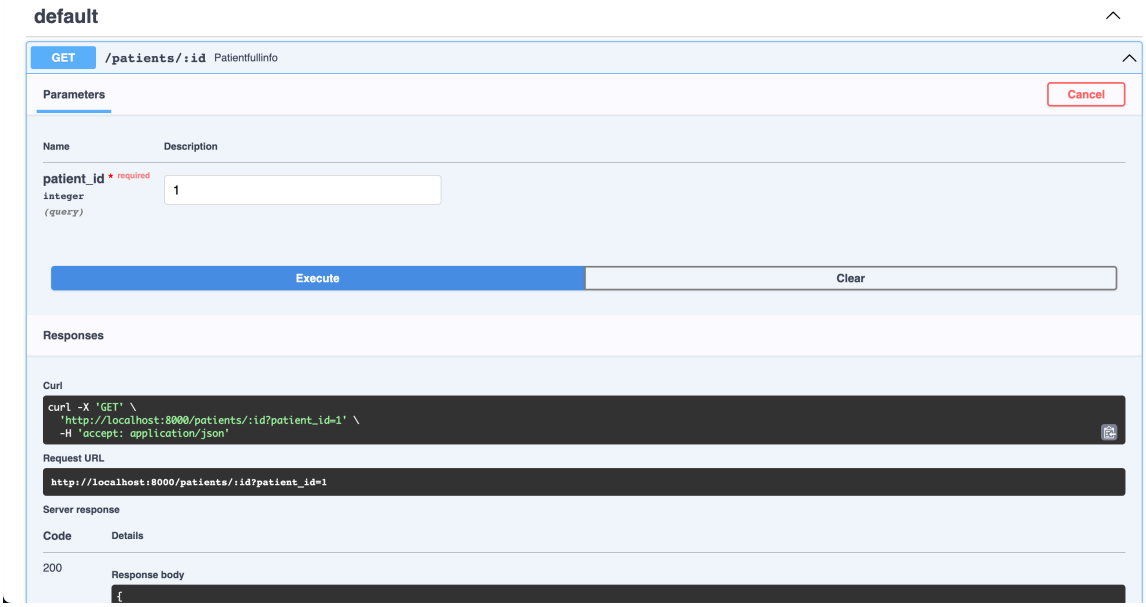

Рисунок 7 – Демонстрация работы программы (часть 1)

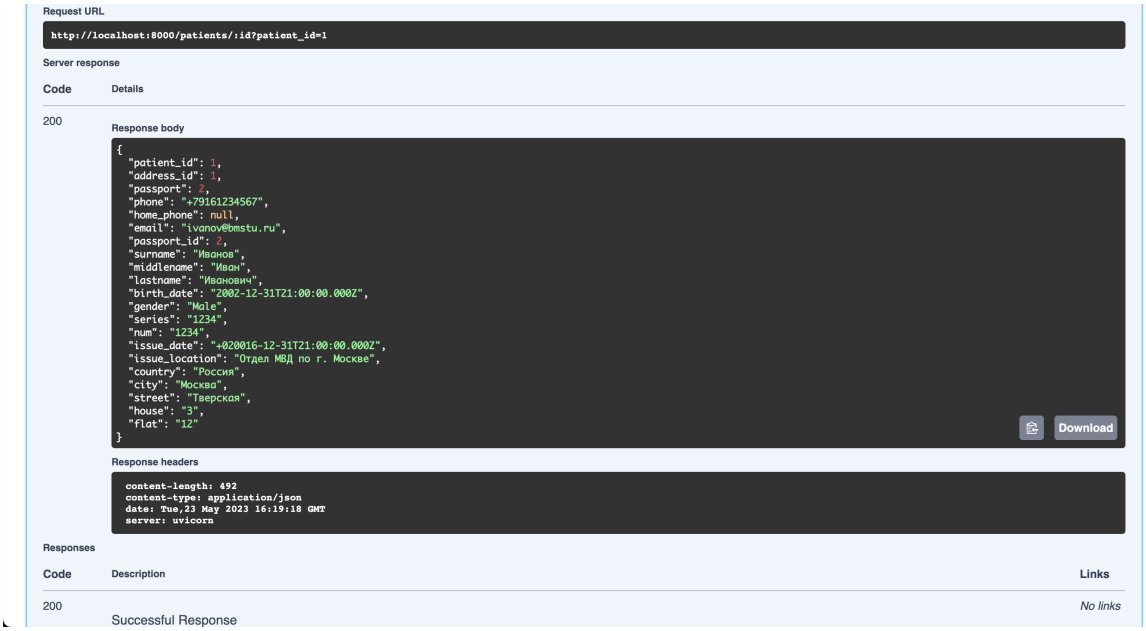

Рисунок 8 – Демонстрация работы программы (часть 2)

4.3 Постановка эксперимента

Целью эксперимента является сравнение времени обработки запроса (GET, на примере получения списка пациентов) в зависимости от количества полей таблицы.

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система: macOS Ventura 13.2.1 (22D68) [19];
— Память: 16 Гб с тактовой частотой 2133 МГц LPDDR3 [20];
— Процессор: Intel Core™ i7-8559U [21] с тактовой частотой 2.70 ГГц;
— Видеокарта: Intel Iris Plus Graphics 655 [22] с объёмом памяти 1536 Мб.

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

4.4 Сравнение времени обработки

Результаты измерений времени обработки запросов в зависимости от количества полей таблицы приведены в таблице 12 и на рисунке 9.

Таблица 12 – Зависимость времени обработки запроса от количества полей в таблице

Количество полей	Время обработки, мс
10	1.10
100	2.07
1000	8.50
10000	80.43
100000	946.73
1000000	8780.69

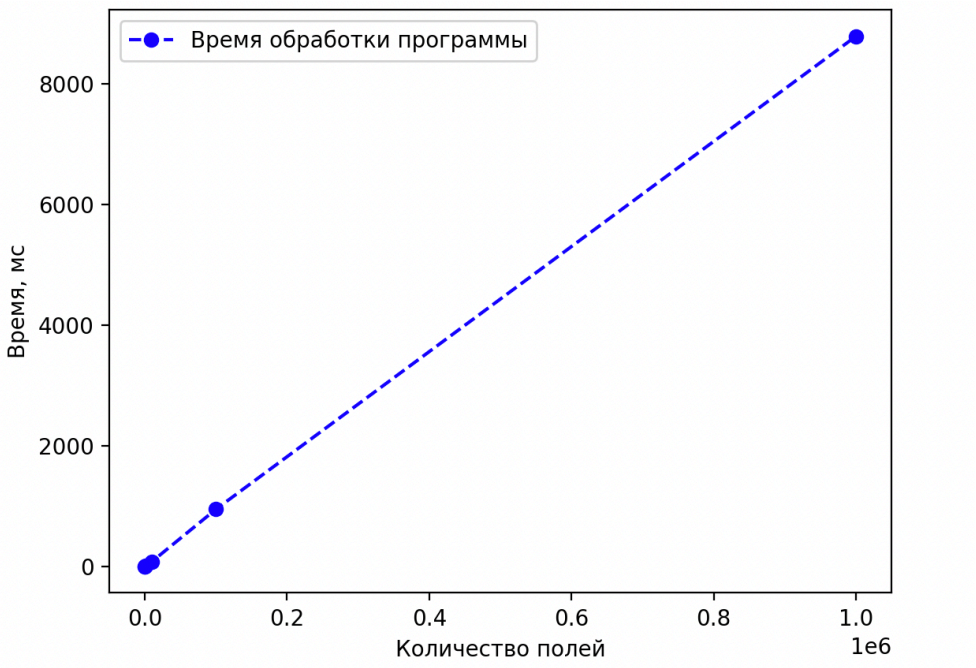

Рисунок 9 – Результаты замеров времени обработки запросов

Результаты тестирования приводят к выводу, что зависимость времени обработки запроса от количества полей таблицы имеет линейный характер.

Вывод

В данном разделе были описаны элементы интерфейса и приведены примеры работы разработанной программы. Графический интерфейс предоставляет возможность просмотреть список методов реализовнного API и их подробные характеристики (тип метода, адрес, используемые структуры), а также отправить конкретный запрос, осуществив ввод необходимых аргументов. Проведён эксперимент по исследованию зависимости времени обработки запроса от количества полей таблицы. Исходя из результатов тестирования, характер зависимости является линейным.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное обеспечение, предназначенное для использования как информационная система для частных поликлиник. Был проведён анализ предметной области, проведён анализ существующих СУБД, разработаны диаграммы сценариев использования, сущностей и связей, а также базы данных. Была спроектирована база данных в соответствии с выбранной СУБД, разработана ролевая модель для управлению доступа к данным, также приведена диаграмма классов приложения в соответствии с выбранной REST архитектурой. Разработанная программа позволяет просматривать методы созданного API, их подробные характеристики, а также осуществлять запросы после ввода необходимых аргументов. Проведено исследование зависимости времени обработки запроса от количества полей в таблице. Из результатов следует, что зависимость имеет линейный характер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DIGITAL 2022: THE RUSSIAN FEDERATION [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2022-russian-federation (дата обращения: 07.12.2023).

2. Forbes. Число доменов зоны .ru [Электронный ресурс]. — Режим доступа: https://www.forbes.ru/tekhnologii/483876-cislo-domenov-zony-ru-po-itogam-2022-goda-opustilos (дата обращения: 07.12.2023).

3. Telecom. В 2023 году проводной трафик превысит 100 эксабайт [Электронный ресурс]. — Режим доступа: https://telecomdaily.ru/news/2023/06/15/v-2023-godu-provodnoy-trafik-prevysit-100-eksabayt (дата обращения: 07.12.2023).

4. NGINX, официальный сайт [Электронный ресурс]. — Режим доступа: https://www.nginx.com/ (дата обращения: 08.12.2023).

5. November 2023 Web Server Survey [Электронный ресурс]. — Режим доступа: https://www.netcraft.com/blog/november-2023-web-server-survey/ (дата обращения: 08.12.2023).

6. Ling Y., Mullen T,, Lin X., Analysis of optimal thread pool size — ACM SIGOPS Operating Systems Review, 200. – 55 с..

7. select(2) — Linux manual page [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man2/select.2.html (дата обращения: 08.12.2023).

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

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

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

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

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

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

14. Fastify Benchmarks — official site [Электронный ресурс]. — Режим доступа: https://vk.cc/coz0VO (дата обращения: 17.05.2023).

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

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

17. FastAPI — official site [Электронный ресурс]. — Режим доступа: https://fastapi.tiangolo.com/lo/ (дата обращения: 17.05.2023).

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

19. macOS Ventura — official site [Электронный ресурс]. — Режим доступа: https://www.apple.com/macos/ventura/ (дата обращения: 17.05.2023).

20. LPDDR3 SDRAM Documentation [Электронный ресурс]. — Режим доступа: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf (дата обращения: 17.05.2023).

21. Процессор Intel Core™ i7-8559U [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/137979/intel-core-i78559u-processor-8m-cache-up-to-4-50-ghz.html (дата обращения: 17.05.2023),

22. Intel Iris Plus Graphics 655 [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/graphics/140931/intel-iris-plus-graphics-655.html (дата обращения: 17.05.2023),