СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Название МО ЭВМ

Индивидуальное домашнее задание
по дисциплине «Введение в нереляционные базы данных»
Тема: База данных заболеваний, анализов, симптомов и синдромов (с возможностью логического вывода)

Студент гр. 1384 _____ Прошичев А.В.

Студент гр. 1384 _____ Шушков Е.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студенты

Прошичев А.В.

Шушков Е.В.

Группа 1384

Тема работы (проекта): База данных заболеваний, анализов, симптомов и синдромов (с возможностью логического вывода)

Исходные данные:

Медицинский датасет[1], содержащий диагнозы, симптоматику, рекомендации к лечению и связь между ними. Требуется реализовать приложение для быстрого медицинского диагностирования на основе СУБД Neo4j[2], позволяющее хранить информацию о пациентах, их обращениях, заболеваниях, симптомах, рекомендуемых анализах для подтверждения диагноза, а также оценивать вероятность по наличия диагноза по набору симптомов.

Содержание пояснительной записки:

“Содержание”

“Введение”

“Качественные требования к решению”

“Сценарий использования”

“Модель данных”

“Разработка приложения”

“Вывод”

“Приложение”

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 24.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студент гр. 1384 Прошичев А.В.

Студент гр. 1384 Шушков Е.В.

Преподаватель Заславский М.М.

АННОТАЦИЯ

В рамках данного курса разработано приложение[3] быстрого медицинского диагностирования на основе СУБД Neo4j[2]. Приложение способно на основе набора симптомов информировать пользователя о возможных заболеваниях. В ходе диагностики пользователь может ознакомиться с описанием диагнозов, с рекомендациями по их профилактике и анализами, которые могли бы помочь утвердить поставленный диагноз. Приложение позволяет хранить информацию не только о заболеваниях, диагнозах и анализах, но и о пользователях (далее - пациентах) и их обращениях. Для пользователей с правами администратора доступны разделы просмотра записей в БД и статистического анализа по множеству сделанных обращений.

SUMMARY

As part of this course, an application for rapid medical diagnostics based on the Neo4j database has been developed. The application can inform the user about possible diseases based on a set of symptoms. During the diagnosis, the user can familiarize themselves with the descriptions of the diagnoses, recommendations for their prevention, and analyses that could help confirm the diagnosis. The application allows storing information not only about diseases, diagnoses, and analyses but also about users (hereafter referred to as patients) and their inquiries. For users with administrator rights, sections for viewing database records and statistical analysis of multiple inquiries are available.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы - создать приложение быстрого медицинского диагностирования на основе открытого датасета[1] с использованием СУБД Neo4j[2]. Приложение должно позволять пользователям на основе набора симптомов получать информацию о возможных заболеваниях, рекомендациях по их профилактике и необходимых анализах для подтверждения диагноза. Приложение должно обеспечивать хранение и управление данными о пациентах, их обращениях, заболеваниях, симптомах и анализах, а также предоставлять возможность логического вывода для оценки вероятности диагноза.

Методы решения

Для достижения поставленной цели будут использованы следующие методы:

1. Использование СУБД Neo4j: Для хранения и управления данными будет использована графовая база данных Neo4j, которая позволяет эффективно моделировать сложные связи между различными сущностями, такими как заболевания, симптомы, анализы и пациенты.
2. Разработка модели данных: Создание модели данных, которая включает в себя сущности "Пациент", "Обращение", "Заболевание", "Симптом", "Анализ" и их взаимосвязи. Модель должна учитывать возможность логического вывода для оценки вероятности диагноза.
3. Реализация приложения: Разработка пользовательского интерфейса и серверной части приложения, которое будет взаимодействовать с базой данных Neo4j. Приложение должно предоставлять функционал для ввода симптомов, отображения возможных диагнозов, рекомендаций по профилактике и необходимых анализов. Для реализации серверной части выбрана библиотека Flask[4] языка программирования Python[5]. Для реализации пользовательского интерфейса - Angular[6] языка программирования TypeScript[7].

4. Логический вывод: Использование алгоритмов логического вывода для оценки вероятности диагноза на основе введенных симптомов. Это позволит пользователям получать более точные и обоснованные рекомендации.

5. Административные функции: Реализация функционала для пользователей с правами администратора, включая просмотр записей в базе данных и статистический анализ обращений пациентов.

Качественное требование к решению:

- Удобный пользовательский интерфейс
- Хранение всех обращений пациентов для реализации повторных запросов и статистического анализа
- Быстрая и точная оценка вероятности диагнозов

1. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Ниже представлен макет (см. Рисунок 1), на основе которого разрабатывался пользовательский интерфейс

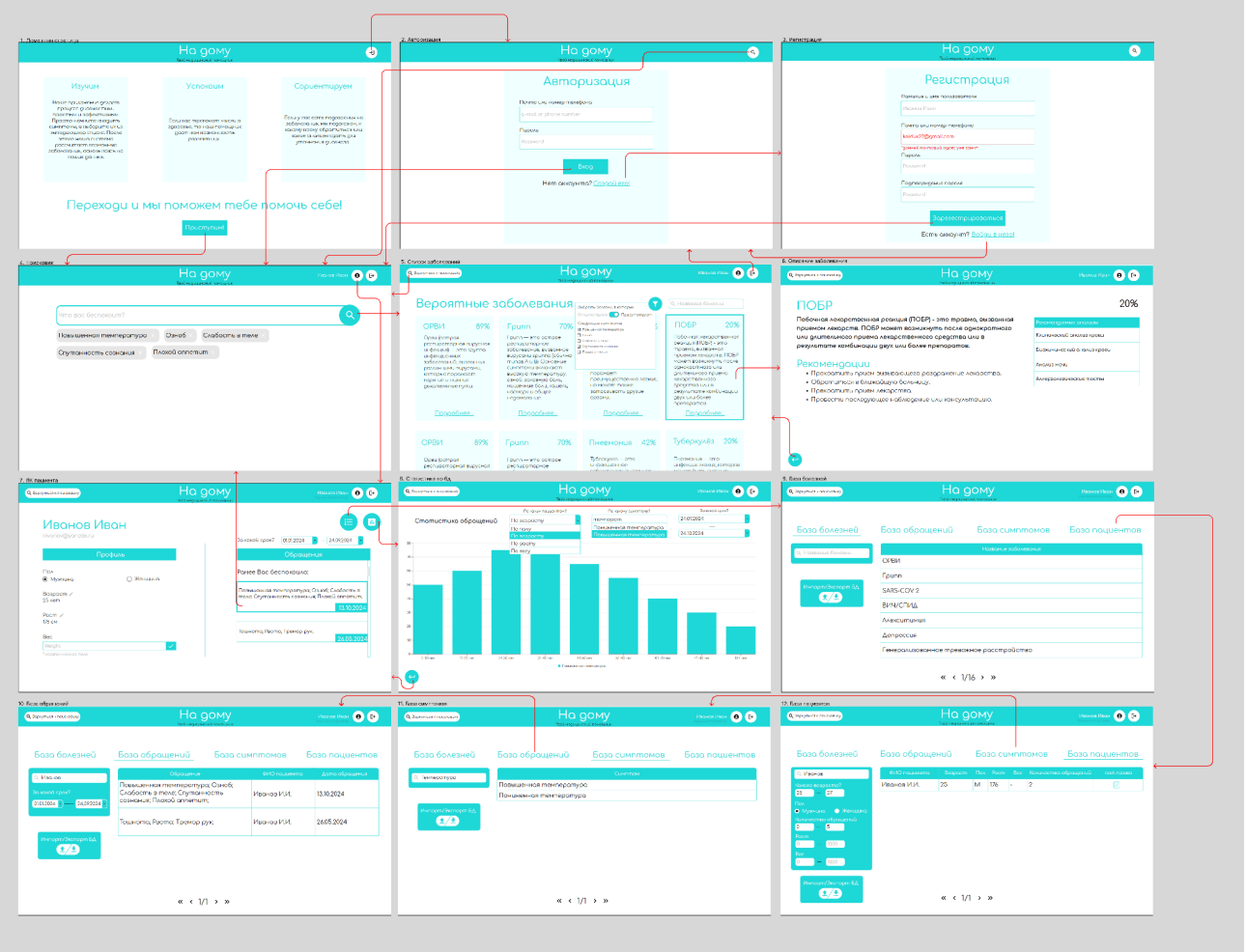


Рисунок 1 - (UI-макет)

1.2. Описание сценариев использования

Для корректного взаимодействия пользователя с интерфейсом были рассмотрены следующие сценарии использования, на основе которых будет реализованно приложение.

Регистрация пользователя

1. Пользователь находит и нажимает кнопку для входа/регистрации.

2. Переходит на страницу входа и видит гиперссылку для создания аккаунта.

3. Нажимает гиперссылку и переходит на страницу регистрации.

4. Вводит данные и подтверждает регистрацию.

5. Система переводит пользователя на поисковик.

Некорректные данные при регистрации

1. Система проверяет введённые данные.

2. Проверяет корректность почты/номера телефона.

3. Проверяет уникальность почты.

4. Проверяет пароль на соответствие критериям.

5. Проверяет подтверждение пароля.

6. Выводит сообщение с предупреждением при нарушении.

Вход с некорректными данными или без аккаунта

1. Система проверяет введённые данные.

2. Проверяет существование аккаунта.

3. Проверяет правильность пароля.

4. Выводит сообщение с предупреждением при нарушении.

Проблемы с базой данных или сервером

1. Система получает ошибку от базы данных или сервера.

2. Переводит пользователя на страницу с сообщением о технических работах.

3. Если пользователь администратор, показывает ошибку.

Ввод данных для определения болезни

1. Пользователь открывает поисковик и вводит симптомы.

2. Выбирает симптомы и нажимает Enter или значок поиска.

3. Получает результаты и завершает работу.

Удаление неправильного симптома

1. Пользователь нажимает на “крестик” около симптома.
2. Не нужный симптом удаляется.

Поиск без ввода симптомов

1. Система не запускает поиск без симптомов.
2. Выдаёт предупреждение о необходимости ввода хотя бы одного симптома.

Изучение списка заболеваний

1. Пользователь вводит симптомы и фильтрует результаты.
2. Изучает список заболеваний по введённым симптомам.

Необходимые анализы для выявления болезни

1. Пользователь вводит симптомы и выбирает болезнь.
2. Нажимает “Подробнее…” и получает перечень необходимых анализов.

Восстановление предыдущего запроса

1. Пользователь заходит в профиль и фильтрует запросы по времени.
2. Находит нужный запрос и переходит на страницу поиска с введёнными симптомами.

Изменение данных о себе

1. Пользователь заходит в профиль и нажимает иконку “Карандаш”.
2. Изменяет данные и сохраняет их.

Ввод некорректных данных

1. Система проверяет тип и корректность введённых данных.
2. Выводит сообщение с предупреждением при нарушении.

Статистика по симптомам

1. Пользователь переходит в профиль и нажимает кнопку статистики.
2. Выбирает характеристики и симптомы.
3. Получает графическую визуализацию данных.

Изучение базы данных (Администратор)

1. Администратор переходит в профиль и выбирает иконку “Базы данных”.
2. Выбирает раздел и фильтрует данные.
3. Получает отфильтрованную информацию.

Выдача административных прав

1. Администратор ищет пользователя в “Базе пациентов”.
2. Ставит или снимает подтверждение в колонке “Права администратора”.

Импорт/экспорт данных

1. Администратор выбирает иконку “Базы данных” и нажимает кнопку импорта/экспорта.
2. Выбирает папку для хранения файла и получает уведомление об успешной загрузке.

Загрузка файла с некорректными данными

1. Система проверяет расширение файла.
2. Выдаёт уведомление об ошибке, если файл не json или возникла ошибка при парсинге.

Эти сценарии охватывают основные действия пользователей и администраторов в системе.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных

Нереляционную модель данных можно описать следующей схемой (см. рисунок 2).

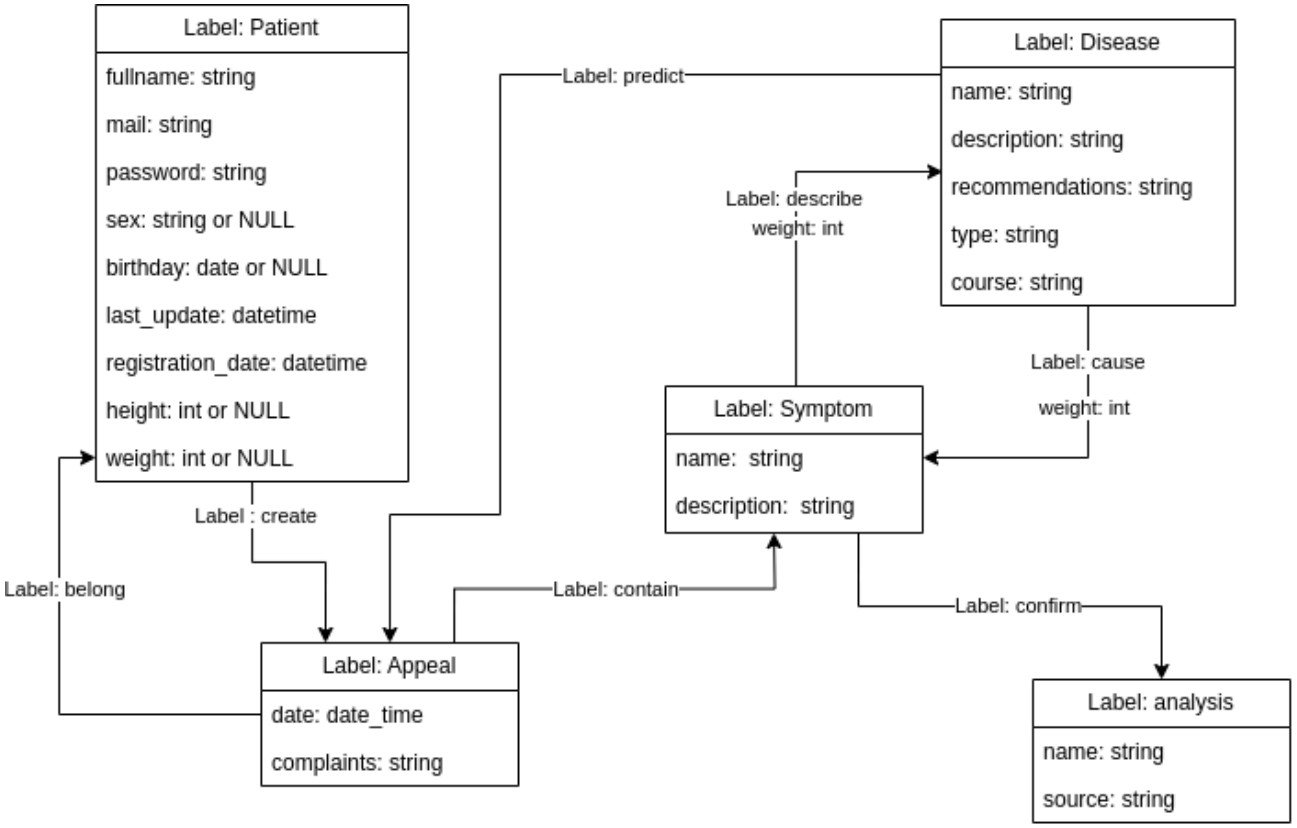


Рисунок 2 (Схема нереляционной модели)

Описание назначений коллекций, типов данных и сущностей:

Узлы

Patient - узел с одноимённым классификационным признаком *Patient*, описывающий аккаунт зарегистрированного пользователя. Имеет следующие атрибуты key-value:

- fullname: string – ФИО пользователя. Максимальная длина - 100 символов
- mail: string - Адрес электронной почты пользователя. Максимальная длина - 255 символов

- password: string - Пароль от данного аккаунта. Максимальная длина - 64 символа
- sex: string or NULL- Пол пользователя (Женский/Мужской). Максимальная длина - 7 символов
- birthday : date or NULL – День рождения пользователя
- last_update : datetime - Время последнего обновления страницы пользователем
- registration_date : datetime - Время регистрации пользователя на сайте
- height: int or NULL - Рост пользователя
- weight: int or NULL - Вес пользователя

Appeal - узел с одноимённым классификационным признаком *Appeal*, описывающий запрос на составление анамнеза зарегистрированного пользователя. Имеет следующие атрибуты key-value:

- date: datetime – Дата и время отправления запроса
- complaints : string or NULL - жалобы пациента, с которым пользователь пришёл к доктору. Максимальная длина - 500 символов.

Symptom - узел с одноимённым классификационным признаком *Symptom*, хранящий информацию об определённом симптоме. Имеет следующие атрибуты key-value:

- name: string – Наименование симптома. Максимальная длина - 50 символов
- description: string - Описание симптома. Максимальная длина - 500 символов

Disease - узел с одноимённым классификационным признаком *Disease*, хранящий информацию об определённом заболевании. Имеет следующие атрибуты key-value:

- name: string – Наименование заболевания. Максимальная длина - 50 символов
- description: string - Описание заболевания. Максимальная длина - 500 символов
- recommendations: string - Рекомендации по лечению и профилактике. Максимальная длина - 500 символов
- type : string - тип заболевания (вирусное, бактериальное, ...). Максимальная длина - 50 символов.
- course : string - течение заболевания (острое, хроническое, ...). Максимальная длина - 50 символов.

Analysis - узел с одноимённым классификационным признаком *Analysis*, хранящий информацию об определённом анализе. Имеет следующие атрибуты key-value:

- name: string – Наименование анализа. Максимальная длина - 50 символов
- source : string or NULL - Источник анализа (анализ крови на гемоглобин, общий анализ крови и т.д.). Максимальная длина - 50 символов.

Связи

Create - связь с одноимённым классификационным признаком *Create*, описывающая принадлежность узла *Appeal* к узлу *Patient*. Позволяет получить по определённому пользовательскому аккаунту сделанные с него обращения.

Belong - связь с одноимённым классификационным признаком *Belong*, описывающая принадлежность узла *Patient* к узлу *Appeal*. Позволяет получить

по определённому обращению пользовательский аккаунт, с которого оно было сделано.

Contain - связь с одноимённым классификационным признаком *Contain*, описывающая принадлежность узла *Symptom* к узлу *Appeal*. Позволяет определять симптомы, введённые в конкретном обращении.

Confirm - связь с одноимённым классификационным признаком *Confirm*, описывающая принадлежность узла *Analysis* к узлу *Symptom*. Позволяет определять анализы, которые помогли бы выявить конкретный симптом.

Predict - связь с одноимённым классификационным признаком *Predict*, описывающая принадлежность узла *Appeal* к узлу *Disease*. Позволяет получить и обработать симптомы, введённые в данные обращении и симптомы, характеризующие болезнь, для последующей подборки рекомендуемых анализов.

Describe - связь с одноимённым классификационным признаком *Describe*, описывающая принадлежность узла *Disease* к узлу *Symptom*. Содержит атрибут *weight*, указывающий на корреляцию между данными симптомом и болезнью. Позволяет определять заболевания, характеризующиеся данным симптомом.

Cause - связь с одноимённым классификационным признаком *Cause*, описывающим принадлежность узла *Symptom* к узлу *Disease*. Содержит атрибут *weight*, указывающий на корреляцию между данными симптомом и болезнью. Позволяет определять симптомы, набор которых характеризует данное заболевание.

Оценка объема информации, хранимой в модели

Предположим, что

- пользователь будет создавать в среднем 7 обращений
- каждый симптом будет иметь в среднем один анализ для своего подтверждения
- каждая болезнь будет иметь в среднем по 5 симптомов
- в каждом обращении будет в среднем по 5 симптомов

- каждое обращение будет прогнозировать с среднем 10 болезней

Будем считать, что каждый символ у нас занимает 2 байта в связи с использованием русского языка.

Общий объём каждого узла

Patient:

- fullname: 100 символов по 2 байта = 200 байт
- mail: 255 символов по 2 байта = 510 байт
- password: 64 символа по 2 байта = 128 байт
- sex: 14 байт
- birthday : 8 байт
- last_update : 8 байт
- registration_date : 8 байт
- height: 4 байта
- weight: 4 байта

Итого: 851 байт

Appeal:

- date: 8 байт
- complaints: 500 символов * 2 байта = 1000 байт

Итого: 1008 байт

Symptom:

- name: 50 символов по 2 байта = 100 байт
- description: 500 символов по 2 байта = 1000 байт

Итого: 1100 байт

Disease:

- name: 50 символов по 2 байта = 100 байт
- description: 500 символов по 2 байта = 1000 байт
- recommendations: 500 символов по 2 байта = 1000 байт
- type : 50 символов * 2 байта = 100 байт.
- course : 50 символов * 2 байта = 100 байт.

Итого: 2300 байт

Analysis:

- name: 50 символов по 2 байта = 100 байт
- source : 50 символов по 2 байта = 100 байт

Итого: 200 байт

Связи
Describe:

- weight: 4 байта

Итого: 4 байта

Cause:

- weight: 4 байта

Итого: 40 байта

Таблица 1 - расчёт "чистого" объёма данных (в байтах)

Label	**Количество**	**Размер (в байтах)**
Patient	N	851 * N
Appeal	7 * N	56 * N
Symptom	133	146300
Disease	41	86100
Analysis	133	13300
Describe	4	820
Cause	4	820

Таким образом на основе таблицы 1, получаем следующую функцию от количество пользователей *N*:

vclean(N)=(851+56)*N+146300+86100+13300+820+820=907*N+247340

Избыточность данных

Теперь учитываем, что бд Neo4j пользуется такими элементами, как узел и связь, а также позволяет накладывать такие метаданные, как классификационные метки. С учётом этого посчитаем фактический размер получающейся базы данных.

Patient:

- Узел: 15 байт
- Метка Patient : 7 символов по 1 байту = 7 байт
- fullname: 100 символов по 2 байта = 200 байт
- mail: 255 символов по 2 байта = 510 байт
- password: 64 символа по 2 байта = 128 байт

- sex: 14 байт
- birthday : 8 байт
- last_update : 8 байт
- registration_date : 8 байт
- height: 4 байта
- weight: 4 байта

Итого: 873 байт

Appeal:

- Узел: 15 байт
- Метка Appeal : 6 символов по 1 байту = 6 байт
- date: 8 байт
- complaints: 500 символов * 2 байта = 1000 байт

Итого: 1029 байт

Symptom:

- Узел: 15 байт
- Метка Symptom : 7 символов по 1 байту = 7 байт
- name: 50 символов по 2 байта = 100 байт
- description: 500 символов по 2 байта = 1000 байт

Итого: 1122 байт

Disease:

- Узел: 15 байт
- Метка Disease: 7 символов по 1 байту = 7 байт
- name: 50 символов по 2 байта = 100 байт
- description: 500 символов по 2 байта = 1000 байт
- recommendations: 500 символов по 2 байта = 1000 байт
- type : 50 символов * 2 байта = 100 байт.
- course : 50 символов * 2 байта = 100 байт.

Итого: 2322 байт

Analysis:

- Узел: 15 байт
- Метка Analysis: 8 символов по 1 байту = 8 байт
- name: 50 символов по 2 байта = 100 байт
- source : 50 символов по 2 байта = 100 байт

Итого: 223 байт

Связи

Create:

- Связь: 31 байт
- Метка Create: 6 символов по 1 байту = 6 байт

Итого: 37 байт

Belong:

- Связь: 31 байт
- Метка Belong: 6 символов по 1 байту = 6 байт

Итого: 37 байт

Contain:

- Связь: 31 байт
- Метка Contain: 7 символов по 1 байту = 7 байт

Итого: 38 байт

Confirm:

- Связь: 31 байт
- Метка Confirm: 7 символов по 1 байту = 7 байт

Итого: 38 байт

Predict:

- Связь: 31 байт
- Метка Predict: 7 символов по 1 байту = 7 байт

Итого: 38 байт

Describe:

- Связь: 31 байт
- Метка Describe: 8 символов по 1 байту = 8 байт
- weight: 4 байта

Итого: 43 байт

Cause:

- Связь: 31 байт
- Метка Cause: 5 символов по 1 байту = 5 байт
- weight: 4 байта

Итого: 40 байт

Таблица 2 - размер всех коллекций (в байтах)

Label	**Количество**	**Размер (в байтах)**
Patient	N	873 * N
Appeal	7 * N	203 * N
Symptom	133	149226
Disease	41	87002
Analysis	133	16359
Create	7 * N	259 * N
Belong	7 * N	259 * N
Contain	35 * N	1330 * N
Confirm	133	5054
Predict	70 * N	2660 * N
Describe	205	8815
Cause	205	8200

Получаем следующую функцию от количество пользователей *N* на основе таблицы 2:

vcommon(N)=(873+203+259+259+1330+2660)*N+149226+87002+16359+5054+8815+8200=5584*N+274656

Таким образом, коэффициент избыточности модели будет равен (см. рисунок 3)

vcommon/vclean=5584*N+274656907*N+247340

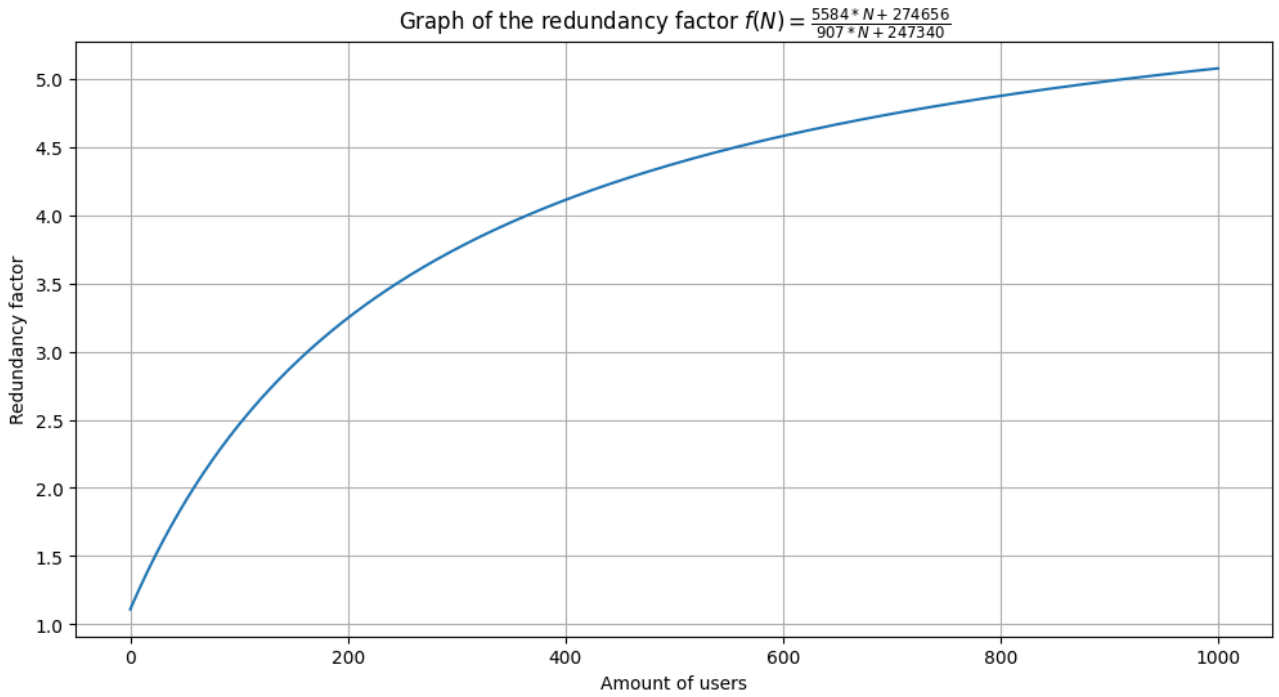


Рисунок 3 - (График коэффициента избыточности)

Направление роста модели при увеличении количества объектов каждой сущности

Увеличение количества узлов типа *Patient* приводит к линейному росту, равному собственному размеру данной сущности. Таким образом, рост составит f(n) = 873n байт.

Увеличение количества узлов типа *Appeal* приводит к появлению связей следующий типов: *Create*, *Belong*, *Contain* и *Predict*. В худшем случае, пользователь введёт все симптомы, которые будут характеризовать все заболевания. Таким образом, рост составит f(n) = (29 + 2 * 37 + 133 + 41) * n = 277n байт.

Увеличение количества узлов типа *Symptom* может привести к росту взвешенных связей типов Describe и Cause, а также к росту связей с определёнными анализами. В худшем случае, будем считать, что симптом свяжется со всеми анализами и заболеваниями. Таким образом, рост составит f(n) = (1122 + 41 * (43 + 40) + 133 * 38) * n = 9579n байт.

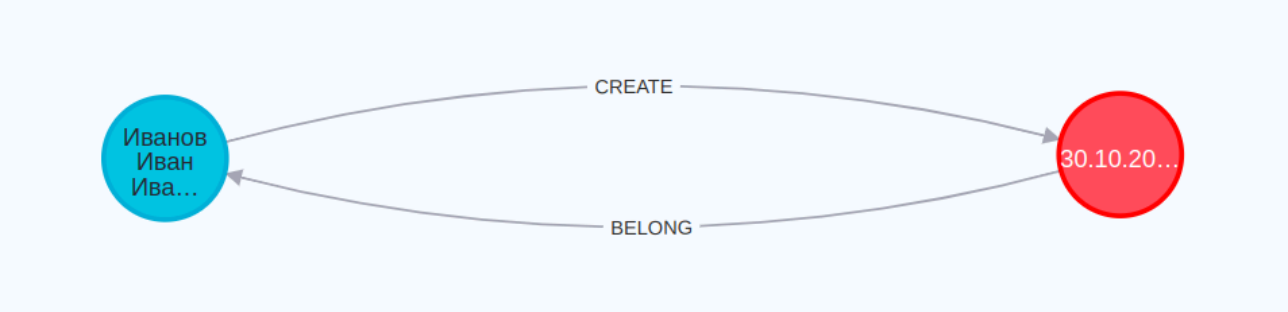
Увеличение количества узлов типа *Disease* может привести к росту взвешенных связей типов *Describe* и *Cause*. В худшем случае, будем считать,

что заболевание свяжется со всеми симптомами. Таким образом, рост составит f(n) = (2122 + 133 * (43 + 40)) * n = 13161n байт.

Увеличение количества узлов типа *Analysis* может привести к росту связей типа *Confirm*. В худшем случае, анализ будет позволять определять любой симптом. Таким образом, рост составит f(n) = (123 + 133 * 38)*n = 5177n байт.

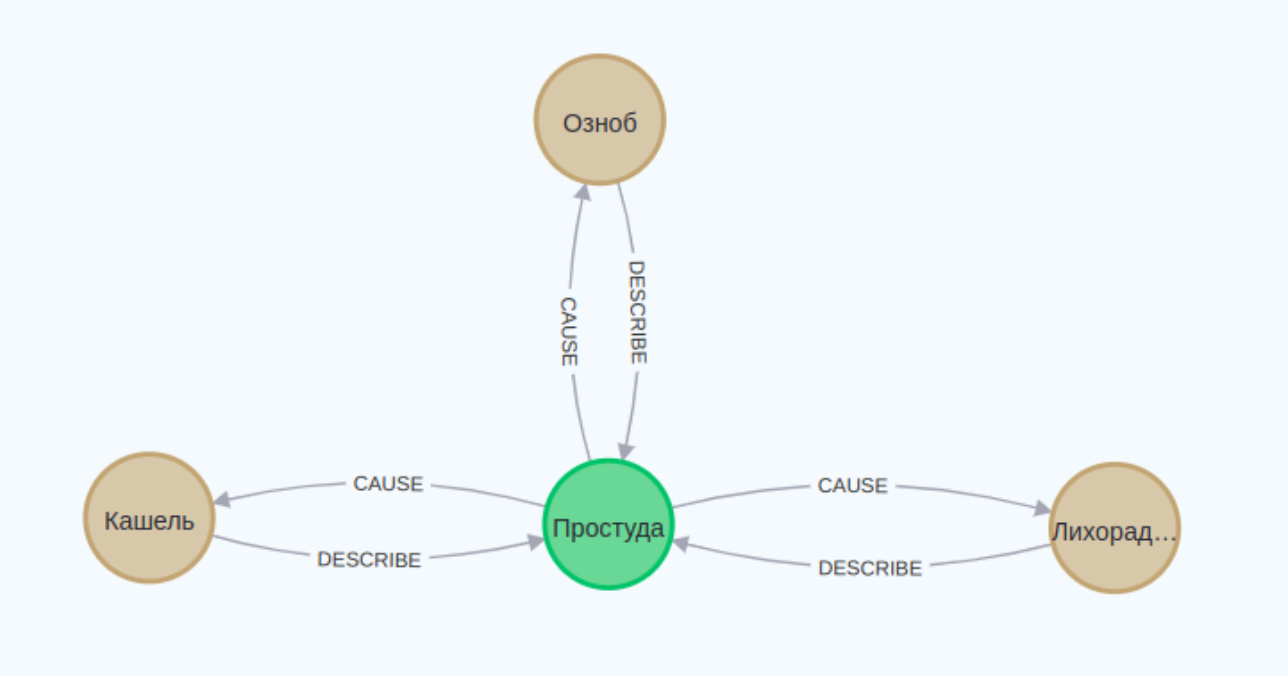
Примеры данных

Пользователь и его обращение (см. рисунок 4)



(Рисунок 4 - пользователь и его обращение)

Болезнь и её симптомы (см. рисунок 5)



(Рисунок 5 - болезнь и её симптомы)

Примеры запросов

Регистрация пользователя

Merge (:Patient {fullname: "Иванов Иван Иванович", email: "iiivanov@etu.ru", password: "securepassword123", "sex": "Male", height: 180, weight: 75, registration_date: "29.10.2024 23:48:13", last_update: "30.10.2024 21:54:13"})

Количество запросов для совершения UseCase: 1

Количество задействованных коллекций: 1

Создание обращения

match (p: Patient {fullname: "Иванов Иван Иванович"})

match (s1:Symptom {name: "Озноб"})-[:DESCRIBE]->(d1)

match (s2:Symptom {name: "Лихорадка"})-[:DESCRIBE]->(d2)

match (s3:Symptom {name: "Кашель"})-[:DESCRIBE]->(d3)

merge (p)-[:CREATE]->(a: Appeal {date: "24.05.2023 15:31:16", complaints: "-"})-[:BELONG]->(p)

merge (d1)-[:PREDICT]->(a)-[:CONTAIN]->(s1)

merge (d2)-[:PREDICT]->(a)-[:CONTAIN]->(s2)

merge (d3)-[:PREDICT]->(a)-[:CONTAIN]->(s3)

Количество запросов для совершения UseCase: 1

Количество задействованных коллекций: 4

Получение статистики распространённости симптома среди пациентов

match (a:Appeal)-[:BELONG]->(p: Patient)

where "20.05.2023" < a.date and a.date < "27.05.2023" and p.sex = True

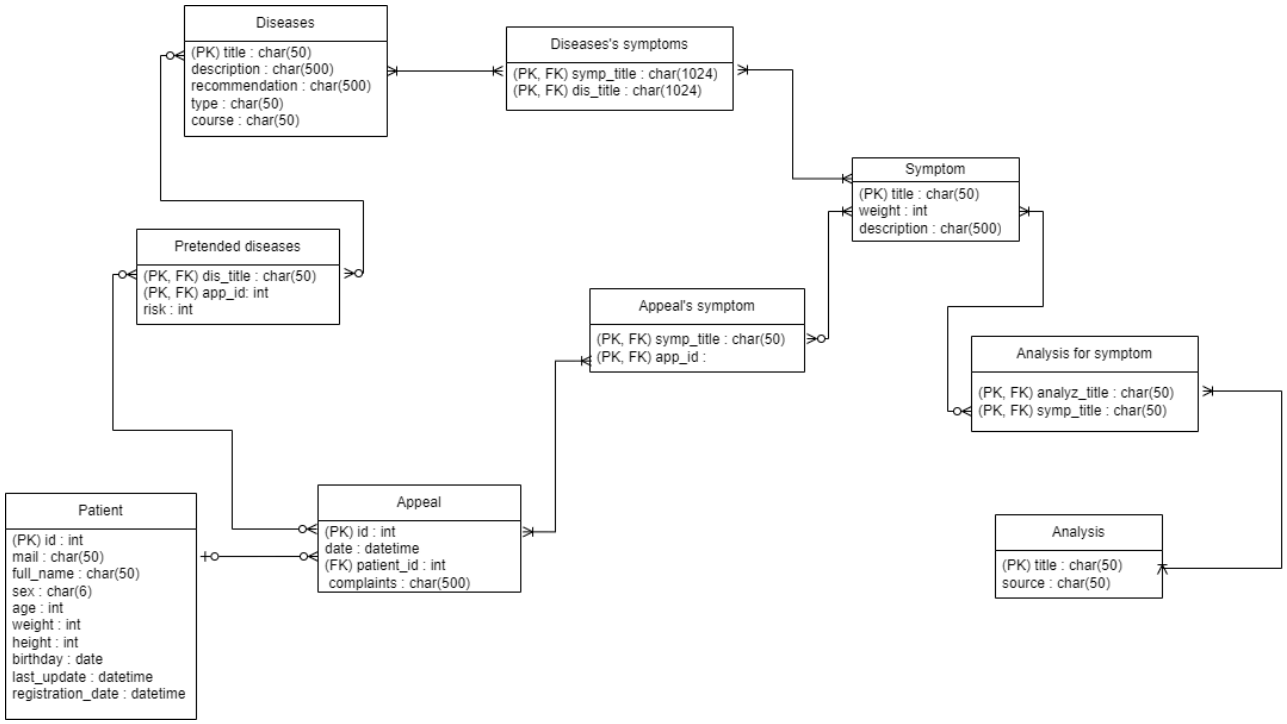
RETURN a

Количество запросов для совершения UseCase: 1

Количество задействованных коллекций: 2

2.2. Реляционная модель данных

Реляционную модель данных можно описать следующей схемой (см. рисунок 6).



(Рисунок 6 - Схема реляционной модели)

Описание назначений коллекций, типов данных и сущностей

Patient – коллекция, которая хранит в себе данные о всех зарегистрированных на сайте пользователях:

- id : int – индификатор для обращения. Обладает свойством автоинкримента. Составной первичный ключ.
- mail : char(255) – почта пользователя, указанная при регистрации. Обязательное поле.
- password : char(64) – пароль от аккаунта для пользователя. Обязательное поле
- full_name : char(100) – ФИО пользователя. Обязательное поле.
- sex : char(6) – пол пользователя. Male ил Female. Необязательное поле.
- birthday : date – День рождения пользователя. Необязательное поле.

- last_update : datetime - время последнего обновления страницы. Обязательное поле.
- registration_date : datetime - время регистрации пользователя на сайте. Обязательное поле
- weight : int – вес пользователя. Необязательное поле.
- height : int – рост пользователя. Необязательное поле.

Необязательные поля указываются для сбора статистики для симптомов.

Appeal – обращение пользователя, которое нужно для сохранения истории поиска болезней пользователя:

- id : int – индификатор для обращения. Обладает свойством автоинкримента. Первичный ключ.
- date : datetime – точное время запроса. Дата показывается пользователю в истории. Также нужно для сбора статистики для симптомов за определённый период. Обязательное поле.
- patient_id : int – индификатор пользователя, к которому привязано обращение. Внешний ключ. Обязательное поле.
- complaints : char(500) - жалобы пациента, с которым пользователь пришёл к доктору. Необязательное поле.

Diaseas – список всех существующих в базе данных болезней, которые могут быть спрогнозированы системой :

- title : char(50) – название болезни. Первичный ключ, т.к. названия болезней не повторяются.
- description : char(500) – текст с описанием болезни. Обязательное поле.
- recommendation : char(500) – текст с рекомендациями по лечению болезни. Обязательное поле.
- type : char(50) - тип заболевания (вирусное, бактериальное, ...). Обязательное поле.

- course : char(50) - течение заболевания (острое, хроническое, ...). Обязательное поле.

Analysis – анализы, которые необходимо сдать, чтобы подтвердить наличие симптома и, соответственно, для подтверждения болезни:

- title : char(50) – название анализа. Первичный ключ.
- source : char(50) - источник анализа (анализ крови на гемоглобин, общий анализ крови и т.д.).

Анализы вынесены в отдельную коллекцию, т. к. у одного симптома может быть сразу несколько анализов для его подтверждения.

Appeal's symptoms – список симптомов, которые ввёл пользователей в рамках одного обращения, использующиеся для поиска возможных заболеваний:

- symp_title : char(50) – название введённого пользователем симптома. Часть составного первичного ключа.
- app_id : int – индификатор обращения. Часть составного первичного ключа.

У обращения должен быть хотя бы один симптом в данной таблице.

Symptoms – список всех существующих в базе данных симптомов:

- symp_title : char(50) – название симптома. Первичный ключ.
- description : char(50) – описание симптома. Обязательное поле.
- weight : int – вес симптома. Симптомы могут иметь разный вес, который определяет, насколько симптом вероятен при заболевании. С его помощью подчитывается процент вероятности наличия заболевания по введённым симптомам. Обязательное поле.

Analysis for symptom – список анализов, которые надо провести для определения симптома:

- symp_title : char(50) – название симптома, для которого определяем необходимые анализы. Часть составного первичного ключа.
- analyz_title : char(50) – название анализа для определения симптома. Часть составного первичного ключа.

Каждый анализ направлен на выявление какого-то симптома, но не у каждого симптома обязан быть анализ для выявления.

Disease's symptoms – у каждой болезни есть свой набор симптомов, которые в совокупности дают 100% вероятность наличия у пользователя болезни:

- symp_title : char(50) – название симптома. Часть составного первичного ключа.
- dis_title : char(50) – название болезни. Часть составного первичного ключа.

У каждой болезни обязан быть набор симптомов, также как и укаждого симптома должна быть хотя бы одна болезнь, в которую он входит.

Predicted diseases – результат выполнения алгоритма, в котором хранятся болезни, вероятность наличия которых у пользователя ненулевая:

- dis_title : char(50) – название болезни. Часть составного первичного ключа.
- app_id : int – индификатор обращения. Часть составного первичного ключа.
- risk : int – процент, который показывает, какова вероятность наличия болезни у пользователя. Составляется по формуле: risk = (Вес выбранных пользователем симптомов) \ (Вес симптомов у вероятной болезни) * 100.

Оценка объема информации, хранимой в модели

Аналогично нереляционной модели предположим, что

- пользователь будет создавать в среднем 7 обращений
- каждый симптом будет иметь в среднем один анализ для своего подтверждения
- каждая болезнь будет иметь в среднем по 5 симптомов
- в каждом обращении будет в среднем по 5 симптомов
- каждое обращение будет прогнозировать с среднем 10 болезней

Будем считать, что каждый символ у нас занимает 2 байта в связи с использованием русского языка.

Рассмотрим объём информации для каждого объекта сущности:

Patien

- id : int – 4 Байта
- mail : char(255) – 510 Байта
- password : char(64) – 128 Байт
- full_name : char(100) – 200 Байт
- sex : char – 6 Байт
- birthday : date - 4 Байта
- registration_date : datetime - 8 Байт
- last_update : datetime - 8 Байт
- age : int – 4 Байта
- weight : int – 4 Байта
- height : int – 4 Байта

Всего: 880 Байт

Appeal

- id : int – 4 Байта
- date : datetime – 8 Байт
- patient_id : int – 4 Байта
- complaints : char(500) - 1000 Байт

Всего: 1016 Байт

Diseases

- title : char(50) – 100 Байт
- description : char(500) – 1000 Байт
- recommendation : char(500) – 1000 Байт
- type : char(50) - 100 Байт
- course : char(50) - 100 Байт

Всего: 2300 Байт

Analysis

- title : char(50) – 100 Байт
- source : char(50) - 100 Байт

Всего: 200 Байт

Appeal's symptoms

- symp_title : char(50) – 100 Байт
- app_id : int – 4 Байта

Всего: 104 Байта

Symptoms

- symp_title : char(50) – 100 Байт
- description : char(500) – 1000 Байт
- weight : int – 4 Байта

Всего: 1104 Байта

Analysis for symptom

- symp_title : char(50) – 100 Байт
- analyz_title : char(50) – 100 Байт

Всего: 200 Байт

Disease's symptoms

- symp_title : char(50) – 100 Байт
- dis_title : char(50) – 100 Байт

Всего: 200 Байт

Predicted diseases

- dis_title : char(50) – 100 Байт
- app_id : int – 4 Байта
- risk : smallint – 2 Байта

Всего: 106 Байт

Таблица 3 - расчёт "чистого" объёма данных (в байтах)

Label	**Количество**	**Размер (в байтах)**
Patient	N	880 * N
Appeal	7 * N	7112 * N
Symptom	133	146832
Disease	41	94300
Analysis	133	26600
Analysis for symptom	133	26600
Disease's symptoms	133	26600
Predicted diseases	7 * N * 10	7420 * N
Appeal's symptoms	7 * N * 5	3640 * N

Таким образом, общий объём на основе таблицы 3 в зависимости от количества пользователей будет выглядеть следующим образом:

$v_{clean}'(N) = 19052 * N + 332432$

Избыточность данных

В качестве альтернативы neo4j в СУБД возьмём PostgreSQL. В данной БД каждая таблица хранится в отдельном файле, которые называют страницей. Объём одной такой страницы равен 8192 Байта. Таким образом, необходимо прибавить это значение для каждой таблицы (всего их 9). Также каждая строка в таблице имеет свои метаданные:

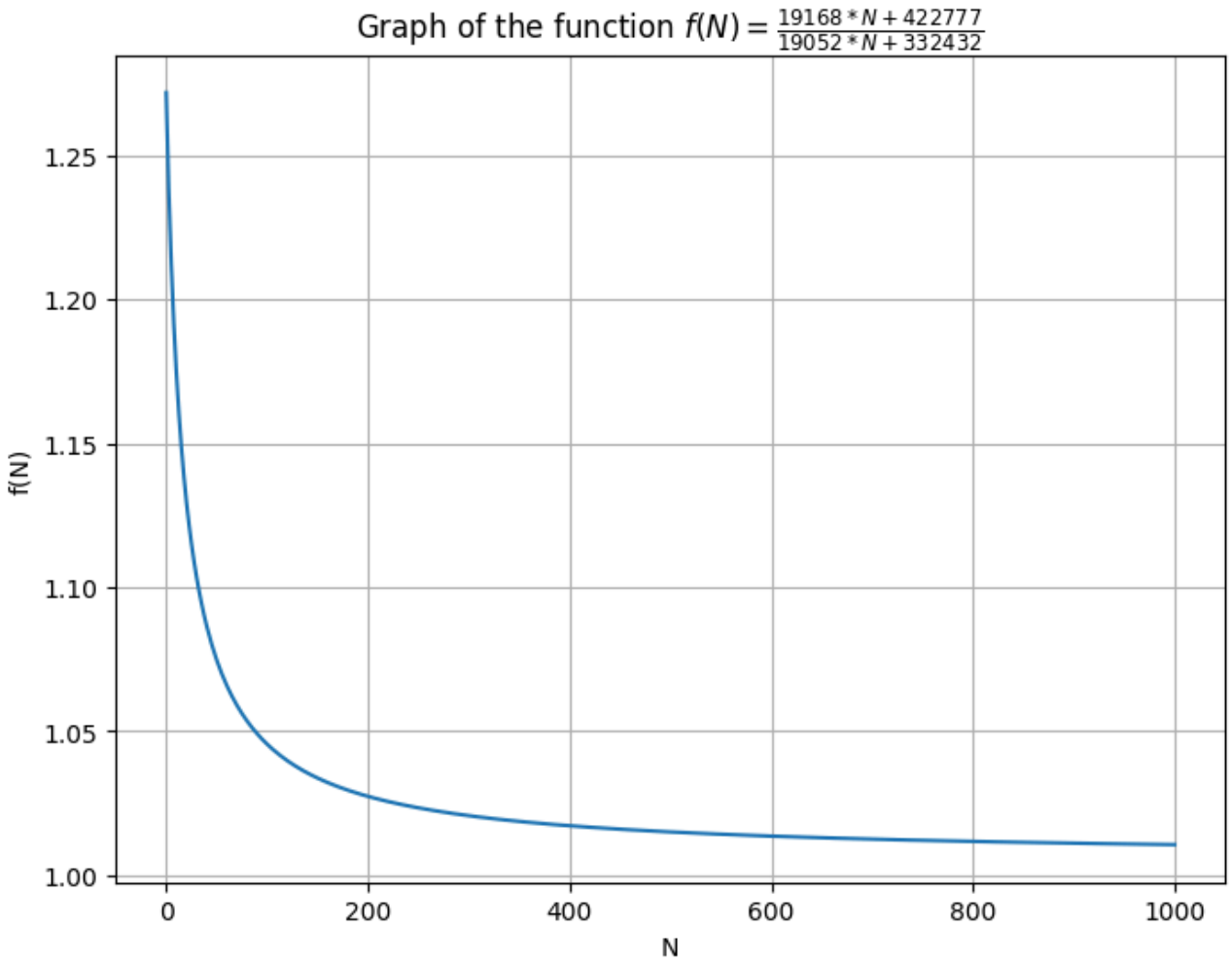
- Заголовок – 23 Байта
- Строковой указатель – 2 Байта

- Смещение данных для выравнивания столбцов, в среднем – 4 Байта на строку

Следовательно, для каждой строки мы также прибавляем 29 Байтов.

Формула оценки избыточности будет выглядеть так (см. рисунок 7):

v*common*/vclean = (19168 * N + 422777)/(19052 * N + 332432)



(Рисунок 7 - оценка избыточности)

По графику видно, что с ростом количества пользователей, разница между чистыми данными и полным объёмом будет уменьшаться. Это связано с тем, что трата на размер листа остаётся постоянной, а объём метаданных строки становится мал по сравнению с весом чистых данных.

Направление роста модели при увеличении количества объектов каждой сущности

Patient

Увеличение количества строк приводит к линейному росту, равному собственному размеру данной сущности.

f(n) = 880*n Байт

Appeal

Увеличение количества строк приводит к созданию новых связей в таблицах Appeal's symptoms и Predicted diseases. В худшем случае, пользователь введёт все симптомы, которые будут характеризовать все заболевания.

f(n) = (16 + 133*104 + 41*106) * n = 18194*n Байт

Symptoms

Увеличение количества строк приводит к созданию новых связей в таблице Disease's symptoms. В худшем случае, симптом будет у всех болезней.

f(n) = (1104 + 41*200) * n = 9304*n Байт

Analysis

Увеличение количества строк приводит к созданию новых связей в таблице Analysis for symptom. В худшем случае, анализ будет необходим для выявления всех симптомов.

f(n) = (100 + 133*200) * n = 26700*n Байт

При это метаданные будут расти согласно этой функции для каждой из таблиц:

f(n) = 29*n Байт

2.3. Сравнение моделей

Удельный объем информации

Рассмотрим, какой объём памяти нужен для нереляционной модели и СУБД, сравним их:

neo4j: 5584 * N + 274656 Байт

PostgreSQL: 19052 * N + 332432 Байт

Как можно заметить, модель в СУБД требует большего хранилища при одинаковом наборе данных. Однако у неё меньшая избыточность данных с увеличением числа пользователей (см. пункт "Избыточность модели"). Это связано с тем, что на метаданные выделяется меньше места по сравнению с neo4j.

Но однозначно можно сказать, что neo4j выигрывает у PostgreSQL в этой задаче по затраченной памяти.

Запросы по отдельным юзкейсам:

Usecase	**Запросов СУБД**	**Коллекций СУБД**	**Запросов neo4j**	**Коллекций neo4j**
Регистрация пользователя	1	1	1	1
Создание обращения	2	2	1	4
Получение статистики	1	2	1	2

Из этой таблицы можно сказать, что neo4j использует меньше или столько же запросов для выполнения юзкейса по сравнению с PostgreSQL, однако для их выполнения затрагивается больше или столько же коллекций.

Таким образом, можно сказать, что для данной задачи лучше подходит neo4j, а не реляционная модель данных. Из явных преимуществ можно выделить разницу в использованной памяти, которая значительно лучше в neo4j. Особенно это ощущается на больших объёмах данных.

Также из плюсов neo4j можно выделить графовую систему БД, которая оптимизирует поиск болезней и определения их рисков при анализе симптомов пользователя, что в данном проекте является основной задачей.

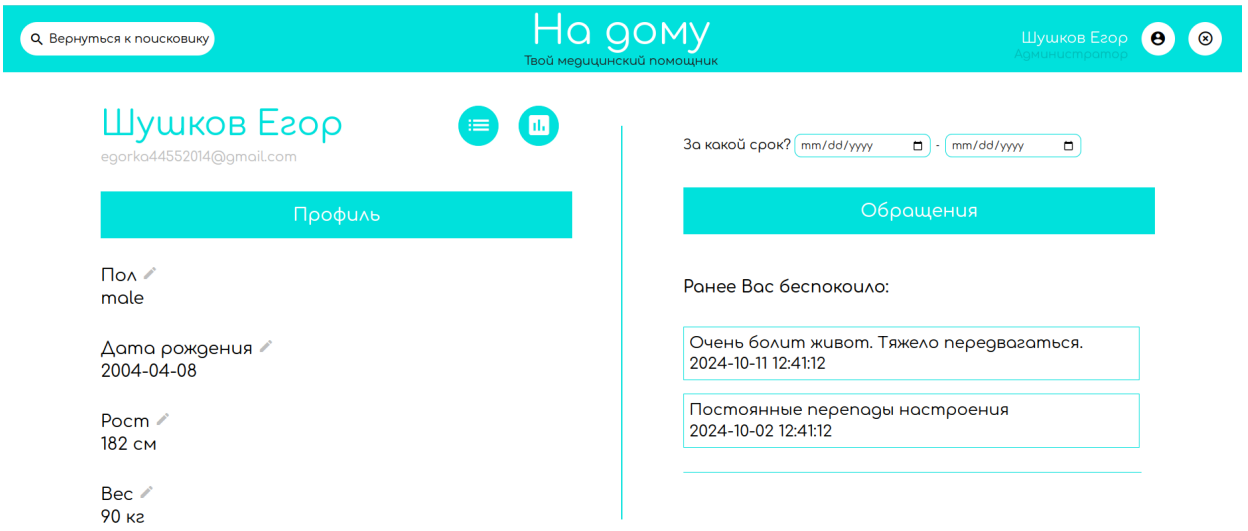
3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Backend-сторона

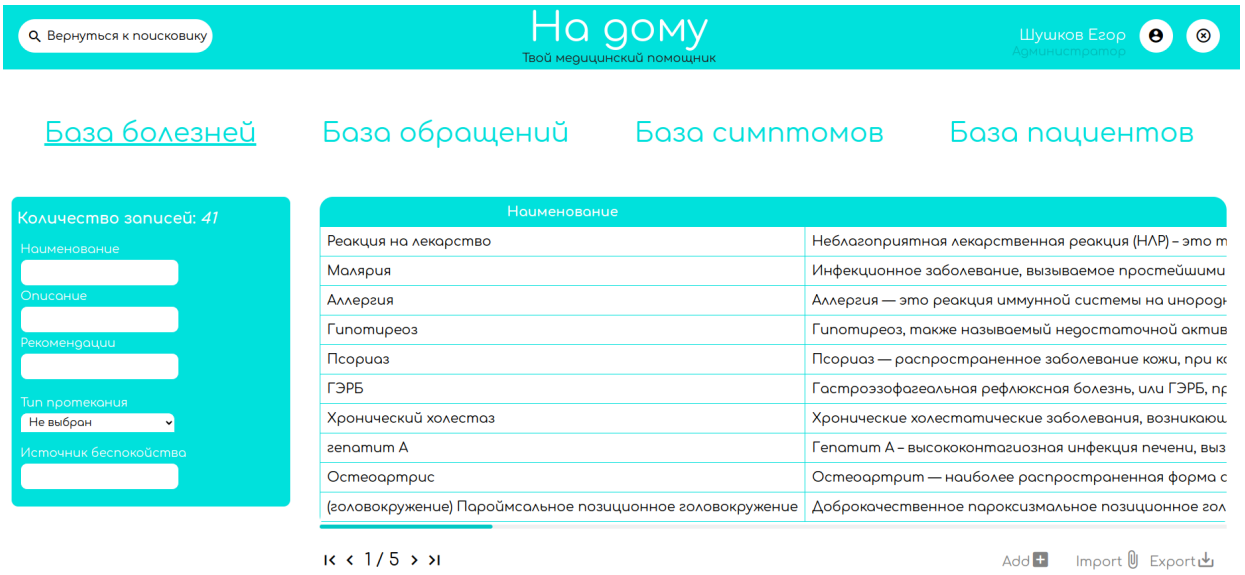
Backend представляет собой приложение на Python с использованием фруймворка Flask. Главная задача Backend - связь пользовательского интерфейса и базы данных. Коммуникация с интерфейсом заключается в получении обращений на определённые endpoint'ы, каждый из которых служит для возвращения конкретной информации из БД. С БД же сервер взаимодействует посредством составления Neo4j запросов.

3.2. Frontend-сторона

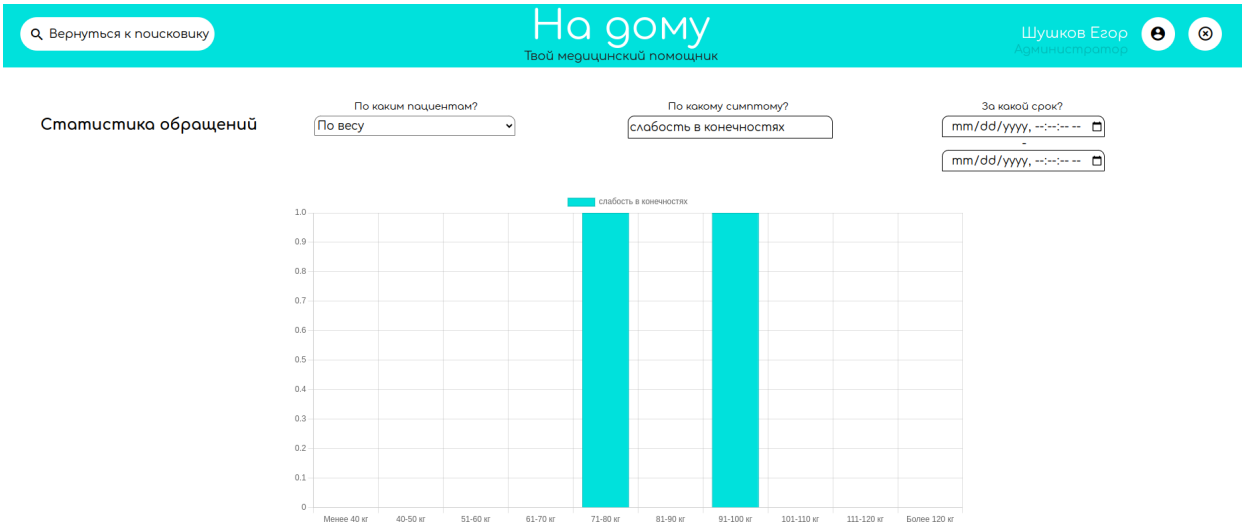
Frontend представляет собой приложение на TypeScript с использованием фреймворка Angular. Ниже представлены скрины некоторых экранов приложения:



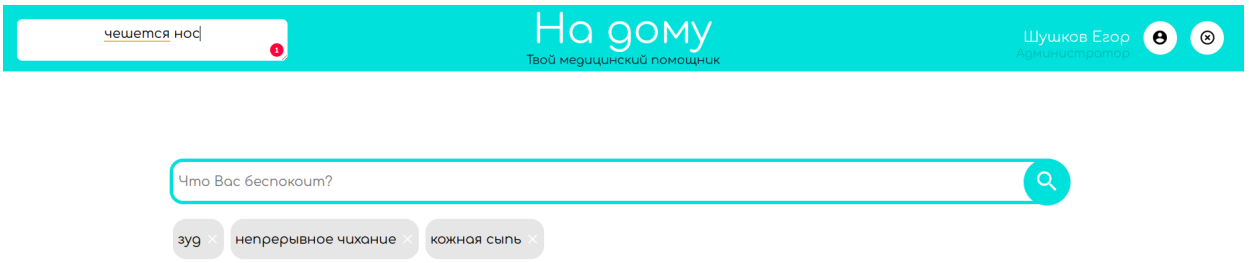
(Рисунок 8 - профиль пациента)



(Рисунок 9 - база данных)



(Рисунок 10 - статистика по обращениям)



(Рисунок 11 - сбор анамнеза)

(Рисунок 12 - диагностирование)

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была разработана система быстрого медицинского диагностирования на основе графовой базы данных Neo4j. Основная цель проекта заключалась в создании приложения, которое позволяет пользователям на основе набора симптомов получать информацию о возможных заболеваниях, рекомендациях по их профилактике и необходимых анализах для подтверждения диагноза.

Достигнутые результаты

1. Разработка модели данных:
   - Создана нереляционная модель данных, включающая узлы "Пациент", "Обращение", "Заболевание", "Симптом", "Анализ" и их взаимосвязи.
   - Разработана реляционная модель данных для сравнения с нереляционной моделью.
2. Реализация приложения:
   - Backend: Приложение на Python с использованием фреймворка Flask, обеспечивающее взаимодействие с базой данных Neo4j.
   - Frontend: Приложение на TypeScript с использованием фреймворка Angular, предоставляющее удобный пользовательский интерфейс.
3. Функциональность:
   - Возможность регистрации и входа пользователей.
   - Ввод симптомов и получение возможных диагнозов.
   - Отображение рекомендаций по профилактике и необходимых анализов.
   - Хранение и управление данными о пациентах, их обращениях, заболеваниях, симптомах и анализах.
   - Административные функции для просмотра записей в базе данных и статистического анализа обращений.

4. Логический вывод:
    - Использование алгоритмов логического вывода для оценки вероятности диагноза на основе введенных симптомов.

Цель проекта была достигнута: разработано приложение, которое эффективно решает поставленные задачи. Пользователи могут быстро получать информацию о возможных заболеваниях на основе введенных симптомов, что соответствует основной цели проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Медицинский датасет // Kaggle. URL: https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset?select=dataset.csv

2. Guia J., Soares V. G., Bernardino J. Graph Databases: Neo4j Analysis //ICEIS (1). – 2017. – С. 351-356.

3. Проект быстрого диагностирования. // GitHub Repository URL: https://github.com/moevm/nosql2h24-diseases

4. Микрофреймворк для создания веб-приложений // Flask. URL: https://flask.palletsprojects.com/en/stable/

5. Язык программирования // Python. URL: https://www.python.org/

6. фреймворк для создания динамических и интерактивных веб-приложений // Angular. URL: https://angular.dev/overview

7. Язык программирования // TypeScript. URL: https://www.typescriptlang.org/

ПРИЛОЖЕНИЕ А
ЗАПУСК ПРИЛОЖЕНИЯ

1) Для запуска требуется выполнить последовательно следующий алгоритм

```
git clone
https://github.com/moevm/nosql2h24-diseases.git
cd nosql2h24-diseases/
sudo docker compose build --no-cache
sudo docker compose up
```

После чего нужно перейти по адресу 127.0.0.1:8080

2) Отладочные пользователи

Имя	**Логин**	**Пароль**	**Права администратора**
Прошичев Александр	kaidux22@gmail.com	1234	Отсутствуют
Шушков Егор	egorka44552014@gmail.com	3214	Присутствуют

3) Требования
   - Docker
   - Ubuntu 22.04+