

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»
Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
на тему
**ИНТЕЛЛЕКТУАЛЬНАЯ ЭКСПЕРТНАЯ
СИСТЕМА ДИАГНОСТИКИ И УЧЕТА
ПАЦИЕНТОВ МЕДИЦИНСКОГО ЦЕНТРА**

Студент Андрей Пронькин 28.05.2024 (подпись, дата) Пронькин Андрей Александрович (ФИО полностью)

Руководитель А.Г. Финогеев 28.05.24 (подпись, дата) Финогеев А.Г. (инициалы, фамилия)

Нормоконтролёр А.С. Бычков 11.06.2024 (подпись, дата) Бычков А. С. (инициалы, фамилия)

Рецензент А.И. Васильев (подпись, дата) (фамилия, инициалы)
Васильев А.И. (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой М.А. Митрохин (подпись) Митрохин М. А. (фамилия, инициалы)

Работа защищена с отметкой (протокол заседания ГЭК от 14.06.24 № 14)

Секретарь ГЭК Е.И. Гурин (подпись) Гурин Е. И. (инициалы, фамилия)

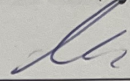
Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой

 М. А. Митрохин
«13» 09 2023 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

1. Студент Пронькин Андрей Александрович, гр. 22ВВиМ1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы: «Интеллектуальная экспертная система диагностики и учета пациентов медицинского центра»

Тема утверждена приказом ПГУ № 765/о от «13» сентября 2022 г.

3. Руководитель магистерской диссертации д.т.н., профессор кафедры Финогеев А.Г.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: диагностика заболеваний и учет пациентов на основании введенных симптомов и медицинских данных.

Моделирование и обучение нейросетевого алгоритма для решения задачи классификации и предсказания заболеваний в области медицинской диагностики.

5. Перечень подлежащих разработке вопросов

1. Исследовать существующие архитектуры нейросетевых алгоритмов и современные инструменты для работы с нейронными сетями, отобрать

наиболее подходящие.

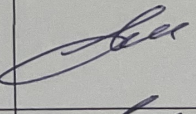
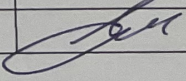
2. Провести анализ исходных данных, на их основе сформировать обучающий набор.

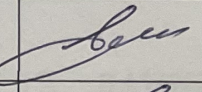
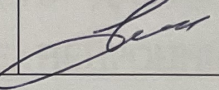
3. Обучить подобранные модели нейронных сетей и провести сравнительный анализ результатов обучения.

4. Разработать:

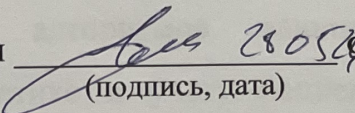
1. Модуль ввода данных и описания симптомов болезни
2. Модуль распознавания симптомов
3. Модуль классификации вероятных симптомов по заболеванию
4. Модуль предварительного диагностирования с определением вероятного заболевания
5. Модуль определения требуемых врачей
6. Модуль рекомендательного выбора конкретного специалиста по максимальной вероятности выявленного заболевания
7. Модуль записи к врачу
8. Модуль формирования медицинской карты пациента с перечнем симптомов вероятного заболевания и рекомендованный врачей
9. База данных пациентов
10. Модуль управления базой данных

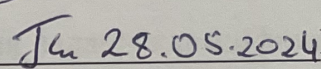
6. Календарный график выполнения работы

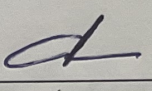
№ п/п	Наименование этапов работы	Объем м работы	Срок выполнения	Подпись руководителя
1.	Исследование существующих архитектур нейросетевых алгоритмов и современных инструментов для работы с нейронными сетями	15%	13.09.2022 – 18.12.2022	
2.	Проведение анализа исходных данных и формирование обучающего набора	35%	19.12.2022 – 11.06.2023	
3.	Обучение подобранных моделей	20%	12.06.2023	

	нейронных сетей, проведение сравнительного анализа результатов обучения		– 15.10.2023	
4.	Проектирование и разработка модулей системы	20%	16.10.2023 – 17.03.2024	
5.	Оформление пояснительной записки	10%	18.03.2024 – 19.05.2024	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации  28.05.2024 Финогеев А.Г.
(подпись, дата)

Задание к исполнению принял студент  28.05.2024 Пронькин А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  11.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 113 листов, 23 рисунка, 1 таблица, 63 источника, 3 приложения

АНАЛИЗ ДАННЫХ, АНАЛИЗ ДАННЫХ, БЕЗОПАСНОСТЬ ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, МЕТОД ОПОРНЫХ ВЕКТОРОВ, PYTHON, SQLITE, SCIKIT-LEARN

Цель работы - исследование алгоритмов машинного обучения применительно к медицинской диагностике и учету пациентов, выявление наиболее подходящей модели для решения задачи классификации заболеваний. Объектом исследования является комплекс алгоритмов машинного обучения, основанных на нейронных сетях.

В результате проведенной работы были исследованы различные нейросетевые алгоритмы, выявлена наиболее успешная модель, обучен конкретный экземпляр сети, а также спроектирована интеллектуальная экспертная система для диагностики и учета пациентов медицинского центра.

Для исследования и реализации было применено большое количество современных технологий для разработки нейронных сетей, такие как язык программирования Python, а также ряд библиотек: scikit-learn, numpy, matplotlib, pickle.

					ПГУ1.090401.56.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Грохьки	ГК	28.05	Интеллектуальная экспертная система диагностики и учета пациентов медицинского центра	Лит.	Лист	Листов
Провер.		Грохьки	ГК	11.06.24			2	113
Н.Контр						ФВТ, каф. ВТ		
Утверд.		Мирохон		11.06.24		гр. 22ВВИМ1		
					Пояснительная записка			

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Анализ технического задания и постановка задачи	8
1.1 Требования к системе	8
1.2 Архитектура системы	8
1.2.1 Разработка модуля ввода данных и описания симптомов болезни ..	10
1.2.2 Разработка модуля распознавания симптомов	10
1.2.3 Модуль классификации вероятных симптомов по заболеванию ..	10
1.2.4 Модуль предварительного диагностирования с определением вероятного заболевания	11
1.2.5 Модуль определения требуемых врачей	11
1.2.6 Модуль рекомендательного выбора конкретного специалиста по максимальной вероятности выявленного заболевания	11
1.2.7 Модуль записи к врачу	12
1.2.8 Модуль формирования медицинской карты пациента с перечнем симптомов вероятного заболевания и рекомендованных врачей	12
1.2.9 Разработка базы данных пациентов	12
1.2.10 Модуль управления базой данных	13
2 Обзор существующих экспертных систем для диагностики и учета пациентов	14
2.1 История развития экспертных систем в медицине	14
2.2 Принципы работы экспертных систем для диагностики	17
2.3 Применение экспертных систем для учета пациентов	19
3 Обзор существующих экспертных систем для диагностики и учета пациентов	22
3.1 Определение требований к системе	22
3.2 Выбор методов и алгоритмов для реализации системы	23
3.3 Проектирование архитектуры системы	26

4 Обзор существующих экспертных систем для диагностики и учета пациентов	31
4.1 Определение и принцип работы машины опорных векторов	40
4.2 Обучение модели машины опорных векторов	44
4.3 Применение машины опорных векторов для распознавания образов ...	46
4.4 Матрица ошибок	50
5 Реализация	53
5.1 Сборка проекта и nginx	56
6 Запуск и тестирование приложения	61
ЗАКЛЮЧЕНИЕ	71
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	72
ПРИЛОЖЕНИЕ А Листинг программы.	82
ПРИЛОЖЕНИЕ Б UML-диаграммы	101
ПРИЛОЖЕНИЕ В Презентация	106

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

БД – база данных

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базой данных

ЯП – язык программирования

API (Application programming interface) – программный интерфейс приложения

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

NoSQL (not only SQL) – это семейство нереляционных баз данных

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SQL (Structured Query Language) – язык структурированных запросов

ВВЕДЕНИЕ

На данный момент искусственный интеллект активно интегрируется в различные области и сферы деятельности повседневной жизни человека, позволяя решать задачи и проблемы, которые довольно трудно решить методами классического программирования. В медицине технологии искусственного интеллекта открывают новые возможности для диагностики и учета пациентов, что способствует повышению качества медицинских услуг и снижению нагрузки на медицинский персонал.

Актуальность данной работы обусловлена тем, что существующие медицинские информационные системы не всегда эффективно справляются с задачами диагностики и учета пациентов. Традиционные методы часто требуют значительных временных и человеческих ресурсов, а также могут быть подвержены субъективным ошибкам. Внедрение интеллектуальных экспертных систем, использующих алгоритмы машинного обучения, может существенно повысить точность и скорость диагностики, а также улучшить управление медицинскими данными.

Темой данной магистерской диссертации является разработка интеллектуальной экспертной системы для диагностики и учета пациентов медицинского центра. Основная цель работы заключается в исследовании алгоритмов машинного обучения применительно к медицинской диагностике и выявлении наиболее подходящей модели для решения задачи классификации заболеваний.

Объектом исследования является комплекс алгоритмов машинного обучения, основанных на нейронных сетях. В процессе работы были исследованы различные нейросетевые алгоритмы, выявлена наиболее успешная модель, обучен конкретный экземпляр сети, а также спроектирована интеллектуальная экспертная система для диагностики и учета пациентов медицинского центра. Для исследования и реализации было применено большое количество современных технологий для разработки нейронных сетей, таких как

язык программирования Python и ряд библиотек: scikit-learn, numpy, matplotlib, pickle.

Основными задачами работы являются:

- изучение методов машинного обучения, применимых к медицинской диагностике,
- исследование алгоритмов классификации заболеваний,
- разработка и обучение модели нейросети для диагностики заболеваний,
- внедрение модели в интеллектуальную экспертную систему для медицинского центра,
- тестирование и апробация разработанной системы.

Данная работа может быть полезна медицинским учреждениям и исследовательским организациям, занимающимся развитием и внедрением интеллектуальных систем для диагностики и учета пациентов. Предложенные подходы и методы могут быть применены в различных областях медицины и способствовать развитию персонализированной медицины, улучшая качество медицинских услуг и оптимизируя процессы управления данными.

1 Анализ технического задания и постановка задачи

Целью данного проекта является создание современной экспертной системы, призванной оптимизировать процессы диагностики и учета пациентов в медицинском центре. Эта система будет включать в себя различные модули, в том числе для ввода данных о пациентах и симптомов, распознавания симптомов, классификации симптомов по заболеваниям, предварительного диагноза, определения необходимых специалистов, рекомендаций специалистов, записи на прием, формирования медицинских карт пациентов и управления базой данных пациентов.

1.1 Требования к системе

Общие требования:

- при разработке системы необходимо использовать современные технологии и инструменты,
- пользовательский интерфейс должен быть интуитивно понятным и удобным для медицинского персонала,
- система должна обеспечивать высокую точность распознавания и классификации симптомов,
- доступ к системе должен быть защищен посредством надежных протоколов аутентификации и авторизации [1],
- соблюдение действующих правил хранения и защиты персональных данных пациентов является обязательным.

1.2 Архитектура системы

Система должна быть модульной, с четким разделением функциональности на отдельные компоненты, на рисунке 1 изображена схема архитектуры системы.

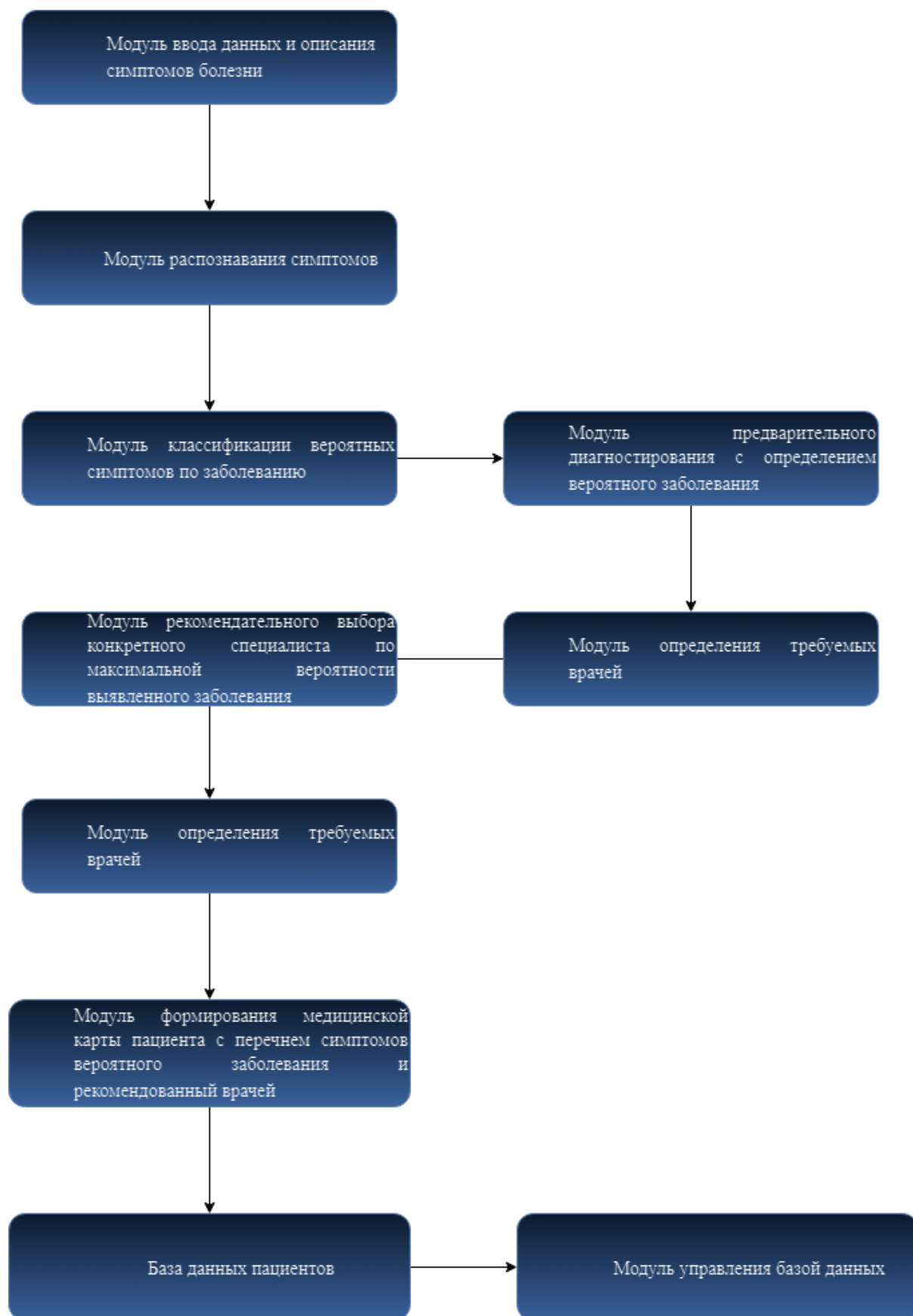


Рисунок 1 - Архитектура системы

Для достижения цели необходимо поставить некоторые задачи.

1.2.1 Разработка модуля ввода данных и описания симптомов болезни

Функциональные требования:

- возможность ввода данных пациента (включая полное имя, возраст, пол, контактную информацию и т. д.),
- функциональность для записи симптомов заболевания в произвольной форме,
- возможность выбора симптомов из заранее определенного списка.

К нефункциональным требованиям относят удобный и интуитивно понятный интерфейс для ввода данных.

1.2.2 Разработка модуля распознавания симптомов

Функциональные требования:

- автоматическая идентификация и интерпретация введенных симптомов,
- реализация алгоритмов машинного обучения и обработки естественного языка [2].

Нефункциональные требования:

- высокая точность распознавания симптомов,
- быстрая обработка введенных данных.

Для обучения модели планируется использовать машинные алгоритмы опорных векторов.

1.2.3 Модуль классификации вероятных симптомов по заболеванию

Функциональные требования:

- классификация выявленных симптомов на возможные заболевания,
- использование обширной базы данных заболеваний и связанных с ними симптомов.

Нефункциональные требования:

- точность классификации должна быть не менее 85%,

- возможность обновления базы данных по заболеваниям по мере необходимости.

1.2.4 Модуль предварительного диагностирования с определением вероятного заболевания

Функциональные требования:

- представление списка потенциальных заболеваний на основе предоставленных симптомов,
- оценка вероятности возникновения каждого потенциального заболевания.

Нефункциональные требования:

- высокоскоростная обработка данных.
- ясное и лаконичное представление результатов для медицинского персонала [2].

1.2.5 Модуль определения требуемых врачей

Функциональные требования:

- создание списка специалистов, необходимых для комплексного обследования пациента,
- учет всех отмеченных симптомов и потенциальных заболеваний.
- К нефункциональным требованиям относят гибкость настройки алгоритмов идентификации специалиста.

1.2.6 Модуль рекомендательного выбора конкретного специалиста по максимальной вероятности выявленного заболевания

Функциональные требования:

- предоставление рекомендаций конкретным специалистам с учетом вероятности диагностированного заболевания,
- учет квалификации и опыта специалистов в рекомендациях.
- К нефункциональным требованиям относят поддержание актуальной базы данных специалистов.

1.2.7 Модуль записи к врачу

Функциональные требования:

- предоставление пациентам возможности записаться на прием к выбранным специалистам,
- отображение расписаний врачей и доступного времени приема [3].

Нефункциональные требования:

- удобный интерфейс для записи на прием,
- интеграция с внешними системами планирования работы врачей.

1.2.8 Модуль формирования медицинской карты пациента с перечнем симптомов вероятного заболевания и рекомендованных врачей

Функциональные требования:

- автоматическое создание и регулярное обновление медицинской карты пациента,
- включение всех введенных симптомов, возможных диагнозов и рекомендаций специалистов в медицинскую карту.
- К нефункциональным требованиям относят обеспечение безопасности и конфиденциальности данных медицинской документации.

1.2.9 Разработка базы данных пациентов

Функциональные требования:

- хранение всех данных о пациенте, включая симптомы, диагнозы и историю посещений,
- обеспечение оперативного доступа к хранимым данным для уполномоченного персонала.

Нефункциональные требования:

- обеспечение высокой надежности и отказоустойчивости системы баз данных.
- соблюдение правил защиты персональных данных [4].

1.2.10 Модуль управления базой данных

Функциональные требования:

- ведение базы данных пациентов,
- реализация механизмов резервного копирования и восстановления данных.

Нефункциональные требования:

- удобный интерфейс для администраторов баз данных,
- масштабируемость для будущего роста и дополнительных данных.

SQLite будет служить запланированной платформой базы данных.

Приложение должно быть совместимо с персональными компьютерами, работающими под управлением ОС Microsoft Windows, в частности версий 7, 8, 8.1 и 10 [5].

2 Обзор существующих экспертных систем для диагностики и учета пациентов

2.1 История развития экспертных систем в медицине

В современном мире здравоохранения экспертные системы стали краеугольным камнем, внося значительный вклад в диагностику пациентов и ведение записей. Эти сложные системы объединяют глубокие знания и обширный опыт медицинских работников, используя методологии искусственного интеллекта для тщательного анализа и интерпретации сложных клинических данных. Целью этой главы является объяснение некоторых выдающихся экспертных систем, которые сыграли важную роль в медицинской сфере, а также описание эволюции экспертных систем в этом секторе.

Система MYCIN является одной из самых известных и широко используемых экспертных систем в области медицинской диагностики и ведения анамнеза. Созданная в 1970-х годах, MYCIN считается новаторской экспертной системой, специально созданной для диагностики инфекционных заболеваний. Основываясь на правилах, сформулированных выдающимися специалистами в области инфекционных заболеваний, MYCIN использовал сложные алгоритмы для принятия решений, основанных на этих экспертных правилах. Способность системы диагностировать инфекции и назначать наиболее эффективные методы лечения заметно повысила производительность труда врачей [6].

После появления MYCIN было создано множество других экспертных систем, каждая из которых ориентирована на отдельные медицинские специализации. Например, система CADUCEUS появилась в 1980-х годах с упором на диагностику и лечение сердечно-сосудистых заболеваний. В него вошли правила и алгоритмы принятия решений, разработанные ведущими кардиологами. Точность, с которой CADUCEUS идентифицировал сердечные заболевания, и его помощь врачам в принятии обоснованных решений о лечении подчеркивали его ценность, отметил А.В. Курочкин [7].

Всплеск достижений в области искусственного интеллекта и машинного обучения в последнее время привел к появлению нового поколения экспертных систем. Эти системы используют все более сложные алгоритмы и модели для анализа и декодирования клинических данных. В качестве примера можно привести Watson for Oncology, детище IBM, созданное для диагностики и разработки стратегий лечения рака. Опираясь на обширное хранилище медицинской литературы, эмпирических клинических данных и коллективную мудрость знатоков онкологии во всем мире, Watson for Oncology использует методы машинного обучения для адаптации рекомендаций по лечению, которые уникально подходят к индивидуальным особенностям каждого пациента [8].

Более того, в отрасли здравоохранения внедряются экспертные системы, предназначенные для тщательного управления записями пациентов и контроля медицинских данных. Например, система Epic разработана для развертывания в больницах и клиниках и облегчает оцифровку записей пациентов, автоматизацию заказов, рецептов и многое другое. Эта система позволяет медицинскому персоналу быстро получать необходимую информацию о пациенте и управлять уходом за пациентами с повышенной эффективностью.

Возникновение экспертных систем в области медицины восходит к новаторским временам вычислительных технологий и зарождению искусственного интеллекта в 1960-х годах. Это был период, характеризующийся острой нехваткой медицинских специалистов, способных диагностировать сложные заболевания и назначать эффективное лечение. Этот дефицит стал катализатором создания экспертных систем, предназначенных для воплощения коллективного опыта и практической мудрости врачей, тем самым позволяя этим системам принимать важные решения в диагностике и терапии пациентов.

Одновременно с появлением первых экспертных систем, таких как MYCIN и DENDRAL, произошло заметное увеличение участия медицинского сообщества в исследованиях и разработках новых систем, адаптированных к различным медицинским дисциплинам. 1980-е годы стали свидетелями

появления таких систем, как CADUCEUS и INTERNIST, которые умело использовались для диагностики и лечения сердечно-сосудистых и внутренних заболеваний.

Последующие десятилетия ознаменовались значительными успехами в области компьютерных технологий и искусственного интеллекта, кульминацией которых стало появление современных экспертных систем. Эти системы характеризуются использованием более сложных алгоритмов и моделей для детального анализа и интерпретации клинических данных. Такие достижения позволили врачам предлагать более точные и надежные рекомендации по диагностике и лечению, тем самым существенно повышая качество оказания медицинской помощи.

Тем не менее, несмотря на значительные достижения в разработке экспертных систем для целей диагностики и регистрации пациентов, эти системы не лишены своих ограничений. Некоторые системы могут представлять собой уровень сложности, который оказывается сложным для медицинских работников и требует специальных знаний в области информационных технологий. Более того, эффективность экспертных систем может ограничиваться динамичным характером медицинской практики и постоянным развитием научных знаний [9].

Короче говоря, экспертные системы, посвященные диагностике и медицинской историографии, играют ключевую роль в современном медицинском ландшафте. Эти системы объединяют глубокие знания и многолетний опыт практикующих врачей, используя возможности искусственного интеллекта и машинного обучения для анализа и разъяснения клинических данных. Ожидается, что с неустанным развитием технологий и возможностей искусственного интеллекта экспертные системы достигнут еще более высокого уровня точности и эффективности, тем самым расширяя возможности врачей принимать обоснованные решения по диагностике и лечению.

2.2 Принципы работы экспертных систем для диагностики

В сфере современной медицины системы экспертной диагностики и тщательного учета пациентов стали важнейшими инновациями, помогающими врачам и медицинским работникам в постановке диагнозов и документировании сведений о пациентах. Эти системы используют принципы искусственного интеллекта и вычислительной техники для точной и быстрой постановки диагнозов, а также для хранения и всестороннего изучения данных пациентов.

Система MYCIN, новатор среди экспертных диагностических систем, была задумана в 1970-х годах и известна своим новаторским использованием правил и эвристики в диагностике инфекционных заболеваний. Основываясь на глубоких знаниях специалистов по инфекционным заболеваниям, MYCIN использовал обширную базу данных, охватывающую симптомы и методы лечения. Образцовая диагностическая точность системы ознаменовала новую эру в медицинской диагностике, ознаменовав важную веху в этой области [10].

В современных условиях было разработано множество диагностических экспертных систем и систем регистрации пациентов, использующих различные методологии и алгоритмы для обработки медицинских данных и облегчения принятия решений. Среди них выделяется система DXplain, созданная в 1987 году. Он работает на основе базы знаний, наполненной данными о симптомах, диагнозах и методах лечения широкого спектра заболеваний. DXplain тщательно оценивает симптомы пациента и предлагает список потенциальных диагнозов вместе с рекомендациями по лечению. Ее широкое использование в клинических условиях и признание в качестве одной из наиболее эффективных диагностических систем подчеркивают ее важность, отметил В.В. Лобанов, отметил В.В. Лобанов [11].

Не менее примечательна введенная в 1990 году система CADUCEUS, которая специализируется на диагностике и лечении сердечно-сосудистых заболеваний. CADUCEUS опирается на базу знаний, наполненную информацией о симптомах, факторах риска и методах лечения, имеющих отношение к

здоровью сердечно-сосудистой системы. Анализируя данные пациентов, CADUCEUS выдает рекомендации по диагностике и лечению. Его успешное применение в клинической практике и доказанная эффективность в диагностике сердечно-сосудистых заболеваний подтверждают его высокое значение в медицинской диагностике [12].

В основе экспертных диагностических систем лежит прочная основа, построенная на базе знаний, дополненная набором правил и алгоритмов, тщательно разработанных для оценки симптомов пациента и принятия обоснованных решений относительно диагноза и лечения. Эти системы являются воплощением передовых методов искусственного интеллекта, включающих индуктивные и дедуктивные рассуждения, обработку естественного языка и комплексный статистический анализ данных.

Центральное место в любой экспертной диагностической системе занимает ее база знаний, хранилище подробной информации о симптомах, потенциальных диагнозах и протоколах лечения множества заболеваний. Этот репозиторий тщательно поддерживается на основе обширного опыта медицинских специалистов в соответствующей области. В этой базе знаний мнения экспертов систематизируются в правила, которые формулируют сложные взаимосвязи между различными симптомами и соответствующими им диагнозами. Иллюстративное правило может утверждать: «Если у пациента появляются такие симптомы, как боль в груди и затрудненное дыхание, это может указывать на начало сердечной недостаточности».

В процессе диагностики пациента экспертная система тщательно применяет эти правила и алгоритмы к сообщаемым пациентом симптомам, проводя тщательное сравнение с данными, хранящимися в ее базе знаний. Такое аналитическое сравнение позволяет системе оценить вероятность различных диагнозов и составить полный список потенциальных состояний, отметил В.В. Лобанов [13]. Кроме того, система способна предлагать рекомендации по лечению, основанные на мудрости медицинских экспертов.

Для обработки данных и принятия решений экспертные системы используют различные методы и алгоритмы. Например, некоторые системы используют индуктивные рассуждения, чтобы выявить скрытые закономерности в данных, тем самым создавая диагностические модели. Эти модели затем способны прогнозировать диагнозы на основе новых данных о пациентах. И наоборот, другие системы используют дедуктивные рассуждения для применения установленных правил и алгоритмов к конкретным симптомам, предъявляемым пациентом, тем самым способствуя принятию точных решений относительно диагностики и лечения.

2.3 Применение экспертных систем для учета пациентов

В современной медицинской практике сложность диагнозов и историй болезни требует повышенной точности и пристального внимания. Экспертные системы, использующие искусственный интеллект, открывают новые возможности для повышения эффективности и точности диагностики, а также для комплексного сбора и анализа данных пациентов. Этот анализ углубляется в различные экспертные системы, используемые в медицине, подчеркивая их роль в управлении медицинской документацией [14].

Одной из первых и наиболее известных экспертных диагностических систем является MYCIN, разработанная в 1976 году. Будучи пионером в диагностике инфекционных заболеваний, MYCIN предоставила рекомендации по лечению, основанные на введенных симптомах и результатах лабораторных исследований. Используя правила, основанные на опыте специалистов по инфекционным заболеваниям, MYCIN достиг высокой диагностической точности. Несмотря на точность, удобство использования системы ограничивалось ее сложностью и необходимостью в значительных компьютерных ресурсах [15].

Еще одна известная экспертная диагностическая система — DXplain, предназначенная для помощи врачам в диагностике сложных клинических случаев. Используя базу данных, охватывающую симптомы, заболевания и

методы лечения, а также алгоритмы машинного обучения, DXplain предлагает диагностические рекомендации. Система отличается высокой диагностической точностью и широкой применимостью, однако для эффективного функционирования необходимы значительные данные и вычислительные ресурсы [16].

Кроме того, внимания заслуживает продукт IBM Watson for Oncology. Эта экспертная система помогает врачам принимать обоснованные решения относительно лечения рака. Благодаря применению алгоритмов искусственного интеллекта и машинного обучения Watson for Oncology анализирует медицинские данные пациентов и формулирует рекомендации по лечению. Система отличается исключительной точностью и эффективностью, хотя требует значительных вычислительных мощностей и доступа к обширным данным [17].

В медицинской практике использование экспертных систем ведения истории болезни играет решающую роль. Эти системы могут упростить сбор и анализ данных о пациентах, тем самым облегчая рабочую нагрузку медицинского персонала и повышая точность и эффективность ведения учета.

Примером таких систем является Электронная медицинская карта (ЭМК). ЭМК — это цифровые версии медицинских карт пациентов, содержащие полную медицинскую информацию, такую как результаты анализов, диагнозы и рецепты. Эти системы позволяют врачам быстро получать доступ к данным пациентов и проводить анализ, чтобы выявить закономерности и тенденции в заболеваниях и эффективности лечения.

Еще одной заслуживающей внимания экспертной системой в области ЭМК является Система поддержки принятия клинических решений (CDS). CDS предлагает врачам рекомендации по диагностике и лечению, анализируя данные пациентов. Используя алгоритмы машинного обучения, система выявляет закономерности и взаимосвязи в данных, предоставляя врачам информацию о наиболее эффективных диагностических и терапевтических методах [18].

Таким образом, экспертные системы, предназначенные для диагностики и ведения пациентов, служат мощным инструментом повышения эффективности и точности медицинской практики. Автоматизируя диагностические процессы, анализ данных и ведение историй болезни пациентов, эти системы повышают качество медицинской помощи и результаты лечения пациентов. Тем не менее, внедрение и использование экспертных систем сопряжено с проблемами, включая необходимость в значительных вычислительных ресурсах и обширном доступе к данным.

3 Обзор существующих экспертных систем для диагностики и учета пациентов

3.1 Определение требований к системе

Проектирование и разработка экспертной системы диагностики и регистрации информации о пациенте — сложная и многогранная задача, требующая тщательного анализа и уточнения системных требований. В этом разделе описываются основные этапы проектирования и разработки такой системы, подчеркивая основные требования, которым она должна соответствовать.

Первоначально этот процесс включает в себя определение целей и задач системы. Основная цель состоит в том, чтобы поставить точный и надежный диагноз путем анализа и интерпретации обширных данных о пациентах и симптомов. Для достижения этой цели система должна эффективно обрабатывать огромные объемы информации, предоставляя рекомендации по лечению и облегчая регистрацию пациентов [19].

Определение системных требований начинается с анализа предметной области, уделяя особое внимание медицине и диагностике заболеваний. Для оптимальной функциональности система должна тщательно изучать существующие методы диагностики и регистрации пациентов, определяя фундаментальные принципы и протоколы, которым следуют врачи при постановке диагноза.

Впоследствии крайне важно установить функциональные требования к системе. К ним относятся возможность сбора и хранения полной информации о пациенте, такой как состояние, симптомы, результаты анализов и предыдущие диагнозы. Более того, система должна анализировать эти данные, чтобы предлагать рекомендации по диагностике и лечению. Он также должен обладать гибкостью, позволяющей обновляться и расширяться по мере появления новых диагностических методов и методологий лечения [20].

При проектировании и разработке экспертной системы определение нефункциональных требований становится ключевым этапом. Среди этих требований первостепенное значение имеет обеспечение точности и надежности диагностики. Система должна предоставлять чрезвычайно точные диагнозы, исключая любую возможность ошибок или ошибочных предположений. Кроме того, простота использования и управления крайне важна, позволяя врачам быстро и эффективно получить доступ к необходимой информации.

Масштабируемость является еще одним жизненно важным требованием. Система должна умело обрабатывать большие объемы данных и одновременно обрабатывать множество пациентов. Гибкость и адаптируемость к развивающимся методологиям диагностики и лечения также необходимы.

Безопасность представляет собой решающую проблему для системы [21]. Защита всех данных пациентов, обеспечение доступа только уполномоченному персоналу требует надежных механизмов аутентификации и авторизации, обеспечивающих конфиденциальность и целостность данных.

Следовательно, при проектировании и разработке экспертной диагностической системы обязательным является определение комплекса требований. Они охватывают функциональные задачи, такие как сбор данных, анализ и рекомендации по диагностике, а также нефункциональные предпосылки, такие как точность, надежность, масштабируемость, безопасность и плавная интеграция с другими системами. Добросовестное выполнение этих требований позволяет создать эффективную и надежную систему, позволяющую медицинским работникам диагностировать и регистрировать пациентов.

3.2 Выбор методов и алгоритмов для реализации системы

Процесс создания и создания экспертной системы, предназначенной для диагностики и ведения медицинской документации, представляет собой многогранную задачу, требующую тщательного выбора методов и алгоритмического применения. В этом сегменте разворачивается исследование, раскрывающее основные этапы концептуализации и построения такой системы, а

также набор методологий и алгоритмов, которые находят применение при ее реализации.

На начальном этапе необходим всесторонний анализ предметной области, чтобы определить системные реквизиты. Это влечет за собой выявление заболеваний и симптомов, которые требуют рассмотрения, определение необходимых данных о пациентах для сбора и анализа, а также определение решений и рекомендаций, которые должна предложить система. Этот процесс расследования включает в себя взаимодействие с медицинскими работниками, изучение соответствующей литературы и стандартов, а также тщательное изучение существующих диагностических систем и программного обеспечения для регистрации пациентов [22].

В дальнейшем структура системы определяется путем разработки ее архитектуры. Этот проект поясняет структурный состав системы, охватывая ее составные элементы и очерчивая взаимодействие между ними. В экспертной системе, предназначенной для диагностики и регистрации пациентов, ключевые компоненты могут включать хранилище знаний, модуль генерации рекомендаций, интерфейс ввода данных о пациентах и блок обработки данных. В рамках этой схемы хранилище знаний содержит исчерпывающую информацию о заболеваниях, симптомах, методах диагностики и методологиях лечения, а также правила, определяющие формулировку рекомендаций. Модуль генерации рекомендаций берет на себя задачу проверки введенных данных и предоставления рекомендаций врачам, а интерфейс ввода данных облегчает ввод информации о пациенте медицинским персоналом. Одновременно блок обработки данных берет на себя ответственность за анализ и хранение этих сопоставленных данных.

При доработке архитектуры системы первостепенное значение приобретает выбор методов и алгоритмов реализации ее разнообразных компонентов. Что касается базы знаний, существует множество методов представления знаний, включая правила производства, семантические сети и

онтологии. Правила производства выражают знания посредством правил «если-то», описывающих условия и действия. Семантические сети представляют знания в виде графа, где узлы обозначают понятия, а ребра символизируют их взаимосвязи, отметил Д.Б. Новиков [23]. Онтологии кодируют знания через классы, свойства и иерархические отношения, в зависимости от требований конкретной предметной области и системы.

Модуль вывода рекомендаций может использовать различные интеллектуальные методы, включая правила вывода, машинное обучение и статистические подходы. Правила вывода позволяют системе тщательно проверять входные данные и применять соответствующие правила для выработки рекомендаций. Методологии машинного обучения позволяют системе усваивать входные данные и формулировать рекомендации, основанные на полученных моделях. Статистические методы облегчают анализ статистических данных, позволяя системе давать рекомендации, основанные на вероятностях.

Для модуля ввода данных о пациенте доступен набор способов ввода, включая формы ввода данных, речевой ввод и сенсорные устройства. Формы ввода данных предоставляют врачам возможность вручную вводить данные о пациенте, включая такие параметры, как частота пульса или температура тела.

Что касается модуля обработки данных, можно использовать различные методы, такие как алгоритмы интеллектуального анализа данных, статистические методологии и подходы машинного обучения. Алгоритмы анализа данных позволяют системе анализировать огромные наборы данных, выявлять закономерности и корреляции. Статистические методологии позволяют системе проводить статистический анализ и получать ценную информацию. Методы машинного обучения позволяют системе учиться на данных и прогнозировать будущие события или результаты, на рисунке 2 изображена схема записи к врачу.

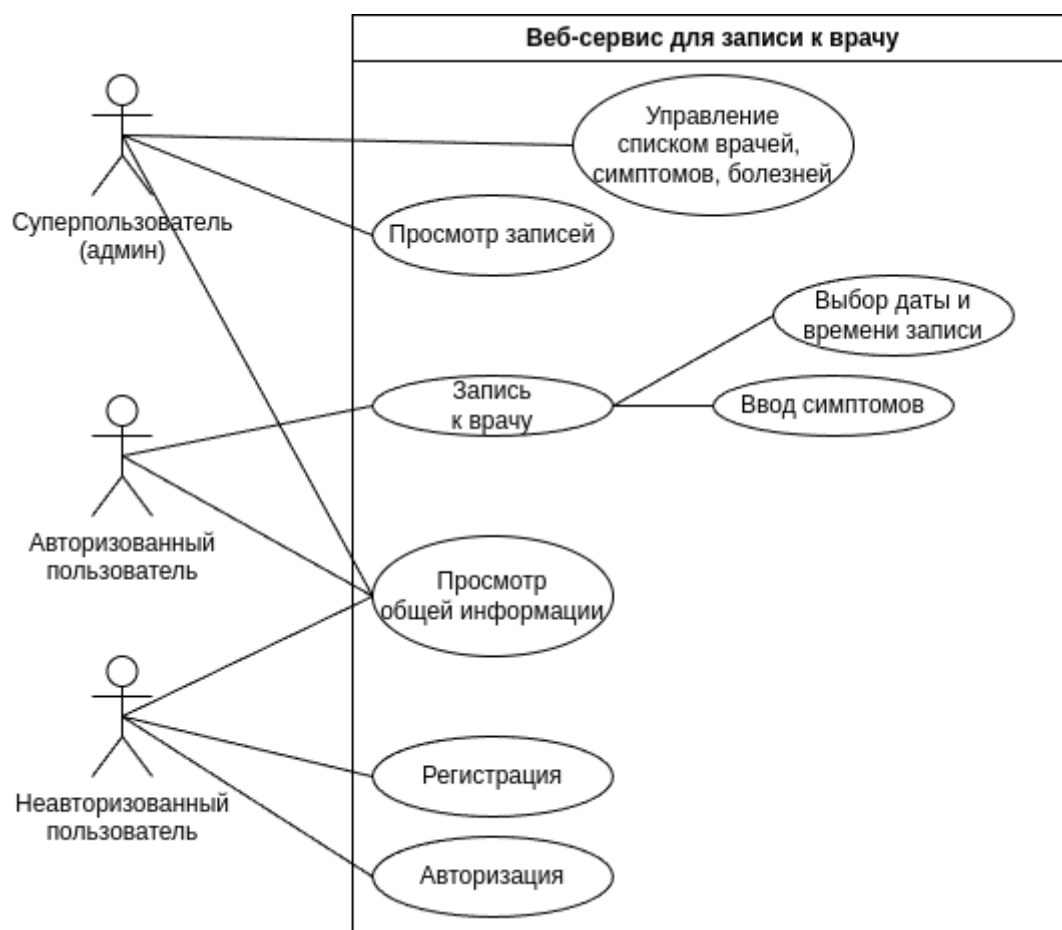


Рисунок 2 - Схема процесса записи к врачу

В процессе создания и совершенствования экспертных систем решающим этапом является тщательное изучение и исправление системных недостатков. Это включает в себя обязательную проверку работоспособности каждого составляющего аспекта и точности рекомендаций, предлагаемых в ответ на входные стимулы. Такие проверки обычно предполагают использование наборов тестовых данных, симуляторов пациентов или прямое взаимодействие с реальными субъектами. Одновременно, на этапе исправления, выявленные аномалии требуют незамедлительного устранения, одновременно используя информацию, полученную в ходе диагностических усилий, для повышения общей эффективности и качества системы.

3.3 Проектирование архитектуры системы

Попытка концептуализации и реализации экспертной системы, предназначенной для диагностики пациентов и ведения записей, влечет за собой

многогранную и сложную задачу. Такая система тщательно разработана для предоставления практикующим врачам диагностических средств и облегчения тщательного документирования данных пациентов. Углубление процесса архитектурного проектирования имеет первостепенное значение в этом дискурсе.

Прежде всего, определение основополагающих компонентов архитектуры экспертной системы представляет собой первый шаг. Здесь лежат основные элементы системы, включающие хранилище знаний, устройство для распространения информации, канал пользовательского интерфейса и модуль, предназначенный для управления досье пациентов. Эта элементарная основа имеет решающее значение [24].

В основе экспертной системы лежит хранилище знаний, служащее ее краеугольным камнем. Этот репозиторий содержит исчерпывающий перечень заболеваний, симптоматических проявлений, диагностических маркеров и перспективных терапевтических методов. Проявляя либо свод правил, либо онтологическую конструкцию, хранилище знаний придает структуру и последовательность. В первой версии правила очерчивают взаимодействие между симптомами, заболеваниями и терапевтическими вмешательствами. И наоборот, в последнем проявлении онтология принимает вид семантической решетки, разграничивающей сущности и их взаимоотношения внутри медицинской области.

Когнитивный аппарат управляет процессом принятия решений, основанным на доступном информационном резервуаре. Используя хранилище знаний и входные данные, относящиеся к симптомам пациента, механизм рассуждения определяет потенциальные диагностические траектории. Этот аппарат вывода может опираться на правила, логические аксиомы или статистические парадигмы. Эффективность и точность метода вывода являются обязательными, обеспечивая надежность диагностических результатов [25].

На рисунке 2 схематически показана траектория прохождения данных из пользовательского интерфейса в репозиторий базы данных. Протокол по умолчанию требует, чтобы пользователь инициировал процесс извлечения HTML-страниц, сценариев и файлов CSS, что необходимо для отображения веб-страницы в интерфейсе браузера. Nginx, которому поручена обработка статических файлов, служит единственным каналом, через который происходит распространение данных. Этот рабочий процесс поясняется на конкретном примере: при обнаружении запроса на аутентификацию пользователь приступает к вводу соответствующих данных в назначенные поля. Впоследствии пользователь отправляет HTTP-запрос, инкапсулирующий эти входные данные, первоначально направляемый в Nginx, а затем пересылаемый в API Django, расположенный на бэкэнде. Примечательно, что Django работает не изолированно, а скорее взаимодействует с сервером Gunicorn, хотя схематическое изображение описывает интеграцию этого сервера в структуру API Django [26]. С помощью внутренней инфраструктуры приложение подключается к репозиторию базы данных, тем самым выполняя проверку, хранение и извлечение данных по мере необходимости. После внутренней обработки ответ передается из репозитория базы данных обратно в пользовательский интерфейс, потенциально охватывая массив компонентов, включая коды ответов HTTP и данные в формате JSON, на рисунке 3 представлена диаграмма последовательности обобщенная для записи к врачу по симптомам.

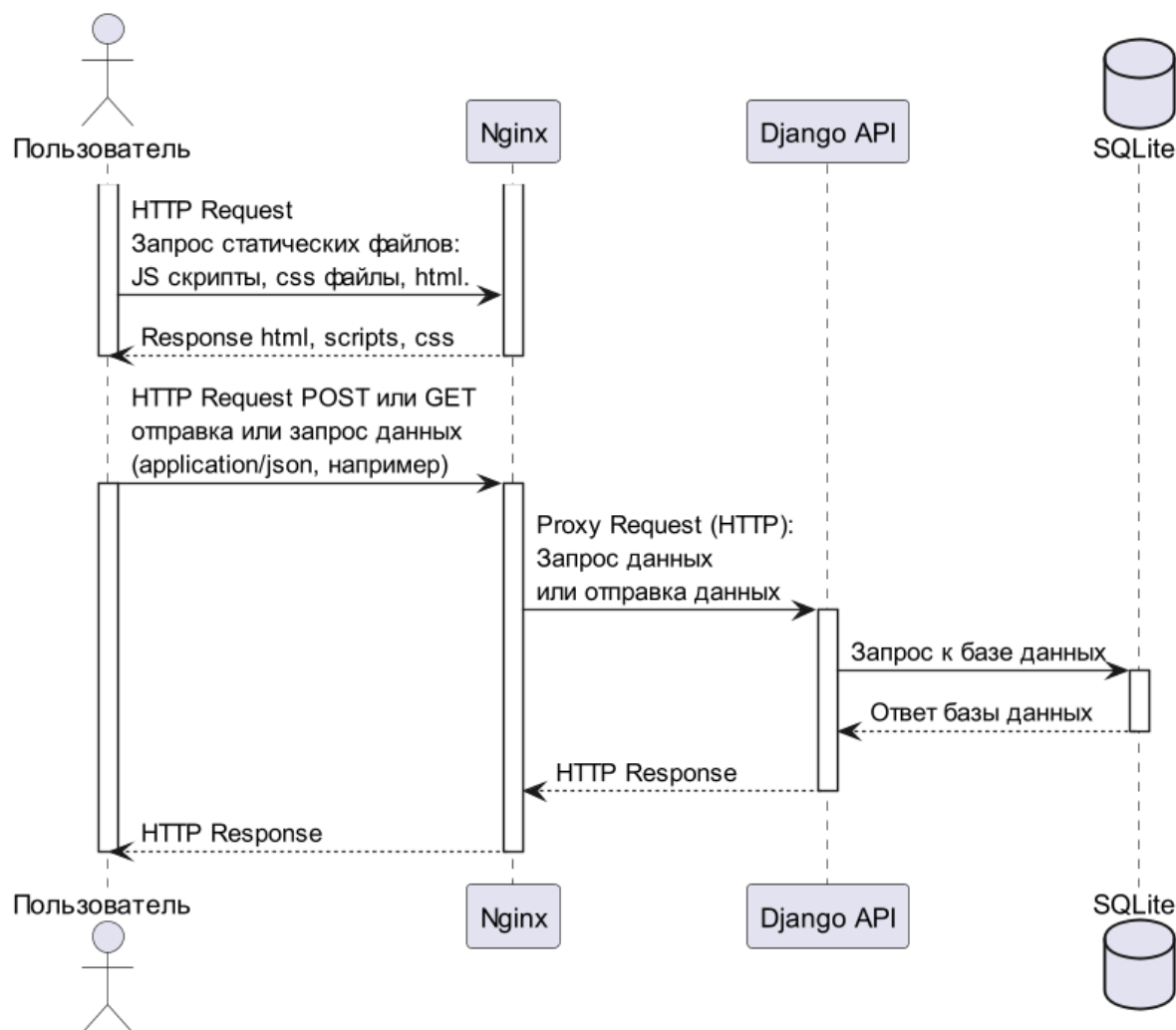


Рисунок 3 - Диаграмма последовательности обобщенная для записи к врачу по симптомам

Пользовательский интерфейс, являющийся неотъемлемой частью функциональности экспертной системы, служит основным каналом, облегчающим взаимодействие между системой и ее пользователями. Его дизайн должен отдавать приоритет интуитивности и удобству использования, обеспечивая плавную навигацию. Предлагая функциональные возможности, включающие ввод симптомов, визуализацию результатов диагностики и рекомендации по лечению, этот интерфейс является краеугольным камнем взаимодействия с пользователем. Кроме того, он может включать в себя функции доступа и изменения записей пациентов [27].

Модуль записи пациентов, занимающий центральное место в архитектуре системы, берет на себя ответственность за хранение и обработку данных пациентов. Его функциональные возможности варьируются от добавления пациентов и редактирования данных до просмотра и анализа показателей пациентов. Используя инфраструктуру базы данных, этот модуль служит хранилищем информации о пациентах и диагнозов.

После определения основных компонентов системы последующий архитектурный этап влечет за собой определение взаимодействия между ними. В рамках этой структуры база знаний взаимодействует с механизмом вывода для вынесения диагнозов на основе симптомов, сообщаемых пациентом. Результаты диагностики распространяются через пользовательский интерфейс и одновременно архивируются в модуле записи пациентов [28].

Для обеспечения операционной эффективности первостепенное значение имеют производительность и масштабируемость. В частности, в сценариях, характеризующихся большой нагрузкой пациентов и обширным хранилищем правил базы знаний, система должна демонстрировать умение обрабатывать запросы с готовностью и точностью. Стратегии, включающие оптимизацию запросов, кэширование результатов и распределенные вычисления, играют решающую роль в решении этих проблем.

Более того, архитектурные решения должны по своей сути включать положения о безопасности данных. Учитывая конфиденциальный характер медицинской информации, необходимы строгие меры для защиты конфиденциальности пациентов от несанкционированного доступа. Для этого необходимы такие механизмы, как аутентификация пользователей, протоколы авторизации и механизмы шифрования данных.

4 Обзор существующих экспертных систем для диагностики и учета пациентов

В рамках этого исследования была создана модель, направленная на диагностику заболеваний на основании симптомов, сообщаемых пациентами. Режим обучения модели включал в себя многогранную последовательность, включающую предварительную обработку данных, разработку алгоритма машинного обучения, параметризацию модели и оценку производительности.

Опираясь на набор данных, полученный с Kaggle.com [29], в диагностическую сферу модели входит широкий спектр заболеваний. Ниже представлен полный список заболеваний, которые модель способна распознать:

- грибковая инфекция,
- аллергия,
- гэрб,
- хронический холестаз,
- лекарственная реакция,
- язвенная болезнь желудка и двенадцатиперстной кишки,
- спид,
- сахарный диабет,
- гастроэнтерит,
- бронхиальная астма,
- гипертония,
- мигрень,
- шейный спондилез,
- паралич (кровоизлияние в мозг),
- желтуха,
- малярия,
- ветряная оспа,
- лихорадка денге,

- брюшной тиф [56],
- гепатит а,
- гепатит в,
- гепатит с,
- гепатит d,
- гепатит е,
- алкогольный гепатит,
- туберкулез,
- обычная простуда,
- пневмония,
- диморфные геморроидальные узлы,
- сердечный приступ,
- варикозное расширение вен,
- гипотиреоз ,
- гипертиреоз [30],
- гипогликемия,
- остеоартрит,
- артрит,
- (головокружение) параимпульсное позиционное головокружение,
- угревая сыпь,
- инфекция мочевыводящих путей,
- псориаз,
- импетиго.

Список симптомов:

- зуд,
- кожная сыпь,
- узловые образования на коже,

- постоянный похмель,
- дрожь,
- озноб,
- боль в суставах,
- боль в желудке,
- повышенная кислотность,
- язвы на языке,
- мышечная слабость,
- рвота,
- жжение на лице,
- мажущее мочеиспускание,
- усталость,
- прибавка в весе,
- беспокойство [31],
- холодные руки и ноги,
- перепады настроения,
- потеря веса,
- беспокойство,
- вялость,
- першение в горле,
- повышенный уровень сахара в крови,
- кашель,
- высокая температура,
- потемневшие глаза,
- одышка,
- потливость,
- обезвоживание,

- расстройство желудка,
- головная боль,
- желтоватая кожа [32],
- темная моча,
- тошнота,
- потеря аппетита,
- боль в глазах,
- боль в спине,
- запор,
- абдоминальная боль,
- диарея,
- легкая лихорадка,
- желтая моча,
- пожелтение глаз,
- острый недостаток жидкости [33],
- перегрузка жидкостью,
- вздутие желудка,
- опухание лимфатических узлов,
- недомогание,
- нечеткое и искаженное зрение,
- мокрота,
- першение в гортани,
- покраснение глаз,
- синусовое давление,
- насморк,
- застойные явления,
- боль в груди,

- слабость в конечностях,
- учащенное сердцебиение,
- боль во время движения мышцами,
- боль в анальной области,
- крованистый стул,
- раздражение в анусе,
- боль в шее,
- головокружение,
- судороги,
- кровоподтеки,
- ожирение,
- опухшие ноги,
- опухшие кровеносные сосуды,
- опухшие лицо и глаза,
- увеличенные щитовидные железы,
- ломкие ногти,
- опухлости экстремальные,
- чрезмерный желтыш,
- экстра-маритальные контакты,
- сухие и двигающие жимы,
- невнятная речь,
- боль в коленях [34],
- боль в тазобедренных суставах,
- мышечная слабость,
- затекшая шея,
- вздутие суставов,
- жесткость движений,

- вращательные движения,
- потеря равновесия,
- неустойчивость,
- слабость одной стороны тела,
- потеря запаха,
- дискомфорт в мочевом пузыре,
- неприятный запах мочи,
- постоянное ощущение мочи,
- выделение газов,
- внутренний зуд,
- токсический вид (брюшной тиф),
- депрессия,
- раздражительность,
- мышечные боли,
- измененная сенсорика,
- красные пятна на теле,
- боль в животе,
- ненормальная менструация,
- дисхромные пятна,
- слезотечение,
- повышенный аппетит [35],
- полиурия,
- семейная история,
- слизистая оболочка,
- ржавая оболочка,
- недостаточная концентрация,
- визуальные нарушения,

- получение крови трансфузии,
- получение стерильных инъекций,
- кома,
- желудочное кровотечение,
- вздутие живота,
- история употребления алкоголя,
- кровь в мокроте,
- выступающие желудки на икре,
- учащенное сердцебиение,
- болезненная ходьба,
- гнойные прыщи,
- угри,
- зуд,
- шелушение кожи,
- серебристая пылеватость,
- маленькие зубцы на ногтях,
- воспаляющиеся ногти,
- волдырь,
- покраснение вокруг носа,
- желтая корочка.

Вначале получены два набора данных: один представлял собой первичное хранилище, в котором содержались симптоматические профили пациентов и соответствующие диагнозы, а другой каталогизировал степень тяжести симптомов. После сбора данных был тщательно выполнен ряд мер по предварительной обработке:

- систематическое изучение структуры и атрибутов набора данных, использование таких функций, как `info()` и `head()`, для получения ценной информации,

- исправление точек данных об отсутствующих симптомах путем замены их нулевыми значениями,
- очистка набора данных от ошибочных записей и стандартизация представлений симптомов с помощью весовых показателей, извлеченных из вспомогательного набора данных,
- упрощение сегментации набора данных на подмножества обучения и оценки с помощью утилиты `train_test_split()`.

Для этапа обучения модели был выбран алгоритм машины опорных векторов (SVM), размещенный в библиотеке `scikit-learn` [36]. Обеспечение оптимальной производительности модели потребовало использования методологии поиска по сетке (`GridSearchCV`) для определения оптимальных конфигураций критических гиперпараметров, таких как `C` (параметр регуляризации) и `gamma` (коэффициент ядра).

После обучения эффективность модели была тщательно оценена с использованием набора показателей производительности, в том числе показателей `F1` и точности [37]. В дополнение к этим оценкам систематически составлялась матрица ошибок, предоставляющая графическое изображение эффективности модели по сравнению с различными классификациями заболеваний.

Важно отметить, что вышеупомянутые процессы были организованы с разумным сочетанием точности и дотошности, что подкрепляло научную строгость, присущую исследовательской деятельности.

```
from sklearn.model_selection import train_test_split,
GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid']
}
svc = SVC()
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid,
cv=5, n_jobs=-1, verbose=2)
grid_search.fit(x_train, y_train)
```

```

model = grid_search.best_estimator_
model
model = grid_search.best_estimator_
preds = model.predict(x_test)
conf_mat = confusion_matrix(y_test, preds)
df_cm = pd.DataFrame(conf_mat, index=df['Disease'].unique(),
columns=df['Disease'].unique())
print('F1-score% =', f1_score(y_test, preds,
average='macro')*100, '|', 'Accuracy% =', accuracy_score(y_test,
preds)*100)
sns.heatmap(df_cm)

```

Наконец, обученная модель была сохранена в файле с помощью библиотеки `pickle` для последующего использования в клинической практике или дальнейших исследованиях.

Этот процесс обучения модели является ключевым шагом к созданию системы поддержки принятия решений в области медицины, позволяя точно и быстро диагностировать заболевания на основе их симптомов.[38].

```

import pickle
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))

```

После использования подхода для подбора параметров была определена наилучшая модель с точностью: F1-score% = 99.48099565029264 | Accuracy% = 99.45799457994579. Параметры модели: SVC(C=0.1, gamma=1, kernel='poly').

Сильные стороны:

- эффективно оптимизируя операции и обеспечивая оптимальную сегментацию данных, модель демонстрирует быстрое выполнение, тем самым повышая операционную эффективность,
- охватывая богатый набор заболеваний и симптомов в наборе данных, модель извлекает выгоду из обширной и разнообразной выборки данных, что повышает ее диагностические возможности,
- адаптируемость модели способствует плавной интеграции новых данных, тем самым повышая ее эффективность. Модульность, присущая реализации модели, обеспечивает легкую замену, тем самым облегчая оптимизированное обновление данных.

Недостатки:

– наличие повторяющихся весов, связанных с симптомами, порождает двусмысленность, тем самым ставя под угрозу способность модели окончательно связывать симптомы с заболеваниями. тем не менее, эта характеристика наделяет модель способностью определять заболевания с той степенью точности, которая зависит от специфичности входных симптомов.

4.1 Определение и принцип работы машины опорных векторов

Support Vector Machine (SVM) представляет собой известный алгоритм машинного обучения, широко используемый как для задач классификации, так и для задач регрессии. В его основе лежит фундаментальная идея определения оптимальной гиперплоскости в пространстве признаков, плоскости, которая эффективно разделяет точки данных, принадлежащие к разным классам [38]. Эта гиперплоскость, подобная многомерному продолжению линии, образует краеугольный камень различительной способности SVM, особенно в сценариях, включающих два класса, где она проявляется как $(N-1)$ -мерная конструкция, где N представляет общее количество элементов внутри набор данных.

Основная цель SVM тяготеет к открытию гиперплоскости, которая не только минимизирует ошибки классификации, но и максимизирует пространственный запас между собой и ближайшими точками данных каждого класса, обычно называемыми опорными векторами. Эта характеристика придает SVM устойчивость к выбросам и повышает его способность к обобщению при работе с невидимыми данными.

Для решения различных сценариев классификации SVM использует две отдельные функции ядра: линейную и нелинейную. Линейное ядро приобретает актуальность, когда данные демонстрируют линейную разделимость, тем самым позволяя разграничивать классы через прямую гиперплоскость. И наоборот, в тех случаях, когда линейное разделение оказывается невозможным, в дело вступают нелинейные ядра, организуя преобразование пространства признаков, чтобы обеспечить создание дискриминационной гиперплоскости.

После обучения модель SVM превращается в мощный инструмент для классификации новых экземпляров данных. Тщательно исследуя вектор признаков неклассифицированного объекта и оценивая его близость к установленной гиперплоскости, модель эффективно относит его к соответствующему классу [39]. Однако этому мастерству классификации предшествует ключевой подготовительный этап, на котором модель SVM проходит тщательную подготовку, что подчеркивает первостепенную важность качества данных в формировании эффективности результирующей модели.

В своей элементарной сути машины опорных векторов пытаются разделить линейное пространство посредством развертывания линий, разделяющих классы. Первоначально ограниченный задачами двоичной классификации, SVM решает сложные проблемы классификации, разлагая их на ряд головоломок двоичной классификации. Комплексный режим тестирования и обучения модели SVM поясняется на рисунке 4, изображающем структурированную структуру, описывающую эти ключевые процессы.

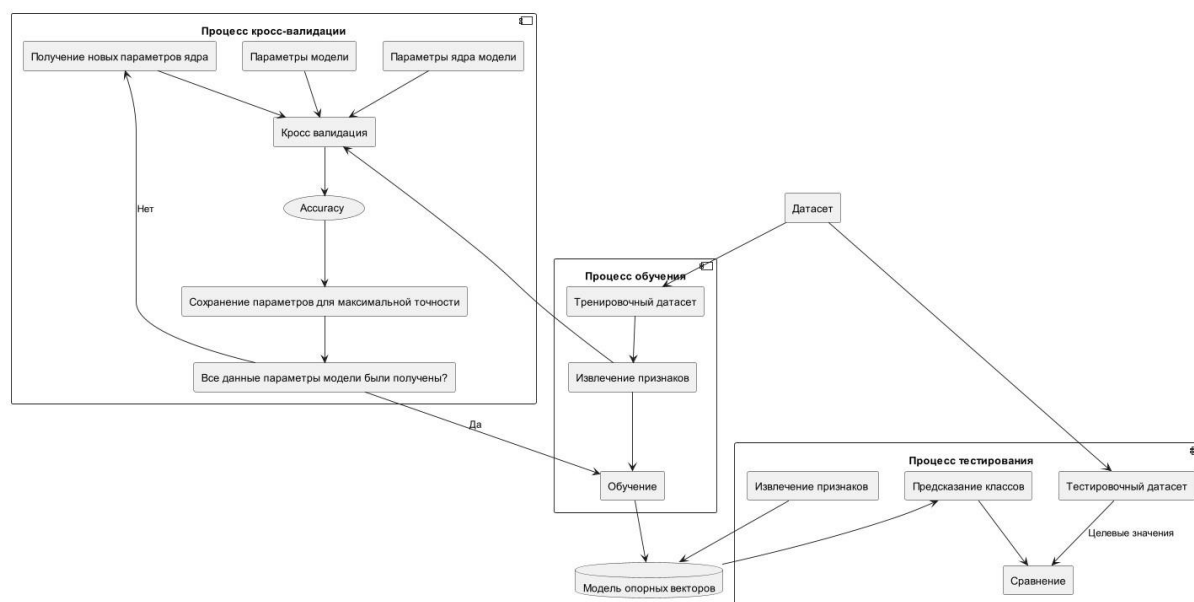


Рисунок 4 - Схема обучения и тестирования модели опорных векторов

Начальный этап влечет за собой накопление соответствующих данных, необходимых для обучения модели, что требует наличия полного ассортимента объектов наряду с явными классификационными индикаторами, обозначающими

категорию каждого объекта. Например, в контексте обучения модели, предназначенной для распознавания изображений кошек или собак, обязательным становится сборка, состоящая из таких изображений, тщательно помеченных соответствующими классами [40].

После сбора данных наступает обязательный этап предварительной обработки, включающий в себя ряд задач, направленных на обеспечение целостности и согласованности данных. Эти задачи включают, помимо прочего, выявление и устранение выбросов, которые могут исказить результаты обучения, что гарантирует их исключение из набора данных [41]. Более того, устранение пропущенных значений в наборе данных становится важнейшей прерогативой, часто достигаемой с помощью методов вменения, таких как замена среднего или медианы.

Кроме того, нормализация данных становится ключевым мероприятием, направленным на стандартизацию диапазона значений различных атрибутов в наборе данных. Уравновешивая шкалу атрибутов, смягчаются потенциальные несоответствия, возникающие из-за непропорциональных диапазонов атрибутов, тем самым создавая более единообразную и справедливую среду обучения модели. Одновременно кодирование категориальных переменных в числовые представления способствует повышению интерпретируемости модели и эффективности обработки.

Важным аспектом оценки эффективности модели является разделение данных на отдельные подмножества обучения и тестирования. В данном случае подмножество обучения берет на себя роль облегчения уточнения модели посредством итеративных процессов обучения, тогда как подмножество тестирования служит лакмусовой бумажкой, оценивая точность модели и возможности обобщения. Это разграничение между подмножествами обучения и тестирования является краеугольным камнем в строгой оценке производительности и эффективности модели.

В обычной практике данные подвергаются секционированию, обычно с соблюдением соотношения 70:30 или 80:20, что означает, что 70 или 80% набора данных выделяется для целей обучения, а оставшиеся 30 или 20% зарезервированы для тестирования [42]. Ключевой шаг на пути к оптимизации результатов и повышению эффективности модели машины опорных векторов зависит от тщательной подготовки данных, предшествующих этапу обучения.

Центральное место в архитектуре машинной модели опорных векторов занимает ядро, служащее фундаментальной составляющей, определяющей преобразование и разграничение данных в пространстве признаков. Выбор подходящего ядра зависит от характера набора данных и сложности решаемой проблемы. Существует несколько вариантов ядер, каждый из которых адаптирован к конкретным характеристикам данных и проблемным областям, что обеспечивает универсальный набор инструментов для настройки и настройки модели.

Линейные ядра, характеризующиеся своей способностью очерчивать гиперплоскость в пространстве признаков, становятся наиболее важным выбором для сценариев, характеризующихся линейно разделяемыми данными. И наоборот, полиномиальные ядра используют полиномиальные функции для преобразования данных в многомерное пространство, доказывая свою эффективность в сценариях, характеризующихся нелинейной разделимостью. Ядра RBF, использующие функции Гаусса для модуляции преобразования данных, создают круговые или эллиптические деления вокруг опорных векторов, предлагая жизнеспособное решение для наборов данных, характеризующихся нелинейностью. Между тем, сигмовидное ядро, подкрепленное сигмовидной функцией, предоставляет действенный способ решения затруднений классификации, типичными для которых являются нелинейные очертания границ [43].

После выбора ядра появляется дополнительная возможность для уточнения посредством точной настройки параметров ядра. Например, в случае

полиномиальных ядер можно настроить степень полинома, тогда как в случае ядер RBF модификация параметров влечет за собой калибровку ширины круга или эллипса. Эта тщательная калибровка ядер представляет собой основополагающий этап в траектории создания модели машины векторной поддержки, облегчая индивидуальную адаптацию к особенностям набора данных и поставленной задачи, тем самым обеспечивая улучшенные результаты производительности и повышенную точность прогнозирования.

4.2 Обучение модели машины опорных векторов

Процесс уточнения параметров на основе данных обучения характеризует попытку обучения модели машины опорных векторов (SVM). Целью здесь является построение гиперплоскости, оптимизирующей разделение между различными классами данных [44].

Приступая к обучению модели SVM, необходимо выполнить ряд методических шагов. Первоначально данные подвергаются подготовительным мерам, включая разделение на подмножества обучения и тестирования, дополненные необходимыми усилиями по расширению данных.

Затем наступает этап выбора ядра, в ходе которого происходит обсуждение выбора ядра, определяющего преобразование данных в многомерное пространство, способствующее линейному разрешению. Спектр ядер, включая линейные, полиномиальные и радиальные базисные функции (RBF), является выдающимся претендентом [45].

Вооружившись определением ядра, происходит запуск модели, знаменуя начало аналитической одиссеи, в которой модель перемещается по пространству обучающих данных, чтобы определить оптимальные классы данных, разграничивающие гиперплоскости. Кульминацией этого поиска является решение загадки оптимизации, направленной на минимизацию ошибок классификации при максимальном увеличении расстояний между классами.

После обучения начинается тигель оценки: неиспользованные тестовые образцы служат лакмусовой бумажкой для оценки эффективности модели.

Диапазон показателей, от точности до полноты и прецизионности, сходится, чтобы дать представление об эффективности модели по отношению к новым экземплярам данных.

Если производительность модели не оправдает ожиданий, следует обратиться к предшествующему этапу, на котором изменения параметров ядра или дополнительные усилия по предварительной обработке данных обещают улучшение.

Итеративный путь обучения модели машины опорных векторов подчеркивается необходимостью разумного выбора параметров и тщательного изучения данных. Тщательно обученная модель SVM становится мощным инструментом классификации и регрессии, особенно в сценариях, устойчивых к линейной дискриминации.

После завершения обучения модели машины опорных векторов с использованием обучающих данных последующий шаг требует оценки ее эффективности при классификации или регрессии новых экземпляров данных. В связи с этим становится обязательным использование тестовых данных, неиспользованных на этапе обучения.

Процесс оценки модели проходит через ряд структурированных этапов. Главным из них является обеспечение соответствия между тестовыми данными и их обучающим аналогом, при этом набор тестовых данных отражает сценарии реальных данных, тем самым согласуясь с ожидаемой рабочей средой модели [46].

Впоследствии обученная модель применяется к тестовым данным, при этом каждое наблюдение в наборе тестовых данных требует прогнозирования значений целевых переменных или меток классов.

Кульминацией прогнозирования модели является вычисление различных показателей производительности, что дает представление об эффективности модели. Общие метрики включают точность, полноту, точность и оценку F1, подходящие для затруднений классификации, в то время как задачи регрессии

требуют таких показателей, как среднеквадратическая ошибка и коэффициент детерминации (R-квадрат).

Последующие усилия влекут за собой детальный анализ вычисленных показателей производительности, предоставляющий необходимые основания для выводов об эффективности модели. Повышенная степень точности и производительности по различным показателям может предвещать хорошую производительность модели. И наоборот, случаи пониженной точности или неудовлетворительных показателей производительности могут служить предвестниками необходимости пересмотра параметров модели или методологий предварительной обработки данных.

Этапы оценки и тестирования имеют решающее значение в сфере разработки и внедрения машин опорных векторов. Эти усилия создают механизм для тщательного изучения производительности модели и обеспечения ее плавной интеграции с новыми экземплярами данных, тем самым подкрепляя эффективность и надежность модели в практических приложениях.

4.3 Применение машины опорных векторов для распознавания образов

Машина опорных векторов (SVM) становится мощным инструментом в области распознавания образов, облегчающим классификацию и дифференциацию данных на основе их внутренних атрибутов [47]. Использование SVM для распознавания образов требует прохождения структурированной траектории, полной многогранных задач.

Начальная фаза включает в себя подготовку данных, на которой разворачиваются агрегирование данных, предварительная обработка и разделение на отдельные анклавы обучения и тестирования [48]. После подготовки данных модель машины подвергается обучению с использованием машин опорных векторов на назначенном обучающем наборе. В течение этой эпохи обучения модель организует идентификацию оптимальной гиперплоскости, осуществляя разделение разрозненных классов данных.

После режима обучения наступает решающий момент оценки модели, предполагающий исчерпывающее изучение ее производительности по сравнению с тестовым образцом. Эта оценочная попытка позволяет получить представление о точности модели, а также о вспомогательных показателях, а также оценить ее эффективность в задачах распознавания изображений.

После успешного завершения этапа оценки модель готова к использованию в целях распознавания образов, касающихся новых данных. Здесь модель умело распознает и классифицирует новые изображения на основе их существенных атрибутов, тем самым распознавая их категориальную принадлежность.

Использование машин опорных векторов для распознавания образов выходит за рамки дисциплинарных границ, находя применение в различных областях, включая компьютерное зрение, биомедицину и финансы, среди других [49]. Его универсальность распространяется на приложения, включающие распознавание лиц, обнаружение объектов и классификацию медицинских изображений, что подчеркивает его значимость в различных областях задач.

Однако по-прежнему необходимо подчеркнуть, что эффективная реализация машин опорных векторов для распознавания образов требует тщательного соблюдения структурированного рабочего процесса, включающего подготовку данных, выбор ядра, настройку, а также исчерпывающую оценку и тестирование модели. Этот целостный подход является неременным условием достижения высоких стандартов точности и надежности, необходимых в области распознавания образов.

Машина опорных векторов (SVM) имеет множество преимуществ: от ее эффективности как одного из наиболее оптимизированных алгоритмов машинного обучения до ее универсальности при решении различных проблемных областей, включая классификацию, регрессию и обнаружение аномалий [58]. Более того, SVM демонстрирует похвальную способность адаптироваться к различным типам данных, включая числовые, категориальные и текстовые данные, что подчеркивает присущую ему гибкость.

Неотъемлемая сила SVM заключается в ее способности обрабатывать нелинейные данные, чему способствует использование различных ядер. Эта функция позволяет создавать сложные границы принятия решений, тем самым способствуя повышению точности классификации [50].

Более того, SVM демонстрирует устойчивость перед лицом выбросов и экстремальных значений данных, стремясь определить оптимальную интервальную гиперплоскость, которая максимизирует ее расстояние от ближайших точек каждого класса. Такая надежность повышает его надежность и эффективность при работе с различными наборами данных.

Несмотря на свои многочисленные преимущества, SVM имеет определенные ограничения, заслуживающие внимания. Главным из них является его склонность к высокой вычислительной сложности, особенно в сценариях, характеризующихся множеством функций или объемными наборами данных. Необходимость выбора параметров еще больше усугубляет эту сложность, поскольку неточности могут вызвать колебания размера и производительности модели [58].

Более того, эффективность SVM зависит от тщательной предварительной обработки данных, включая масштабирование параметров и устранение выбросов. Пренебрежение этими подготовительными мерами может подрывать точность модели, что потребует скрупулезного внимания к качеству данных.

Более того, SVM может проявлять неэффективность при столкновении с наборами данных, содержащими обширный набор категорий, что потенциально может потребовать обращения к альтернативным методологиям машинного обучения, лучше подходящим для таких сценариев в таблице 1 описаны свойства машины опорных векторов.

Таблица 1 — Свойства машины опорных векторов.

Свойство	Описание
Определение	Машина опорных векторов (SVM) – это алгоритм машинного обучения, который используется для классификации и регрессии. Он основан на идее поиска оптимальной гиперплоскости, которая разделяет данные разных классов.[58, С.147]
Принцип работы	SVM стремится найти гиперплоскость, которая максимально разделяет данные разных классов. Он ищет такую гиперплоскость, которая имеет наибольшее расстояние до ближайших точек каждого класса, называемых опорными векторами.
Подготовка данных для обучения	Данные для обучения SVM должны быть числовыми и нормализованными. Категориальные переменные могут быть преобразованы в числовые значения. Также важно удалить выбросы и обработать пропущенные значения.
Выбор и настройка ядра	Ядро SVM определяет способ преобразования данных в более высокую размерность, где они могут быть линейно разделимыми. Различные ядра, такие как линейное, полиномиальное и радиальное базисное функциональное ядро (RBF), могут быть выбраны в зависимости от типа данных и задачи.
Обучение модели SVM	Обучение модели SVM включает в себя настройку параметров ядра и оптимизацию гиперплоскости. Это может быть выполнено с использованием алгоритма оптимизации, такого как Sequential Minimal Optimization (SMO).
Оценка и тестирование модели	Модель SVM может быть оценена с использованием различных метрик, таких как точность, полнота и F-мера. Она также может быть протестирована на новых наборах данных для проверки ее обобщающей способности.[58, С.137]
Применение для распознавания образов	SVM может быть использован для распознавания образов, таких как распознавание лиц, рукописного текста или обнаружение объектов на изображениях. Он может быть эффективным в задачах, где данные имеют сложную структуру и требуют высокой точности классификации.

Продолжение Таблицы 1.

Свойство	Описание
Преимущества и ограничения	SVM имеет преимущества, такие как хорошую обобщающую способность, эффективность в пространствах высокой размерности и возможность работы с различными типами данных. Однако он также имеет ограничения, такие как сложность выбора и настройки ядра, чувствительность к шуму и большое время обучения на больших наборах данных.

4.4 Матрица ошибок

Матрица ошибок или матрица неточностей, как показано на рисунке 5, была сформулирована для визуального отображения точности модели. Как показано на рисунке 6, снимке, полученном из блокнота Jupyter, показаны расчеты точности и показателя F1, демонстрирующие значения, близкие к 100%. Такие результаты предполагают чрезвычайно повышенную точность модели, о чем свидетельствуют ссылки [51].

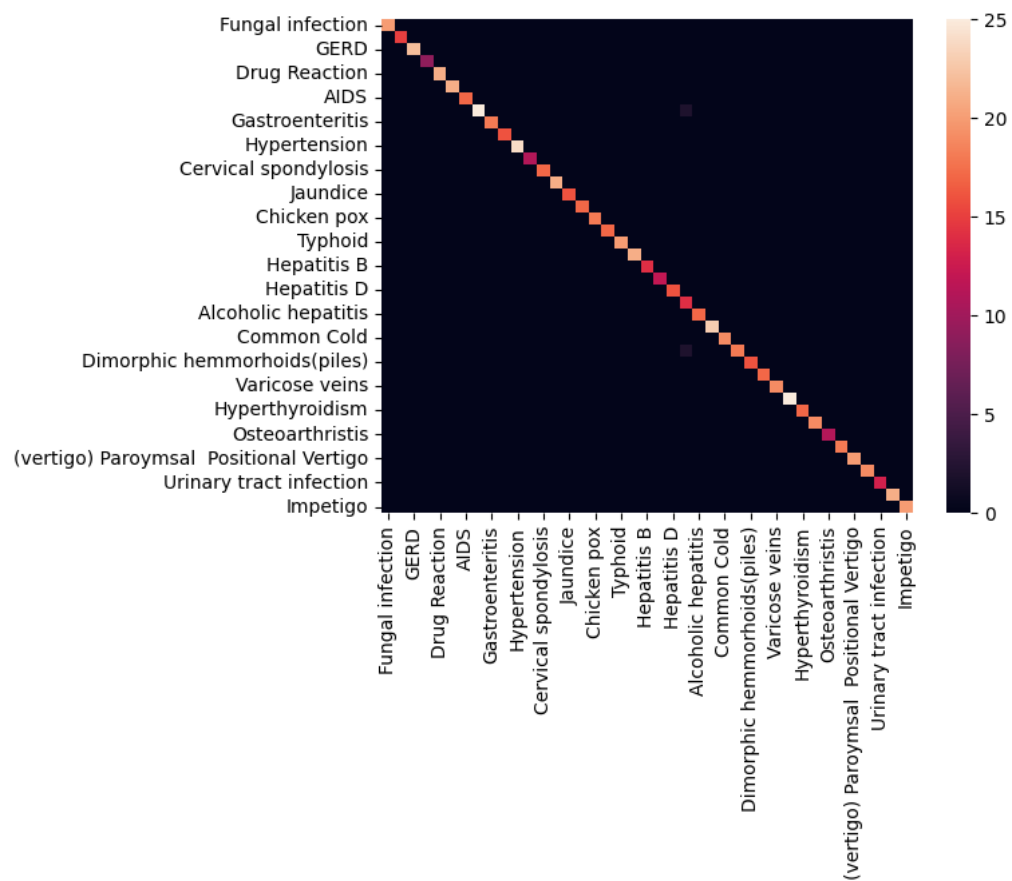


Рисунок 5 - Матрица ошибок (Confusion matrix)

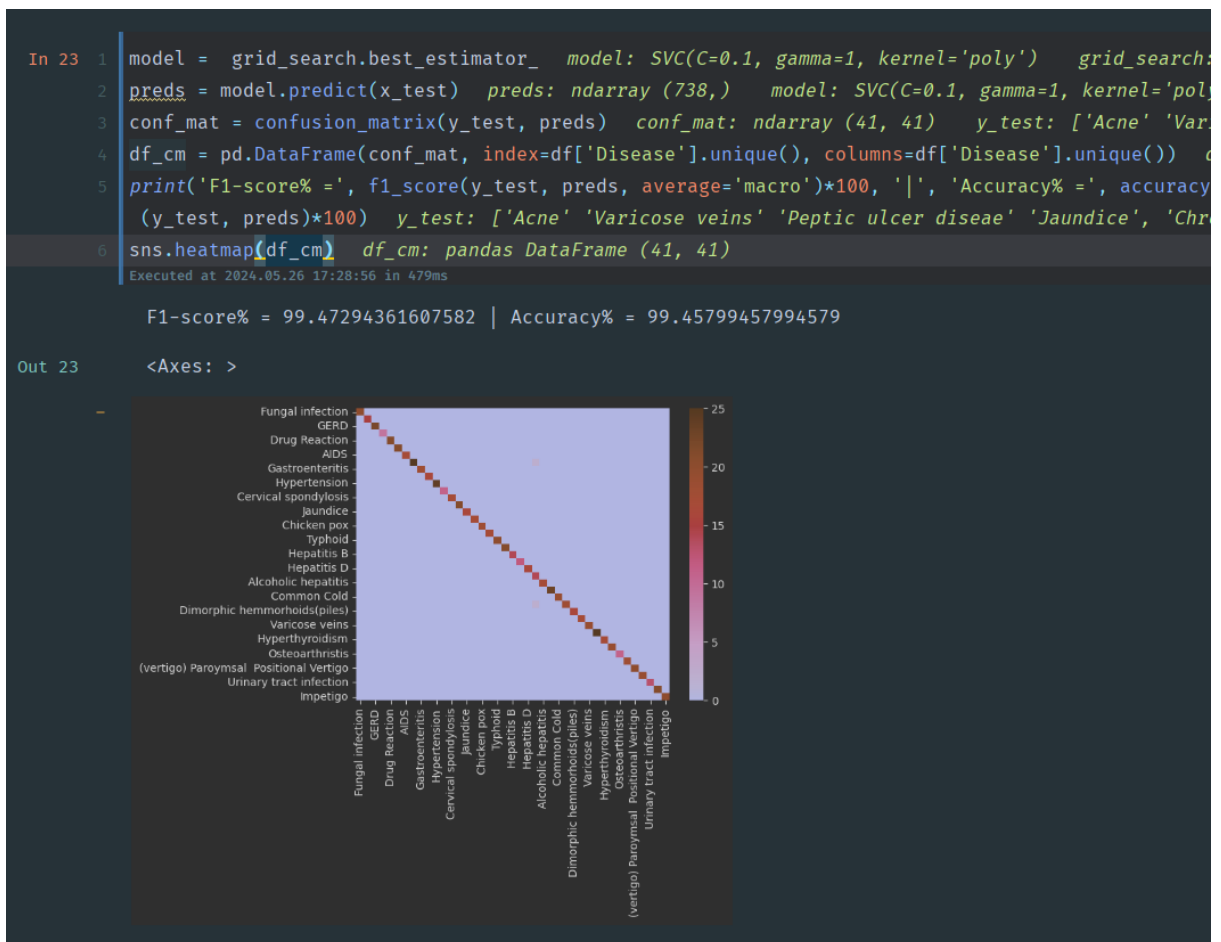


Рисунок 6 Матрица ошибок, F1-score и ассураку

5 Реализация

В рамках данной магистерской диссертации работы реализовано комплексное веб-приложение, состоящее из двух частей. Первая часть является клиентской и реализована с помощью фреймворка React на языке программирования JavaScript. Вторая часть является серверной и используется фреймворк Django в паре с фреймворком Django REST framework для реализации API на языке программирования Python [52].

Веб-приложение способно классифицировать заболевания по предоставленным симптомам, на рисунке 7 изображена структура взаимодействия пользователей с базой данных.

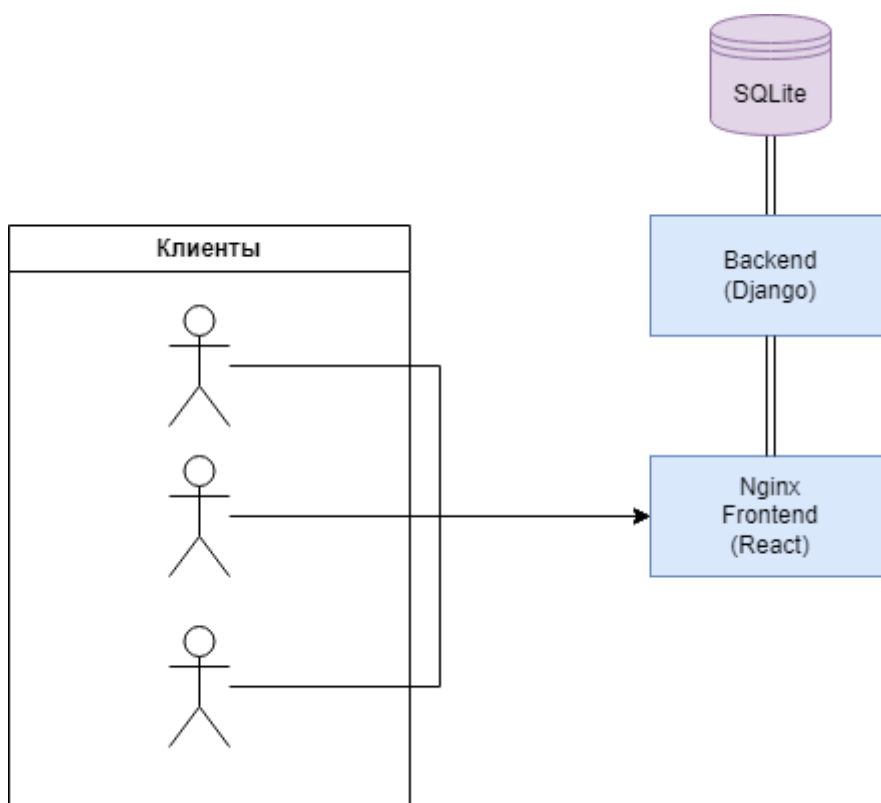


Рисунок 7 - Структура взаимодействия пользователей с базой данных.

Для реализации на стороне клиента, где доступ к пользовательскому интерфейсу осуществляется напрямую через веб-браузеры, основным языком программирования служит JavaScript, широко распространенный язык для браузерных сред. Углубление в архитектурную структуру верхнего эшелона приложения требует первоначального изучения взаимодействия пользователя с

веб-сервером nginx. Этот сервер действует как прокси-сервер запросов, так и как распространитель статических файлов для пользователей. Более того, nginx устанавливает взаимосвязь между фронтенд- и бэкенд-компонентами приложения. Примечательно, что в этой конфигурации nginx действует как обратный прокси-сервер, обеспечивая бесперебойную связь между различными элементами. Как показано на рисунке 8, комплексная система состоит из отдельных компонентов, а именно: серверной части Django или API Django, составляющей сегмент API приложения, базы данных (БД) и Nginx, функционирующего как обратный прокси-сервер.

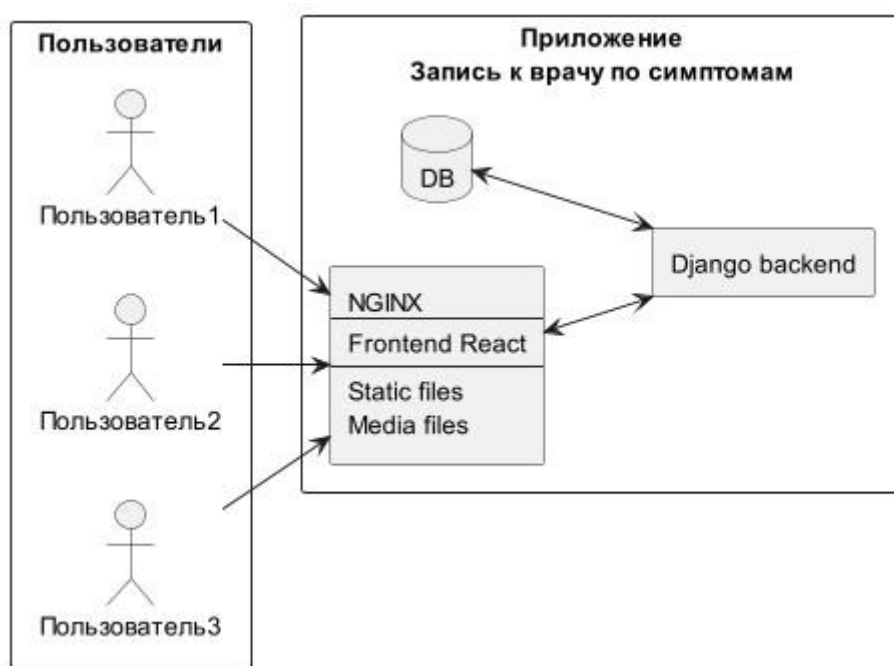


Рисунок 8 Структура приложения на верхнем уровне

Хотя создание клиентской части целиком с использованием чистого JavaScript + HTML + CSS вполне возможно, использование различных платформ предпочтительнее для ускорения разработки веб-приложений. В рамках этого проекта фреймворк React оказался предпочтительным выбором. Обеспечивая сборку интерфейсов из отдельных компонентов, разработанных независимо, React облегчает создание сложных интерфейсов и повторно используемых компонентов, тем самым повышая эффективность проекта в различных сферах [53].

Для передачи данных между фронтендом и бэкендом достаточно встроенного метода `fetch` в JavaScript; однако в настоящее время `Axios` является преобладающим методом из-за его расширенной функциональности и повышенной гибкости. С точки зрения серверной реализации особое место занимает язык программирования Python версии 3.12.1. Использование известной версии 5-й платформы Django для серверной разработки представляет собой стратегический выбор. Django, почитаемый за свою универсальность, считается одной из ведущих платформ для создания веб-сайтов, веб-приложений и платформ электронной коммерции. Основанный на шаблоне MVC (Model View Controller), Django предлагает изначально структурированную структуру, способствующую оптимизации процессов разработки, как указано в [54]. На рисунке 9 изображена структура базы данных.

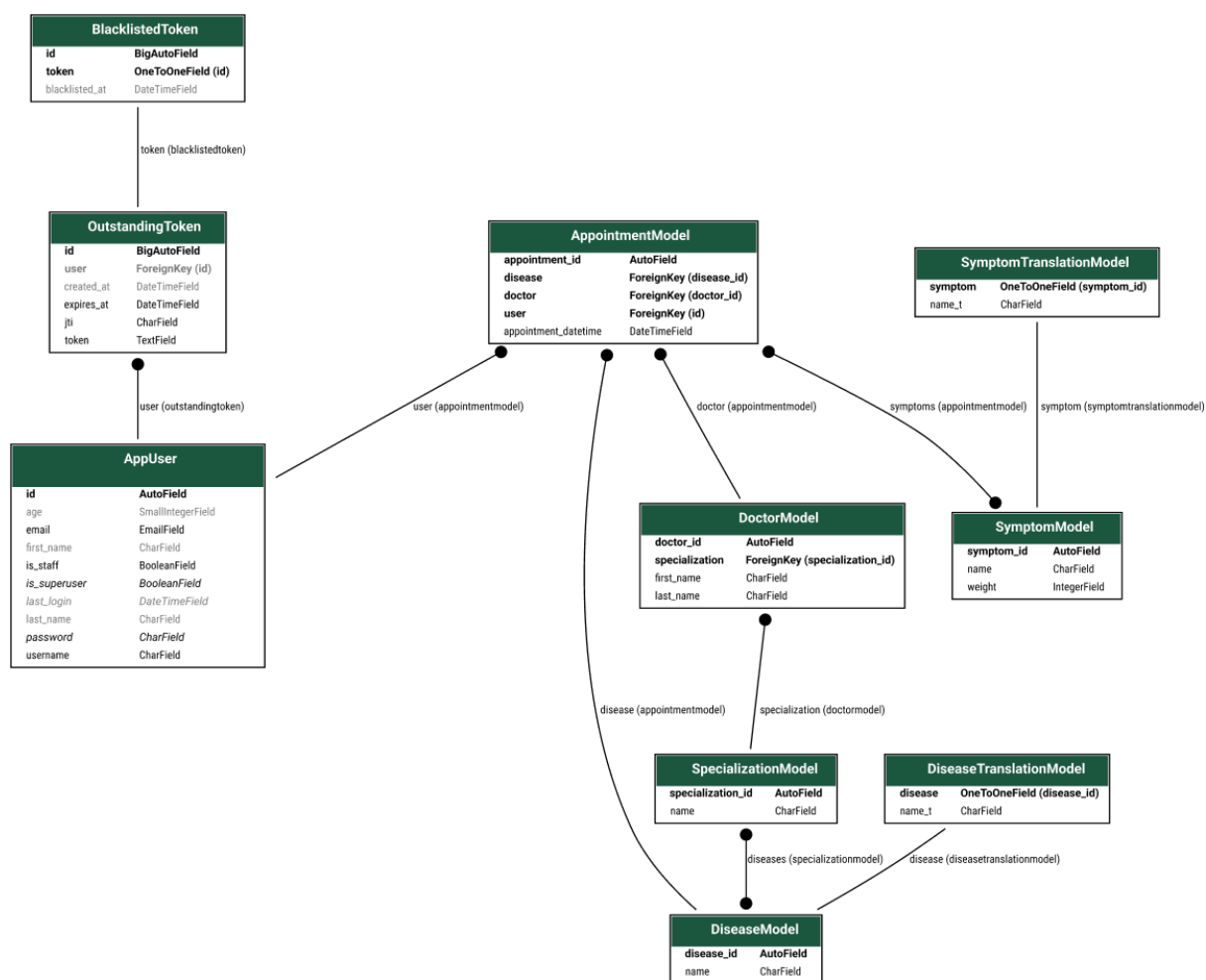


Рисунок 9 - Структура базы данных

В контексте шаблона Модель-Представление-Контроллер (MVC) Модели представляют модели базы данных, связанные с таблицами в базе данных, что позволяет взаимодействовать с этими таблицами как с объектами Python. Контроллер, с другой стороны, управляет потоком представления или обновляет данные модели. Представления, в свою очередь, представляют пользовательский интерфейс (UI) и передают информацию пользователям через веб-браузеры или интерфейсы приложений.

Для обеспечения бесперебойной связи между компонентами Frontend и Backend было сочтено необходимым внедрение RESTful API для обработки входящих и исходящих данных, отметил В.В. Лобанов [5555555555]. Этот подход потребовал принятия платформы Django REST (DRF), мощной среды для создания приложений REST в экосистеме Django. Кроме того, для удобства была включена документация Swagger с OpenAPI.

Примечательно, что приложения REST придерживаются концепции без сохранения состояния, то есть каждый запрос должен содержать достаточно данных для получения или передачи необходимой информации клиенту. Кроме того, разработанное приложение требует аутентификации, позволяя пользователям входить в систему и получать доступ к записям о своих встречах. Для этого была использована библиотека `djangorestframework-simplejwt`, предоставляющая возможности аутентификации на основе JWT. Этот механизм аутентификации включает в себя генерацию токена доступа и токена обновления в ответ на данные авторизации пользователя. При регистрации пользователи указывают свой адрес электронной почты и пароль, которые служат данными аутентификации.

5.1 Сборка проекта и nginx

Аспект, относящийся к языку программирования Python, наряду с другими интерпретируемыми языками, заключается в необходимости наличия интерпретатора для выполнения его операций. Это требование требует обслуживания всей среды, включая виртуальную среду и библиотеки, как на

сервере для окончательного вывода, так и на этапе разработки. Этот сценарий оказывается далеко не идеальным для развертывания и поддержки исходного кода. Развертывание проектов, разработанных на Python или использующих Python вместе с другими технологиями, часто предполагает использование Docker [56].

Docker предлагает уровень стандартизации среды, оптимизируя развертывание и масштабируемость приложений, одновременно способствуя сотрудничеству между разработчиками. Платформа работает по принципу контейнеризации, который изолирует приложения друг от друга и от базовой системы. Такая изоляция способствует повышению безопасности и надежности компонентов программного обеспечения.

Развертывание каждого контейнера по отдельности может по-прежнему вызывать проблемы, поэтому docker-compose часто используется для управления приложениями, состоящими из нескольких контейнеров. Docker Compose представляет собой утилиту, позволяющую специфицировать и выполнять многоконтейнерные приложения с помощью файлов конфигурации в формате YAML. Используя Docker Compose, можно наметить архитектуру приложения, разграничивая контейнеры, их взаимозависимости, конфигурации сети, объемы данных и различные другие параметры [57].

В работе реализован следующий конфиг:

```
services:
  django:
    build: ./backend
    command: >
    sh -c " poetry run gunicorn core.wsgi:application --bind
    0.0.0.0:8000"
    expose:
      - 8000
    volumes:
      - ./backend:/app/
    nginx:
    build: ./nginx
    ports:
      - 80:80
    depends_on:
```

```
- django
volumes:
- ./backend/staticfiles:/home/app/web/staticfiles/back
- ./frontend/dist:/home/app/web/staticfiles/front
```

В такой конфигурации всего создано два контейнера. Первый контейнер `django` отвечает за работу `api` части или `backend` части веб-приложения.[61] Основная логика установки пакетов определяется в файле `./backend/Docker` для этого приложения. Содержимое этого файла представлено ниже:

```
FROM python:3.12.2

WORKDIR /app

COPY pyproject.toml ./

RUN python -m pip install poetry

RUN pip install poetry && poetry install --only main --no-root -
-no-directory

RUN apt update
RUN apt install gunicorn3 --yes
```

Конфигурационный файл `Docker` определяет команды, которые позволяют установить все необходимые зависимости.

Также в сервисах создан сервис `nginx`, `Docker` файл, которого представлен ниже:

```
FROM nginx:1.25

RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d
```

Здесь происходит только удаление стандартного конфига `nginx` и копирование своего конфигурационного файла внутрь контейнера.

Для работы также дополнительно используется `nginx`-сервер. `Nginx` – это веб-сервер и прокси-сервер, который изначально был разработан для обработки большой нагрузки и обеспечения масштабируемости[61]. Он широко используется для обслуживания веб-сайтов, обратного прокси, балансировки нагрузки, обработки запросов `API` и других задач, связанных с обработкой `HTTP`- и `HTTPS`-трафика.

Конфигурационный файл `Nginx` для данного проекта представлен ниже.

```

upstream api {
server django:8000;
}

server {
listen 80;

location /api/ {
proxy_pass http://api;
}

location /admin/ {
proxy_pass http://api;
}

location /staticfiles/ {
alias /home/app/web/staticfiles/back/;
}

location / {
alias /home/app/web/staticfiles/front/;

try_files $uri $uri/ /index.html;
}

}

```

Upstream api определяет группу серверов, которые будут обрабатывать запросы для проксирования. Определен только один сервер Django, доступный по адресу django:8000. Это имя django соответствует имени контейнера или IP-адресу сервера, где запущено Django приложение.[58]

Location /api/ определяет правило маршрутизации для URI, начинающихся с /api/. Все такие запросы будут перенаправлены на сервер, определенный в блоке upstream api.

Location /admin/: Аналогично предыдущему, этот блок определяет маршрутизацию для URI, начинающихся с /admin/, и также перенаправляет их на сервер, определенный в блоке upstream api.

Location /staticfiles/: Этот блок определяет правило для обработки запросов к статическим файлам. Запросы к URI, начинающимся с /staticfiles/, будут обслуживаться из локальной файловой системы по указанному пути /home/app/web/staticfiles/back/.

Location `/`: Этот блок является обработчиком по умолчанию для всех остальных запросов. В случае, если URI не соответствует ни одному из предыдущих правил (`/api/`, `/admin/`, `/staticfiles/`), запрос будет перенаправлен на файл `index.html` в директории `/home/app/web/staticfiles/front/`.

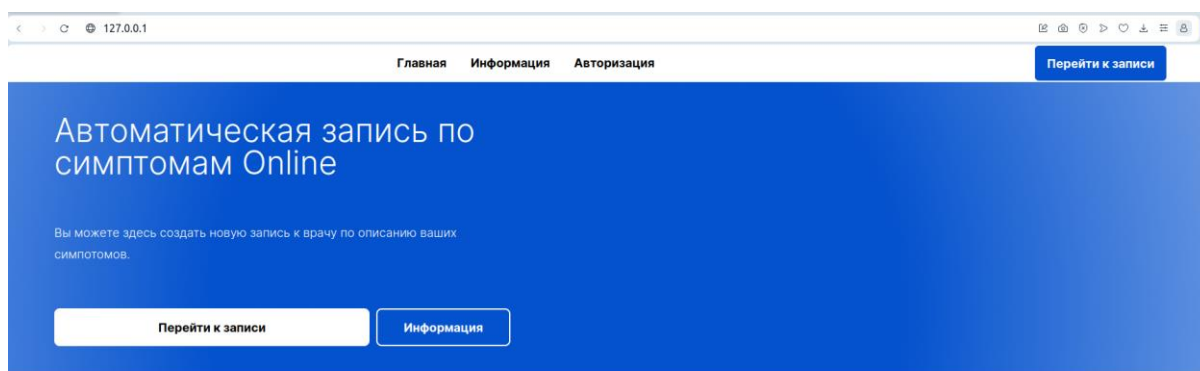
6 Запуск и тестирование приложения

Для запуска приложения необходимо установить Docker Desktop: Docker Desktop: The #1 Containerization Tool for Developers | Docker.[59]

После этого нужно его запустить, чтобы отобразилось окно Docker.

Далее нужно открыть терминал так, чтобы по команде ls или dir отображался файл docker-compose.yml

Для запуска всего проекта достаточно выполнить docker-compose up



Как это работает?

Все, что нужно, чтобы вы создали свой аккаунт, а затем записались к врачу, введя Ваши симптомы

Запись к врачу работает полностью в автоматическом режиме благодаря использованию модели машинного обучения SVC. Подробнее можно изучить на [странице информации](#).

Рисунок 10 - Главное окно сайта.

Перед тем как начать вводить симптомы заболевания пользователю необходимо зарегистрироваться, на рисунках 10 – 23 изображен полный цикл работы с приложением.

127.0.0.1/auth

Главная Информация Авторизация

Перейти к записи

Авторизация

Email

Введите Email...

Пароль

Введите пароль...

Авторизоваться

[Еще нет аккаунта? Зарегистрироваться](#)

Рисунки 11 - Окно авторизации пользователя

127.0.0.1/register

Главная Информация Авторизация

Перейти к записи

Регистрация

Email

Введите Email...

Имя пользователя

Введите имя пользователя...

Пароль

Введите пароль...

Повторите пароль

Пароли должны совпадать

Зарегистрировать

[Уже зарегистрированы? Войти](#)

Рисунки 12 - Окно регистрации нового пользователя

The screenshot shows a web browser window with the address bar displaying "127.0.0.1/register". The page has a blue header with the word "Регистрация" (Registration) on the left and a "Главная" (Home) link on the right. A white confirmation box in the center reads: "Подтвердите действие на 127.0.0.1", "Зарегистрирован", "username: konstitycii", "email: konstitycii@mail.ru", with an "OK" button. Below this, a registration form contains fields for "Email" (filled with "konstitycii@mail.ru"), "Имя пользователя" (filled with "konstitycii"), "Пароль" (masked with dots), and "Повторите пароль" (masked with dots). A blue "Зарегистрировать" (Register) button is at the bottom of the form. A link "Уже зарегистрированы? Войти" (Already registered? Log in) is located below the button.

Рисунок 13 - Регистрация нового пользователя успешно выполнена

The screenshot shows a web browser window with the address bar displaying "127.0.0.1/profile". The page has a blue header with the text "Пользователь: konstitycii" (User: konstitycii) on the left and navigation links "Главная" (Home), "Информация" (Information), "Профиль" (Profile), and "Выход" (Logout) on the right. A "Перейти к записи" (Go to record) button is also present. The main content area is titled "Данные аккаунта" (Account data) and "Личные данные" (Personal data). It contains a form with fields for "Имя" (Name, filled with "Андрей"), "Фамилия" (Surname, filled with "Пронькин"), and "Возраст (лет)" (Age in years, filled with "23"). A blue "Изменить" (Change) button is at the bottom of the form. A blue sidebar on the left contains the links "Данные профиля" (Profile data) and "Записи" (Records).

Рисунок 14 - Личные данные аккаунта

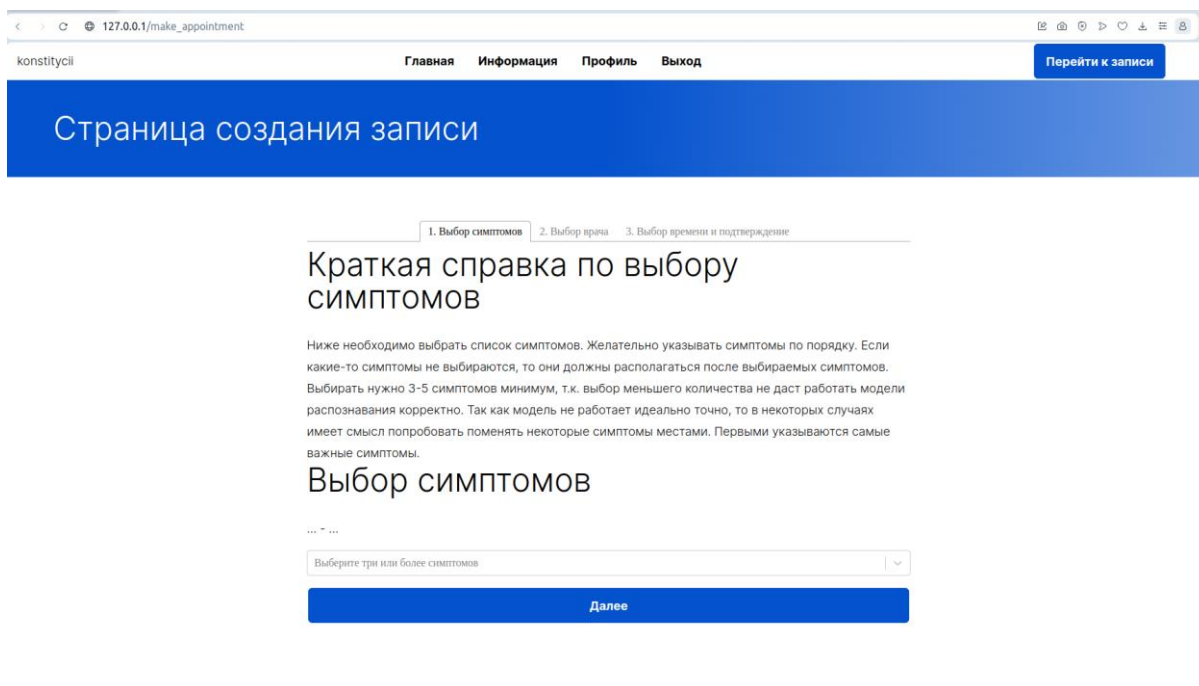


Рисунок 15 - Выбор симптомов болезни

Необходимо выбрать минимум 3 симптома болезни, для постановки предварительного диагноза и выбора специалиста.

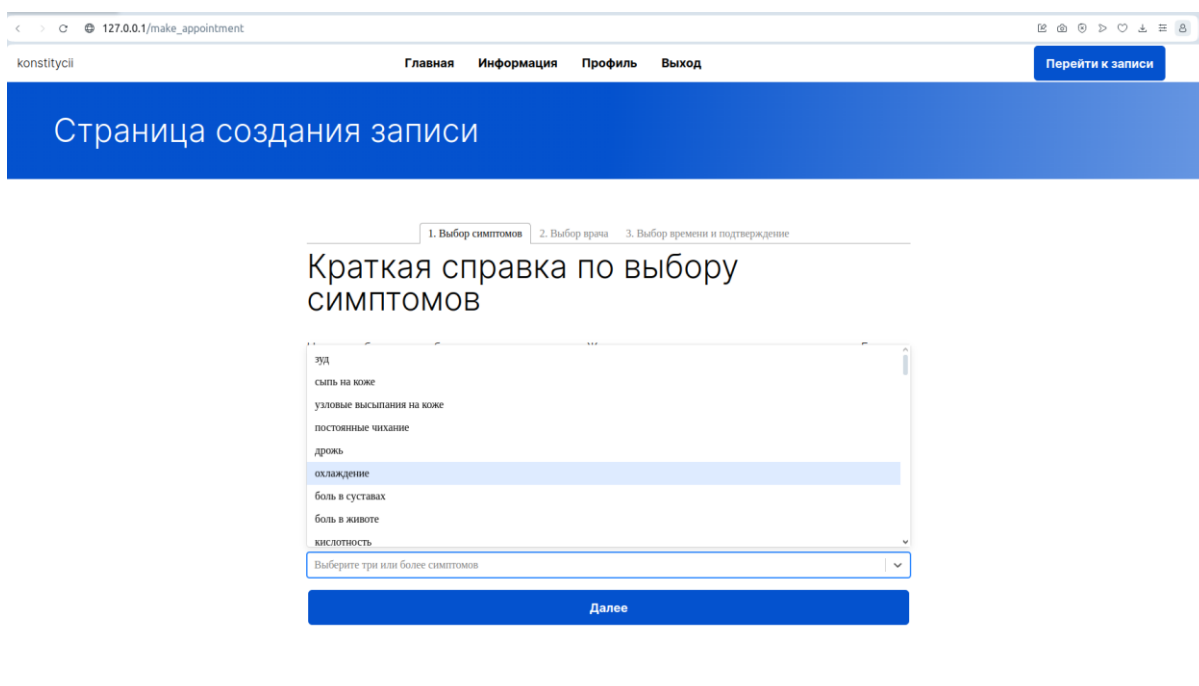


Рисунок 16 - Выбор симптомов

Я выбрал 3 симптома.

The screenshot shows a web browser at the URL 127.0.0.1/make_appointment. The page title is 'Страница создания записи' (Appointment Creation Page). The navigation bar includes 'Главная', 'Информация', 'Профиль', and 'Выход', along with a 'Перейти к записи' button. The main content area has a progress bar with three steps: '1. Выбор симптомов' (selected), '2. Выбор врача', and '3. Выбор времени и подтверждение'. Below the progress bar is the heading 'Краткая справка по выбору симптомов' (Brief guide to symptom selection) followed by explanatory text. The section 'Выбор симптомов' (Symptom Selection) shows 'Allergy - аллергия' as the selected condition. Below this, three tags are visible: 'потеря аппетита', 'потеря веса', and 'потеря обоняния'. A blue 'Далее' (Next) button is at the bottom.

Рисунок 17 - Выбор симптомов болезни

Система поставила мне предварительный диагноз - Аллергия. На следующей странице она предлагает запись к аллергологу.

The screenshot shows the same website at the same URL, but at the '2. Выбор врача' (Select Doctor) step. The progress bar now highlights '2. Выбор врача'. The heading is 'Выбор специализации врача' (Select Doctor Specialization). A dropdown menu labeled 'Выбор специализации' shows 'Аллерголог' (Allergologist) as the selected option. A blue 'Далее' (Next) button is at the bottom.

Рисунок 18 - Выбор специализации

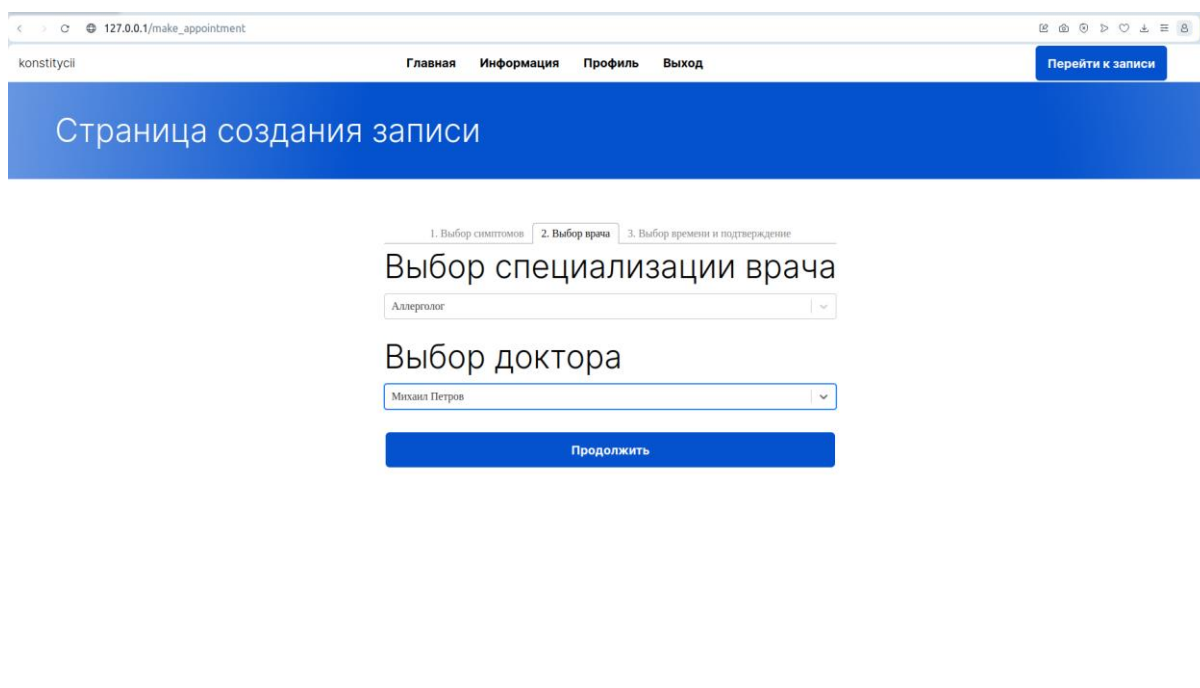


Рисунок 19 - Выбор врача

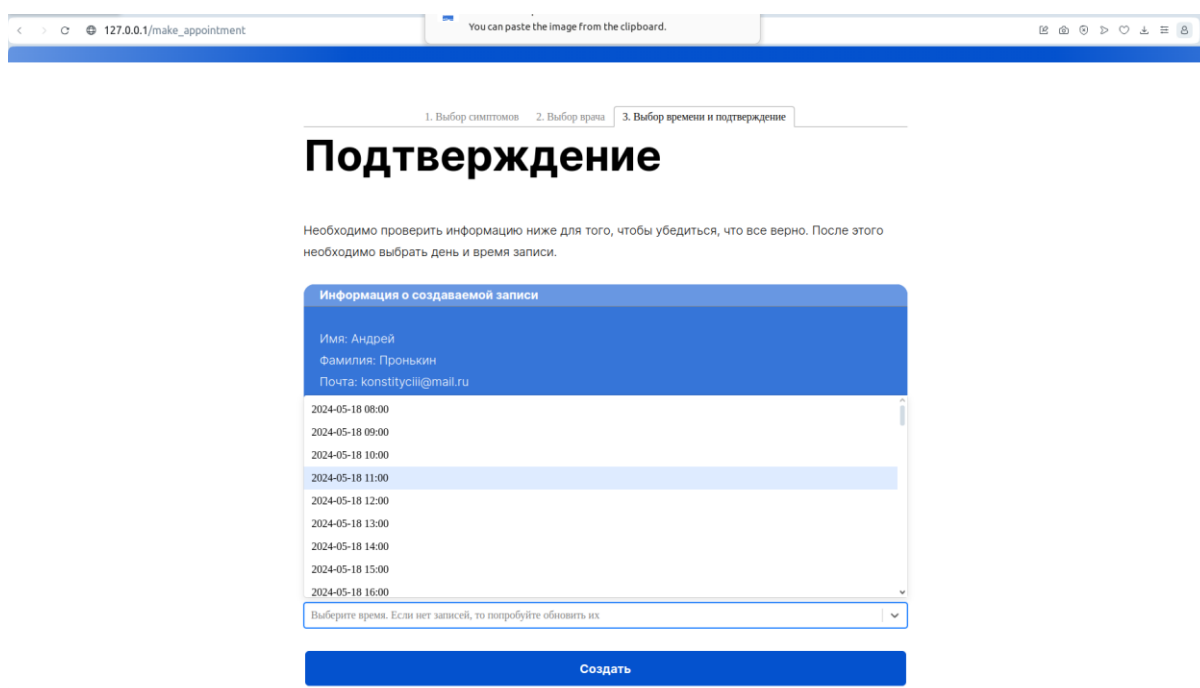


Рисунок 20 - Выбор времени записи

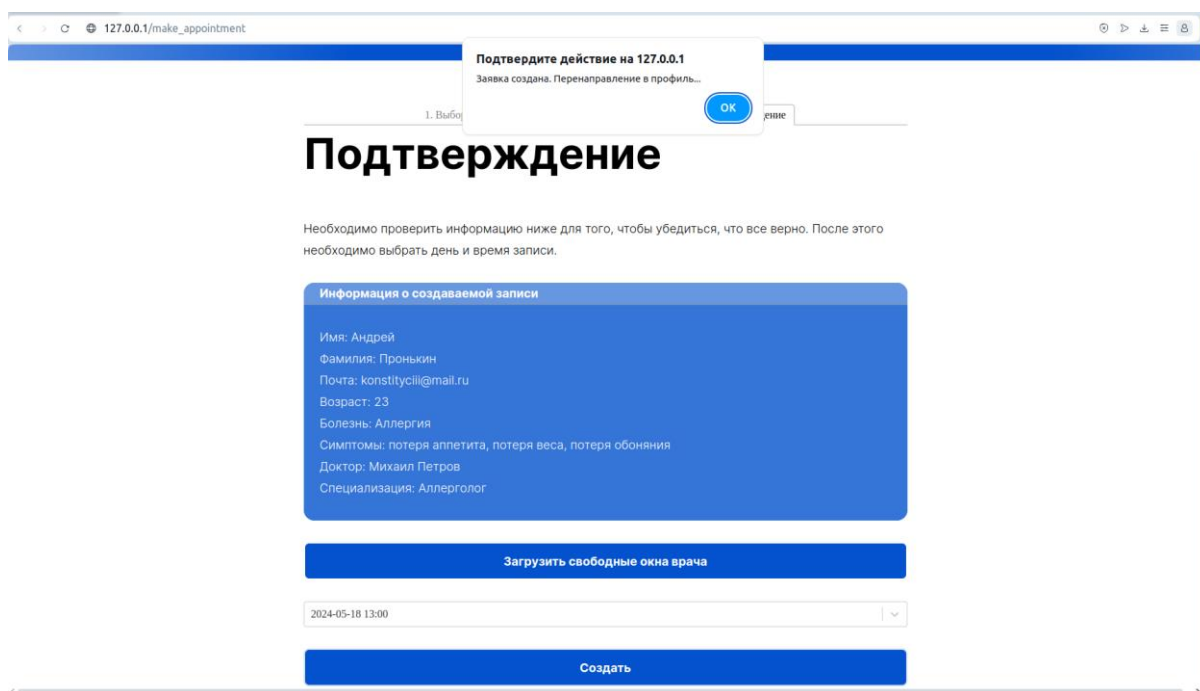


Рисунок 21 - Успешная запись к врачу

Сервис успешно записал к врачу. Дальше я ввел симптомы: боль в животе, потеря аппетита, усталость, высокая температура.

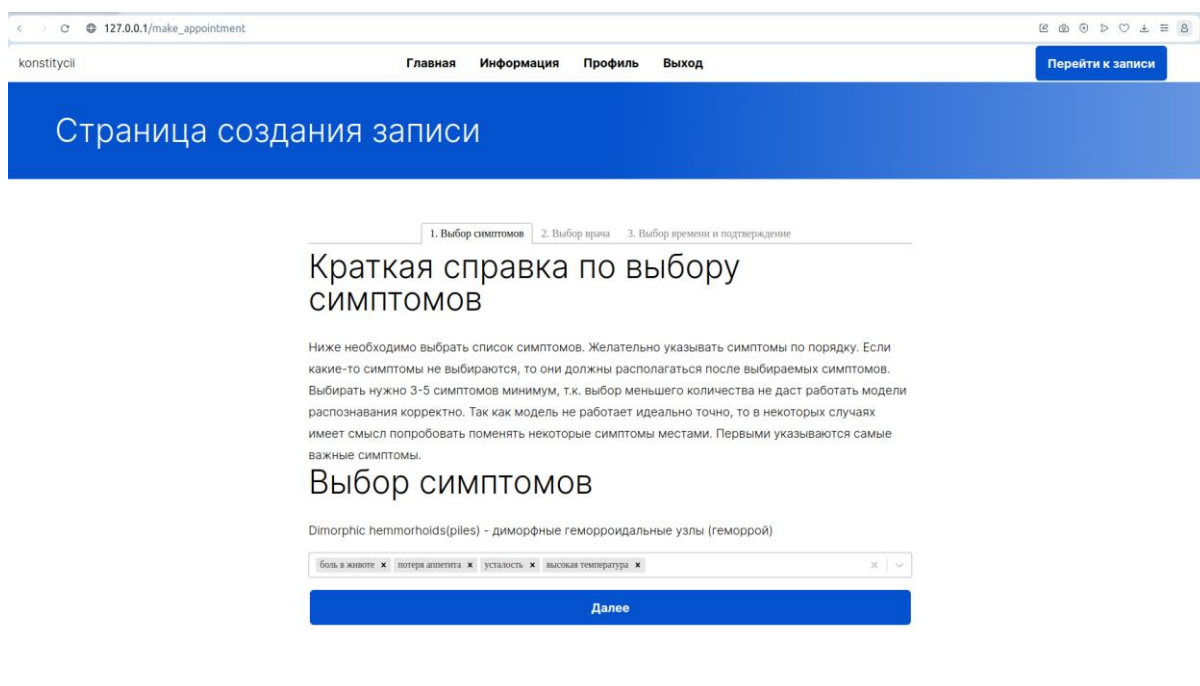


Рисунок 22 - Ввод симптомов болезни

Сервис поставил мне предварительный диагноз - геморрой, что является верным.

Следующий тестовый набор симптомов я ввел следующий: высокая температура, дрожь, язвы на языке, пятна в горле, усталость.

konstitucii Главная Информация Профиль Выход [Перейти к записи](#)

Страница создания записи

1. Выбор симптомов 2. Выбор врача 3. Выбор времени и подтверждение

Краткая справка по выбору симптомов

Ниже необходимо выбрать список симптомов. Желательно указывать симптомы по порядку. Если какие-то симптомы не выбираются, то они должны располагаться после выбираемых симптомов. Выбирать нужно 3-5 симптомов минимум, т.к. выбор меньшего количества не даст работать модели распознавания корректно. Так как модель не работает идеально точно, то в некоторых случаях имеет смысл попробовать поменять некоторые симптомы местами. Первыми указываются самые важные симптомы.

Выбор симптомов

Peptic ulcer disease - язвенная болезнь желудка

высокая температура x дрожь x язвы на языке x усталость x пятна в горле x

[Далее](#)

Рисунок 23 - Ввод симптомов болезни

Сервис поставил мне предварительный диагноз - язвенная болезнь желудка, что является верным.

ЗАКЛЮЧЕНИЕ

Система включает в себя следующие модули:

- модуль ввода данных и описания симптомов болезни,
- модуль распознавания симптомов,
- модуль классификации вероятных симптомов по заболеванию,
- модуль предварительного диагностирования с определением вероятного заболевания,
- модуль определения требуемых врачей,
- модуль рекомендательного выбора конкретного специалиста,
- модуль записи к врачу,
- модуль формирования медицинской карты пациента с перечнем симптомов, вероятных заболеваний и рекомендованных врачей,
- база данных пациентов,
- модуль управления базой данных.

Каждый из этих модулей был разработан с использованием современных технологий, обеспечивающих высокую точность и скорость работы. Особое внимание было уделено удобству использования системы как пациентам, так и медицинскому персоналу [63].

Реализация данной системы помогает автоматизировать многие процессы в медицинском центре, что позволяет медицинским работникам сосредоточиться на непосредственном лечении пациентов, а также улучшает общую эффективность работы центра.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Абросимова И. А., Данилов С. И. Социально-этические проблемы развития искусственного интеллекта // Гуманитарные и экономические стратегии общероссийского и регионального развития. – 2018. – С. 13-19..
2. Артеменко М. В. и др. Синтез antecedenta продукционного правила медицинских экспертных систем логическими искусственными нейронными сетями // Вестник евразийской науки. – 2018. – Т. 10. – №. 4. – С. 38..
3. Аюш Ю. Анализ результатов тестирования медицинской экспертной системы на основе интеграции традиционной западной и восточной медицины // Системы анализа и обработки данных. – 2019. – №. 1 (74). – С. 141-156..
4. Бабаев Р. Р. Обзор и анализ современных интеллектуальных систем для управления медицинскими учреждениями // xx республиканская научная конференция магистрантов. – 2021. – С. 227-229.
5. Белозеров В. А. и др. Дифференциальная диагностика очаговых образований поджелудочной железы по данным эндоскопической ультрасонографии на основе анализа эхографической текстуры с использованием нечетких математических моделей // Онкологический журнал: лучевая диагностика, лучевая терапия. – 2021. – Т. 4. – №. 3. – С. 64-73.
6. Бледжянц Г. А. и др. Ключевые технологии формирования искусственного интеллекта в медицине // Ремедиум. Журнал о российском рынке лекарств и медицинской технике. – 2015. – №. 12. – С. 10-15.
7. Богомолов А.И., Невежин В.П., Жданов Г.А. Искусственный интеллект и экспертные системы в мобильной медицине // Хроноэкономика. 2018.
8. Брызгалина Е. В. Медицина в оптике искусственного интеллекта: философский контекст будущего // Человек. – 2019. – Т. 30. – №. 6. – С. 54-71.
9. Виен Л. Н. Дистанционная диагностическая система на основе гибридных моделей знаний: дис. – Виен Ле Нгуен.–Волгоград: ВолгГТУ, 2015.– 123 с, 2015 (дата обращения: 25.04.2024).

10. Гаврилов А. В. Системы искусственного интеллекта // Новосибирск: Изд-во НГТУ. – 2001 (дата обращения: 25.04.2024).

11. Головин П.А., Нечаев В.А., Нечаев Д.А. Экспертные системы для классификации болезней в медицинской диагностике // Научно-технический вестник информационных технологий, механики и оптики. 2006 (дата обращения: 25.04.2024).

12. Гусев А.В. Перспективы нейронных сетей и глубокого машинного обучения в создании решений для здравоохранения // Врач и информационные технологии. 2017. №3 (дата обращения: 25.04.2024).

13. Држевецкий Юрий Алексеевич, Затылкин Александр Валентинович, Юрков Николай Кондратьевич Экспертные системы как Прикладная область искусственного интеллекта // НиКа. 2011 (дата обращения: 25.04.2024).

14. Дудов О. А., Фетисов В. С. Алгоритмические аспекты построения систем для электропунктурной диагностики состояния пациента // Вестник Уфимского государственного авиационного технического университета. – 2008. – Т. 11. – №. 2. – С. 169-177 (дата обращения: 25.04.2024).

15. Дэвенпорт Т. Внедрение искусственного интеллекта в бизнес-практику: Преимущества и сложности. – Альпина Паблишер, 2021 (дата обращения: 25.04.2024).

16. Елфимов Д. А. и др. Применение информационных технологий в практическом здравоохранении // Медицинская наука и образование Урала. – 2019. – Т. 20. – №. 1. – С. 129-132 (дата обращения: 25.04.2024).

17. Ефименко И. В., Хорошевский В. Ф. Интеллектуальные системы поддержки принятия решений в медицине: ретроспективный обзор состояния исследований и разработок и перспективы. – 2017 (дата обращения: 25.04.2024).

18. Катасёв А. С. Математическое и программное обеспечение формирования баз знаний мягких экспертных систем диагностики состояния сложных объектов. – 2013 (дата обращения: 25.04.2024).

19. Катасёв А. С. Модели и методы формирования нечетких правил в интеллектуальных системах диагностики состояния сложных объектов : дис. – Казанский национальный исследовательский технологический университет, 2014 (дата обращения: 25.04.2024).

20. Кобринский Б. А. Консультативные интеллектуальные медицинские системы: классификации, принципы построения, эффективность // Врач и информационные технологии. – 2008. – №. 2. – С. 38-47 (дата обращения: 25.04.2024).

21. Кобринский Б. А. Системы поддержки принятия решений в здравоохранении и обучении // Врач и информационные технологии. – 2010. – №. 2. – С. 39-45 (дата обращения: 25.04.2024).

22. Кобринский Б. А., Хавкин А. И., Волынец Г. В. Перспективы применения систем искусственного интеллекта в гастроэнтерологии // Экспериментальная и клиническая гастроэнтерология. – 2020. – №. 7 (179). – С. 109-117 (дата обращения: 25.04.2024).

23. Кобринский Б.А. Системы поддержки принятия врачебных решений: история и современные решения // Методология и технология непрерывного профессионального образования. 2020. №4 (дата обращения: 25.04.2024).

24. Ковалев С. М. и др. Аналитический обзор трудов конференции ПТГ21 // Вестник Ростовского государственного университета путей сообщения. – 2021. – №. 4. – С. 100-121 (дата обращения: 25.04.2024).

25. Кочеткова И. А., Рубанов В. Г. Методы и алгоритмы принятия решений на основе механизмов визуализации и нечеткой логики. – 2015 (дата обращения: 25.04.2024).

26. Курочкин А. В. Модель, методы и алгоритмы построения информационно-аналитических экспертных систем учета и анализа данных медицинских обследований. – 2022 (дата обращения: 25.04.2024).

27. ЛУНИНА Н. В. и др. Представлением знаний для проектирования и диагностики биотехнических изделий. – 1991 (дата обращения: 25.04.2024).

28. Лобанов В. В. Модели и методы синтеза и реализации специализированных гибридных экспертных систем : дис. – Саратовский государственный технический университет, 2006 (дата обращения: 25.04.2024).

29. Мальцев Э. Г., Собакин С. Л. Использование интеллектуальных систем в лечебно-диагностическом процессе типового многопрофильного лечебного учреждения // Инноватика и экспертиза: научные труды. – 2019. – №. 3. – С. 100-112 (дата обращения: 25.04.2024).

30. Мелдо А. А., Уткин Л. В., Трофимова Т. Н. Искусственный интеллект в медицине: современное состояние и основные направления развития интеллектуальной диагностики // Лучевая диагностика и терапия. – 2020. – Т. 11. – №. 1. – С. 9-17 (дата обращения: 25.04.2024).

31. Молчанов А. А. Использование экспертных систем в системе открытого образования // Гаудеамус. – 2014. – №. 2 (24). – С (дата обращения: 25.04.2024).

32. Морозова Ю. А. Цифровая трансформация российского здравоохранения как фактор развития отрасли // Интеллект. Инновации. Инвестиции. – 2020. – №. 2. – С. 36-47 (дата обращения: 25.04.2024).

33. Нестерова Е. В. и др. О некоторых подходах к построению самообучающихся экспертных систем // Экономика. Информатика. – 2012. – Т. 21. – №. 1-1 (120). – С. 162-165 (дата обращения: 25.04.2024).

34. Новиков Д. Б. Автоматизация оценки состояния здоровья человека для обеспечения безопасности длительного пребывания в условиях изоляции на основе системы поддержки принятия решений // Автореф. дисс.... канд. тех. наук. М., 2007. 25 с. – 2007 (дата обращения: 25.04.2024).

35. Оразбаев Б. Б. Экспертные системы для медицинской диагностики с применением методов теории нечетких множеств // ИТпортал. – 2016. – №. 4 (12). – С. 1 (дата обращения: 25.04.2024).

36. Поповская Т. Н., Раскин Л. Г., Серая О. В. Информационные технологии диагностики–медицинские экспертные системы // Клиническая

информатика и телемедицина. – 2004. – №. 1. – С. 81-85 (дата обращения: 25.04.2024).

37. Пустынникова И. Н. Технология использования экспертных систем для диагностики знаний и умений // Образовательные технологии и общество. – 2001. – Т. 4. – №. 4. – С. 77-101 (дата обращения: 25.04.2024).

38. Пылькин А. Н. и др. Построение медицинских экспертных систем сопровождения медикотехнологического процесса // Вестник РГРТУ. – 2017. – №. 60. – С. 123-130 (дата обращения: 25.04.2024).

39. Румовская С. Б., Колесников А. В., Литвин А. А. Компьютерная персонализированная диагностика. Проблемы и перспективы // Гибридные и синергетические интеллектуальные системы. – 2022. – С. 9-33 (дата обращения: 25.04.2024).

40. Рыбина Г. В. и др. Программные средства и технология автоматизированного построения интегрированных экспертных систем // Программные продукты и системы. Software & Systems. – 1997. – №. 4. – С. 37 (дата обращения: 25.04.2024).

41. Степанова М. Д. и др. ББК 32.813 С 79 (дата обращения: 25.04.2024).

42. Степанова М. Д., Самодумкин С. А. Прикладные интеллектуальные системы в области медицины: учебно-метод. пособие. – 2000 (дата обращения: 25.04.2024).

43. Татжибаева О. А. Разработка экспертных систем: методические указания к расчетно-графическим работам по дисциплине " Системы искусственного интеллекта". – 2005 (дата обращения: 25.04.2024).

44. Творошенко И. С. Структура и функции интеллектуальных средств принятия решений в сложных системах // Искусственный интеллект. – 2004. – №. 4. – С. 462-470 (дата обращения: 25.04.2024).

45. Тополь Э. Искусственный интеллект в медицине: Как умные технологии меняют подход к лечению. – Альпина Паблишер, 2022 (дата обращения: 25.04.2024).

46. Ульянов С. В., Колбенко Е. В. Гибридная экспертная система с глубинным представлением знаний для проектирования и диагностики биотехнических изделий // Системный анализ в науке и образовании. – 2011. – №. 2. – С. 112-135 (дата обращения: 25.04.2024).

47. Хохлов Александр Леонидович, Белоусов Дмитрий Юрьевич Этические аспекты применения программного обеспечения с технологией искусственного интеллекта // Качественная клиническая практика. 2021. №1 (дата обращения: 25.04.2024).

48. Червинская К. Р. Психология извлечения экспертных знаний субъектов труда // Автореферат по ВАК. – 2010. – Т. 19. – №. 03 (дата обращения: 25.04.2024).

49. Шумский В. И. и др. Компьютерные медицинские экспертные системы как средство повышения эффективности диагностики и лечения // Альманах клинической медицины. – 1999. – №. 2. – С. 363-376 (дата обращения: 25.04.2024).

50. Щелкова О. Ю. Психологическая диагностика в медицине (системное исследование) // СПб.: Санкт-Петербургский гос. университет. – 2009 (дата обращения: 25.04.2024).

51. Юдин В. И., Широкова О. В. Перспективы и роль цифровых технологий в реформировании российского здравоохранения // Здоровье мегаполиса. – 2020. – Т. 1. – №. 1 (дата обращения: 25.04.2024).

52. Ясницкий Л. Н. Медицина и искусственный интеллект глазами пермских ученых // Издается по решению кафедры прикладной математики и информатики Пермского государственного национального исследовательского университета Сборник подготовлен при финансовой поддержке Министерства образования и науки Пермского края: Договор Д-26/120 от 27.07. 2020, а также гранта РФФИ: № 19-010-00307. – 2020. – С. 51. (дата обращения: 25.04.2024).

53. Яхшибоева Д. Э., Эрметов Э. Я., Яхшибоев Р. Э. Развитие цифровых технологии в медицине // Journal of new century innovations. – 2023. – Т. 20. – №. 1. – С. 100-107 (дата обращения: 25.04.2024).

54. Disease Symptom Prediction URL: <https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset?resource=download&select=dataset.csv> (дата обращения: 25.04.2024).

55. Confusion_matrix URL : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (дата обращения: 25.04.2024).

56. Царев В. П. Внутренние болезни // Учебное пособие. – Минск: Издательство "Новое знание", 2013. – 512 с. (дата обращения: 25.04.2024).

57. Кондауров Д. В., Иванов А. П., Петров С. М. Основы программирования на Python // Учебное пособие. – Москва: Издательство "Наука", 2021. – 256 с. (дата обращения: 04.06.2024).

58. Петров И. В., Сидоров А. Л., Кузнецов Н. М. Метод машинных векторов: теория и практика // Учебное пособие. – Санкт-Петербург: Издательство "Технолит", 2019. – 320 с. (дата обращения: 04.06.2024).

59. Смирнов Д. А., Козлов Е. Н., Андреев В. П. Паттерн проектирования MVC (Model-View-Controller): принципы и применение // Учебное пособие. – Москва: Издательство "Технополис", 2022. – 288 с. (дата обращения: 04.06.2024).

60. Иванов С. П., Кузьмин В. А., Лебедев Н. Ю. SQLite: основы и практика работы с базами данных // Учебное пособие. – Новосибирск: Издательство "Инфра-М", 2020. – 256 с. (дата обращения: 04.06.2024).

61. Сергеев А. Н., Петров Д. В., Смирнова Е. И. Веб-разработка с использованием Django: руководство для начинающих // Учебное пособие. – Санкт-Петербург: Издательство "Питер", 2021. – 384 с. (дата обращения: 04.06.2024).

62.Васильев И. А., Романов П. В., Тихонов М. Н. Docker: контейнеризация приложений // Учебное пособие. – Москва: Издательство "Диалектика", 2020. – 320 с. (дата обращения: 04.06.2024).

63.Кобринский Б. А., Хавкин А. И., Волынец Г. В. Перспективы применения систем искусственного интеллекта в гастроэнтерологии // Экспериментальная и клиническая гастроэнтерология. – 2020. – №. 7 (179). – С. 109-117. (дата обращения: 25.04.2024).

ПРИЛОЖЕНИЕ А

(обязательное)

Интеллектуальная экспертная система диагностики и учета пациентов

медицинского центра

Листинг программы

Файл index.hmt

```
<!doctype html>
<html lang="ru">

<head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Запись к врачу по симптомам</title>
</head>

<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>

</html>
```

Файл vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

Файл app.css

```
#root {
  /* max-width: 1280px; */
  display: flex;
  flex-direction: column;
  justify-content: stretch;
  /* align-items: center; */
  width: 100%;
  margin: 0;
  padding: 0;
  text-align: center;
}
```

Файл Index.css

```
@font-face {
  font-family: 'Inter';
  src: url('../assets/fonts/Inter-Bold.ttf');
  font-weight: bold;
}
```

```

    font-style: normal;
}

@font-face {
    font-family: 'Inter';
    src: url('../assets/fonts/Inter-Light.ttf');
    font-weight: normal;
    font-style: normal;
}

* {
    /* all: initial; */
    box-sizing: border-box;
}

html,
body {
    all: initial;
    box-sizing: border-box;
    margin: 0 !important;
    padding: 0 !important;
    width: 100%;
    height: 100%;
}

:root {

    /* font-family: Inter, system-ui, Avenir, Helvetica, Arial,
    sans-serif; */
    line-height: 1.5;
    font-weight: 400;
    /* color-scheme: light dark; */
    /* color: rgba(255, 255, 255, 0.87); */
    background-color: white;

    font-synthesis: none;
    text-rendering: optimizeLegibility;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;

```

```

/* Colors */
--bl: #0452CE;
--bl1: #3675D8;
--bl2: #6897E2;
--bl3: #9BBAEB;
--bl4: #CDDCF5;
--bl5: #0342A5;
--bl6: #02317C;
--bl7: #022152;
--bl8: #011029;

}

h1 {
  margin: 0;
  font-family: Inter;
  font-size: 64px;
  font-weight: bold;
  line-height: 64px;
  /* color: var(--bl); */
  word-break: break-word;
}

h2 {
  margin: 0;

  font-family: Inter;
  font-size: 48px;
  font-weight: normal;
  line-height: 48px;
  word-break: break-word;
}

p {
  margin: 0;

  font-family: Inter;
  font-size: 18px;
  font-weight: normal;
  line-height: 32px;
  word-break: break-word;
}

a {
  all: initial;
  margin: 0;
  font-family: Inter;
  font-size: 18px;
  font-weight: normal;

```

```

    line-height: 18px;
    cursor: pointer;
    color: var(--bl);
    text-decoration: underline;
  }

  @media screen and (max-width: 768px) {
    h1 {
      font-size: 48px;
      line-height: 48px;
    }

    h2 {
      font-size: 32px;
      line-height: 32px;
    }
  }

  input[type=time],
  input[type=date] {
    all: initial;
    height: 56px;
    border: 2px solid var(--bl);
    border-radius: 8px;
    font-size: 18px;
    font-family: Inter;
    padding: 0 24px;
    color: #000;
    width: 100%;
  }

  input[type=time]:focus,
  input[type=date]:focus {
    border: 2px solid var(--bl2);
  }

```

Файл main.jsx

```

import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
import { UserProvider } from './userContext.jsx'
document.documentElement.lang = "ru";
ReactDOM.createRoot(document.getElementById('root')).render(
  <UserProvider>
    <App />
  </UserProvider>
)

```

Файл AuthPage.jsx

```

import React, { useRef, useState } from "react";
import { useContext } from "react";
import UserContext from "../../userContext";
import client from "../../apiClient";
import MainCard from "../../components/MainCard/MainCard";
import MainCardContent from
"./MainCardContent/MainCardContent";
import LabelledTextInput from
"../../components/inputs/LabelledTextInput";
import BlueButtonWithWhiteFrame from
"../../components/buttons/BlueButtonWithWhiteFrame/BlueButtonWithWhi
teFrame";
import AuthForm from "../AuthForm/AuthForm";
import { Link, Navigate, redirect, useNavigate } from "react-
router-dom";
import axios from "axios";

export default function AuthPage() {
  const [user, setUser] = useContext(UserContext);
  const email = useRef(null);
  const password = useRef(null);
  const navigate = useNavigate();

  const handleEmailChange = (e) => {
    setEmail(e.target.value);
  }

  const handlePasswordChange = (e) => {
    setPassword(e.target.value);
  }
  const getApiUser = async () => {
    try {
      const response = await client.get("/api/user");
      if (response.status === 200) {
        setUser(response.data.user);
      }

    } catch (error) {
      setUser(null);
    }
  }
  const login = async () => {
    try {
      const response = await client.post(
        "/api/token",
        {

```

```

        email: email.current.value,
        password: password.current.value
    }
    );
    // console.log('Response');
    console.log(response);
    if (response.status === 200) {
        localStorage.clear();
        localStorage.setItem('access_token',
response.data.access);
        localStorage.setItem('refresh_token',
response.data.refresh);
        client.defaults.headers.common['Authorization']
= `Bearer ${response.data['access']}`;
        await getApiUser();

        // setUser({ email: response.data.email })
        return true
    }
}
catch (error) {
    if (error.isAxiosError) {

        if (error.response.status === 404 ||
error.response.status === 401) {
            return false;
        }
        } else {
            console.log(error);
        }
    }
    return false;
}

const handleSubmit = (e) => {
    e.preventDefault();
    login()
        .then(authStatus => {
            console.log(`AuthStatus ${authStatus}`)
            if (authStatus) {
                alert(`Logged with Email:
${email.current.value}`);
                navigate("/");
            }
            else {
                setUser(null);
                alert("User not found");
            }
        })
        .catch(error => console.error(error));
}

```

```

    }

    return (
      <>

        <MainCard>
          <h2>Авторизация</h2>
        </MainCard>
        <AuthForm
          onSubmit={handleSubmit}
          // onChangeEmail={handleEmailChange}
          // onChangePassword={handlePasswordChange}
          emailRef={email}
          passwordRef={password}
        />
        <Link style={{ textAlign: 'center' }}
to="/register" >Еще нет аккаунта? Зарегистрироваться </Link>
      </>
    )
  }
}

```

Файл MainPage.jsx

```

import React from "react";
import MainCard from "../components/MainCard/MainCard";
import MainCardContent from "../Content/MainCardContent";
import { motion } from 'framer-motion'
import { Link } from "react-router-dom";
export default function MainPage() {
  return (
    <>
      <MainCard >
        < MainCardContent />
      </MainCard>
      <motion.h1
        initial={{ y: -30, rotateX: 90, opacity: 0 }}
        whileInView={{ x: 0, y: 0, rotateX: 0, opacity:
100 }}

        transition={{ duration: 0.5 }}
        style={{ zIndex: -9999 }}
      >Как это работает?</motion.h1>

      <div style={{ display: "flex", flexDirection:
"column", gap: 32, maxWidth: 900, flexGrow: 1, alignSelf: "center"
}}>
        <motion.div initial={{ opacity: 0, y: -30,
rotateX: 90 }} whileInView={{ opacity: 100, y: 0, rotateX: 0 }}
transition={{ duration: 0.5, delay: 0.3 }}>
          <p>

```

Все, что нужно, чтобы вы создали свой аккаунт, а затем записались к врачу, введя Ваши симптомы

```

        </p>
      </motion.div>
      <motion.div initial={{ opacity: 0, y: -30,
rotateX: 45 }} whileInView={{ opacity: 100, y: 0, rotateX: 0 }}
transition={{ duration: 0.5, delay: 0.7 }}>
        <p>

```

Запись к врачу работает полностью в автоматическом режиме благодаря использованию модели машинного обучения SVC.

Подробнее можно изучить на <Link to="about">странице информации.</Link>

```

        </p>
      </motion.div>
    </div>
  </>
)
}

```

Файл RegistrePage.jsx

```

import React, { useContext, useEffect, useRef, useState } from
"react";
import { useNavigate } from "react-router-dom";
import client from "../../apiClient";
import MainCard from "../../components/MainCard/MainCard";
import UserContext from "../../userContext";
import MainCardContent from
"./MainCardContent/MainCardContent";
import RegisterForm from "./RegisterForm/RegisterForm";

const validatePassword = (password) => {
  if (password.length < 8)
    return false
  return true
}

export default function RegisterPage() {
  const { user } = useContext(UserContext);
  const email = useRef();
  const username = useRef();
  const password = useRef(null);
  const password2 = useRef(null);
  const navigation = useNavigate();

  useEffect(() => {
    if (user) {
      navigation("/");
    }
  }, [])

  const register = async () => {

```

```

    try {
      const response = await client.post(
        "/api/register",
        {
          email: email.current.value,
          username: username.current.value,
          password: password.current.value,
        }
      );
      console.log(response)
      if (response.status === 201) {
        alert(`Запегистрирован\nusername:
        ${response.data.username}\nemail: ${response.data.email}`);
      }
    } catch (error) {

      if (error.isAxiosError) {
        if (error.response.status === 500) {
          alert("Внутренняя ошибка. Возможно, что
          логин и пароль уже существуют");
        }
      }
    }
  }

  const handleSubmit = (e) => {
    // Проверяем пароли
    e.preventDefault();
    if (validatePassword(password.current.value) &&
    password.current.value == password2.current.value) // Если первый
    верен и равен второму, значит все верно
      register()
    else
      alert("Пароли должны совпадать. Каждый пароль
      должн превышать 8 символов")
  }

  const handleChangePasswords = () => {

    let password_elemet =
    document.getElementById("password");
    let password2_element =
    document.getElementById("password2");
    if (password.current &&
    validatePassword(password.current.value)) {

```

```

        password_elemet.classList.remove('error');
    } else {
        password_elemet.classList.add('error');
    }
    if (password2.current && ((password.current.value !==
password2.current.value) ||
!validatePassword(password2.current.value))) {
        password2_element.classList.add('error');
    }
    else {
        password2_element.classList.remove('error');
    }
}

return (
    <>
        <MainCard >
            <MainCardContent />
        </MainCard>

        <RegisterForm
            email={email}
            username={username}
            password2={password2}
            password={password}
            onSubmit={handleSubmit}
            onPasswordChange={handleChangePasswords}
        />
    </>
)
}

```

Файл views.py

```

import logging
from django.http import HttpRequest
from django.utils.datastructures import MultiValueDictKeyError
from rest_framework import permissions, status
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.viewsets import ModelViewSet
from rest_framework_simplejwt.exceptions import TokenError
from rest_framework_simplejwt.tokens import RefreshToken

from user_api.models import AppUser

```

```

from .serializers import UserRegisterSerializer, UserSerializer
from .validations import custom_validation
from .validations import validate_email

class UserRegister(APIView):
    """Register new user"""

    permission_classes = (permissions.AllowAny,)

    def post(self, request: HttpRequest) -> Response | None:
        logging.error("ERROR")
        clean_data = custom_validation(data=request.data)
        serializer = UserRegisterSerializer(data=clean_data)
        if serializer.is_valid(raise_exception=True):
            user = serializer.create(clean_data)
            if user:
                return Response(serializer.data,
status=status.HTTP_201_CREATED)

class UserLogout(APIView):
    permission_classes = (permissions.AllowAny,)
    # authentication_classes = ()

    def post(self, request: HttpRequest) -> Response:
        try:
            print(request.data)
            refresh_token = request.data["refresh_token"]
            token = RefreshToken(refresh_token)
            token.blacklist()
            return
Response(status=status.HTTP_205_RESET_CONTENT)
        except TokenError as e:
            # print(e)
            return Response({"details": str(e)},
status=status.HTTP_401_UNAUTHORIZED)
        except (MultiValueDictKeyError, KeyError):
            return Response(status=status.HTTP_400_BAD_REQUEST)
        # logout(request=request)
        # return Response(status=status.HTTP_200_OK)

class UserView(APIView):
    permission_classes = (permissions.IsAuthenticated,)
    # authentication_classes = (SessionAuthentication,)

    def get(self, request: HttpRequest) -> Response:
        serializer = UserSerializer(request.user)
        return Response({"user": serializer.data},
status=status.HTTP_200_OK)

```

```

def put(self, request: HttpRequest) -> Response:
    data = request.data
    print("data = ", data)
    if "password" in data:
        del data["password"]
    if "email" in data:
        assert validate_email(data)
    user = request.user
    serializer = UserSerializer(user, request.data,
partial=True)

    if serializer.is_valid():
        serializer.save()
        return Response({"user": serializer.data},
status=status.HTTP_200_OK)
    return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

```

```

class AllUsersSet(ModelViewSet):
    permission_classes = (permissions.IsAdminUser,)

    queryset = AppUser.objects.all()
    serializer_class = UserSerializer
    # permission_classes = permissions.IsAdminUser
    # def get(request: HttpRequest) -> HttpResponse:
    # serializer = UserSerializer(AppUser)

```

Файл validators.py

```

from django.core.exceptions import ValidationError

NUMBER_OF_SYMPTOMS = 17

def symptoms_validator(data) -> bool:
    if "symptoms_ids" not in data:
        raise ValidationError("A symptoms_ids field is needed")
    if not isinstance(data["symptoms_ids"], list):
        raise ValidationError("A symptoms_ids field must be
list")
    if len(data["symptoms_ids"]) == 0 or
len(data["symptoms_ids"]) > NUMBER_OF_SYMPTOMS:
        raise ValidationError("Number of symptoms should be > 0
and <= 17")
    return True

```

Файл views.py:

```

from django.http import HttpRequest, HttpResponse
from django.shortcuts import get_list_or_404, get_object_or_404

```

```

import numpy as np
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import permissions
from rest_framework import status

from classification_api.classification_model import
ClassificationModel
from classification_api.validators import symptoms_validator,
NUMBER_OF_SYMPTOMS
from disease_api.models import DiseaseModel, SymptomModel
from disease_api.serializers import DiseaseSerializer

# Create your views here.
class ClassificationDiseaseBySymptoms(APIView):
    # permission_classes = (permissions.IsAuthenticated,)
    model = ClassificationModel()

    def post(self, request: HttpRequest) -> HttpResponse:
        try:
            symptoms_validator(request.data)
        except Exception as e:
            return Response({"details": str(e)},
status.HTTP_400_BAD_REQUEST)
        symptoms = request.data["symptoms_ids"]
        # print(symptoms)
        symptoms_objects =
SymptomModel.objects.filter(symptom_id__in=symptoms).all()
        symptoms_objects = get_list_or_404(symptoms_objects)
        user_weights = [symptom.weight for symptom in
symptoms_objects]
        weights = np.zeros(NUMBER_OF_SYMPTOMS).astype(int)
        weights[: len(user_weights)] = np.array(user_weights)
        disease_name = self.model.get_disease(weights)
        # print("disease_name", disease_name)
        disease = get_object_or_404(DiseaseModel,
name=disease_name)
        serializer = DiseaseSerializer(disease)
        return Response(serializer.data,
status=status.HTTP_200_OK)

```

Файл serializers.py

```

from rest_framework import serializers

from disease_api.models import (
    DiseaseModel,
    DiseaseTranslationModel,
    SymptomModel,
    SymptomTranslationModel,

```

```

)

class AddDiseaseSerializer(serializers.ModelSerializer):
    class Meta:
        model = DiseaseModel
        fields = ("name",)

class DiseaseSerializer(serializers.ModelSerializer):
    class Meta:
        model = DiseaseModel
        fields = "__all__"

# class
AddDiseaseTranslationSerializer(serializers.ModelSerializer):
#     class Meta:
#         model = DiseaseTranslationModel
#         fields =
class
DiseaseTranslationSerializer(serializers.ModelSerializer):
    def create(self, validated_data):
        disease_translation = DiseaseTranslationModel(
            name_t=validated_data["name_t"],
            disease=DiseaseModel.objects.filter(
                disease_id=validated_data["disease"]
            ).first(),
        )
        disease_translation.save()
        return disease_translation

    class Meta:
        model = DiseaseTranslationModel
        fields = "__all__"

class AddSymptomSerializer(serializers.ModelSerializer):
    class Meta:
        model = SymptomModel
        fields = ("name", "weight")

class SymptomSerializer(serializers.ModelSerializer):
    class Meta:
        model = SymptomModel
        fields = "__all__"

class
SymptomTranslationSerializer(serializers.ModelSerializer):

```

```

def create(self, validated_data):
    symptom_translation = SymptomTranslationModel(
        name_t=validated_data["name_t"],
        symptom=SymptomModel.objects.filter(
            symptom_id=validated_data["symptom"]
        ).first(),
    )
    symptom_translation.save()
    return symptom_translation

class Meta:
    model = SymptomTranslationModel
    fields = "__all__"

```

Файл views.py

```

from django.http import HttpRequest, HttpResponse
from django.shortcuts import get_object_or_404
from rest_framework import permissions, status
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.viewsets import ModelViewSet

from disease_api.models import (
    DiseaseModel,
    DiseaseTranslationModel,
    SymptomModel,
    SymptomTranslationModel,
)
from disease_api.serializers import (
    AddDiseaseSerializer,
    AddSymptomSerializer,
    DiseaseSerializer,
    DiseaseTranslationSerializer,
    SymptomSerializer,
    SymptomTranslationSerializer,
)

# Create your views here.
class DiseaseViewSet(ModelViewSet):
    queryset = DiseaseModel.objects.all()
    serializer_class = DiseaseSerializer

    def retrieve(self, request: HttpRequest, pk: int = None) ->
HttpResponse:
        queryset = self.queryset
        disease = get_object_or_404(queryset, disease_id=pk)
        serializer = self.serializer_class(disease)
        return Response(serializer.data,
status=status.HTTP_200_OK)

```

```

class DiseaseAdd(APIView):
    permission_classes = (permissions.IsAdminUser,)

    def post(self, request: HttpRequest) -> HttpResponse:
        # cleaned_data = request.data.strip()
        serializer = AddDiseaseSerializer(data=request.data)
        if serializer.is_valid(raise_exception=True):
            disease = serializer.create(request.data)
            if disease:
                return Response(serializer.data,
status=status.HTTP_201_CREATED)

class DiseaseTranslationViewSet(ModelViewSet):
    queryset = DiseaseTranslationModel.objects.all()
    serializer_class = DiseaseTranslationSerializer

    def retrieve(self, request: HttpRequest, pk: int = None) ->
HttpRequest:
        queryset = self.queryset
        disease_t = get_object_or_404(queryset, disease_id=pk)
        serializer = self.serializer_class(disease_t)
        return Response(serializer.data,
status=status.HTTP_200_OK)

class DiseaseTranslationAdd(APIView):
    permission_classes = (permissions.IsAdminUser,)

    def post(self, request: HttpRequest) -> HttpResponse:
        serializer =
DiseaseTranslationSerializer(data=request.data)
        if serializer.is_valid(raise_exception=True):
            disease_translate = serializer.create(request.data)
            if disease_translate:
                return Response(serializer.data,
status=status.HTTP_201_CREATED)

class SymptomViewSet(ModelViewSet):
    queryset = SymptomModel.objects.all()
    serializer_class = SymptomSerializer

    def retrieve(self, request, pk: int = None) ->
HttpResponse:
        queryset = self.queryset
        symptom = get_object_or_404(queryset, symptom_id=pk)
        serializer = self.serializer_class(symptom)

```

```

        return Response(serializer.data,
status=status.HTTP_200_OK)

class SymptomAdd(APIView):
    permission_classes = (permissions.IsAdminUser,)

    def post(self, request: HttpRequest) -> HttpResponse:
        serializer = AddSymptomSerializer(data=request.data)
        if serializer.is_valid(raise_exception=True):
            symptom = serializer.create(request.data)
            if symptom:
                return Response(serializer.data,
status=status.HTTP_201_CREATED)

class SymptomTranslationViewSet(ModelViewSet):
    queryset = SymptomTranslationModel.objects.all()
    serializer_class = SymptomTranslationSerializer

    def retrieve(self, request: HttpRequest, pk: int):
        queryset = self.queryset
        symptom_t = get_object_or_404(queryset, symptom_id=pk)
        serializer = self.serializer_class(symptom_t)
        return Response(serializer.data,
status=status.HTTP_200_OK)

class SymptomTranslationAdd(APIView):
    permission_classes = (permissions.IsAdminUser,)

    def post(self, request: HttpRequest) -> HttpResponse:
        serializer =
SymptomTranslationSerializer(data=request.data)
        if serializer.is_valid(raise_exception=True):
            symptom_translation =
serializer.create(request.data)
            if symptom_translation:
                return Response(serializer.data,
status=status.HTTP_201_CREATED)

```

Файл TrainModel.ipynb

```

# # Обучение модели
# ## Импортирование всех библиотек
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import f1_score, accuracy_score,
confusion_matrix
import seaborn as sns
import numpy as np

```

```

# ## Загрузка данных
df = pd.read_csv("../dataset/dataset.csv")
df1 = pd.read_csv("../dataset/Symptom-severity.csv")
df.head()
df1.head()
df.sample(10)
df.info()
df.Disease.unique(), len(df.Disease.unique())
df1.info()
df1.Symptom.unique(), len(df1.Symptom.unique())
# ## Очистка данных
print(df.isna().sum())
print(df.isnull().sum())
cols = df.columns
data = df[cols].values.flatten()
data
s = pd.Series(data)
s = s.str.strip()
s = s.values.reshape(df.shape)
s
df=pd.DataFrame(s, columns=df.columns)
df = df.fillna(0)
df.head()
df.Symptom_1.unique()
# ## Сопоставление симптомов с их весами
import warnings
warnings.filterwarnings('ignore')

values = df.values
symptoms = df1['Symptom'].unique()

for i in range(len(symptoms)):
    values[values == symptoms[i]] = df1[df1['Symptom'] ==
symptoms[i]]['weight'].values[0]

d = pd.DataFrame(values, columns=cols)

d = d.replace('dischromic _patches', 0)
d = d.replace('spotting_ urination',0)
df = d.replace('foul_smell_of urine',0)
df.head()
df[df.Disease=="Peptic ulcer disease"]
df.Symptom_10.unique()
# ## Создание обучающей и тестовой выборки
data = df.iloc[:,1:].values
labels = df['Disease'].values
x_train, x_test, y_train, y_test = train_test_split(data,
labels, shuffle=True, train_size = 0.85)
help(SVC)
# ## Обучение модели

```

```

    from sklearn.model_selection import train_test_split,
GridSearchCV
    param_grid = {
        'C': [0.1, 1, 10, 100],
        'gamma': [1, 0.1, 0.01, 0.001],
        'kernel': ['rbf', 'linear', 'poly', 'sigmoid']
    }

    svc = SVC()
    grid_search = GridSearchCV(estimator=svc,
param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
    grid_search.fit(x_train, y_train)
    model = grid_search.best_estimator_
    model
    model = grid_search.best_estimator_
    preds = model.predict(x_test)
    conf_mat = confusion_matrix(y_test, preds)
    df_cm = pd.DataFrame(conf_mat, index=df['Disease'].unique(),
columns=df['Disease'].unique())
    print('F1-score% =', f1_score(y_test, preds,
average='macro')*100, '|', 'Accuracy% =', accuracy_score(y_test,
preds)*100)
    sns.heatmap(df_cm)
    # ## Сохранение модели
    import pickle
    filename = 'finalized_model.sav'
    pickle.dump(model, open(filename, 'wb'))

```

Файл LoadModel.ipynb

```

import pickle
filename = 'finalized_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
import numpy as np
x_test = np.array([1,1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 7, 5, 1, 5,
7, 1]).reshape(1, -1)
loaded_model.predict(x_test).item()
x_test.shape

```

ПРИЛОЖЕНИЕ Б

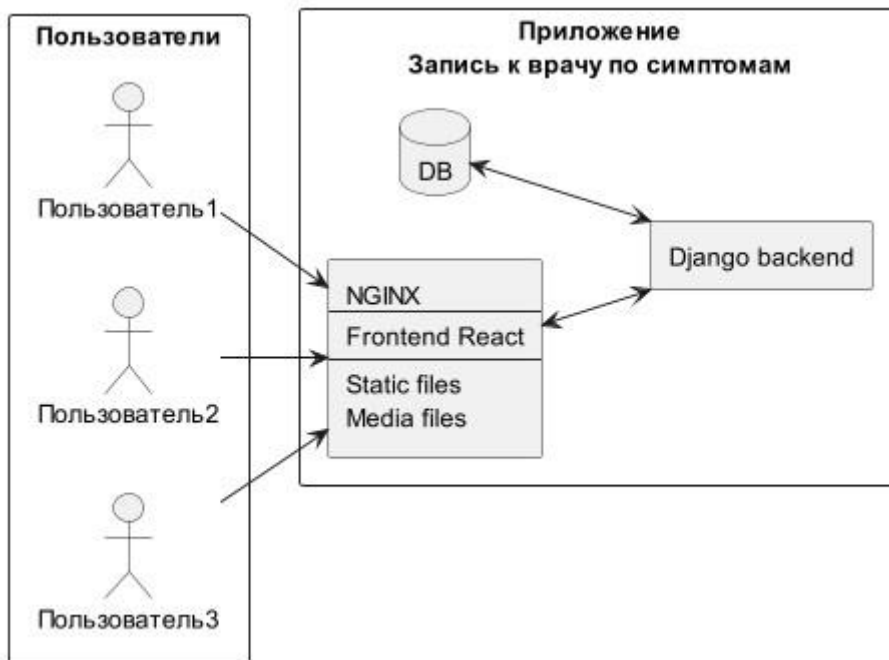
Интеллектуальная экспертная система диагностики и учета пациентов

медицинского центра

UML –диаграммы

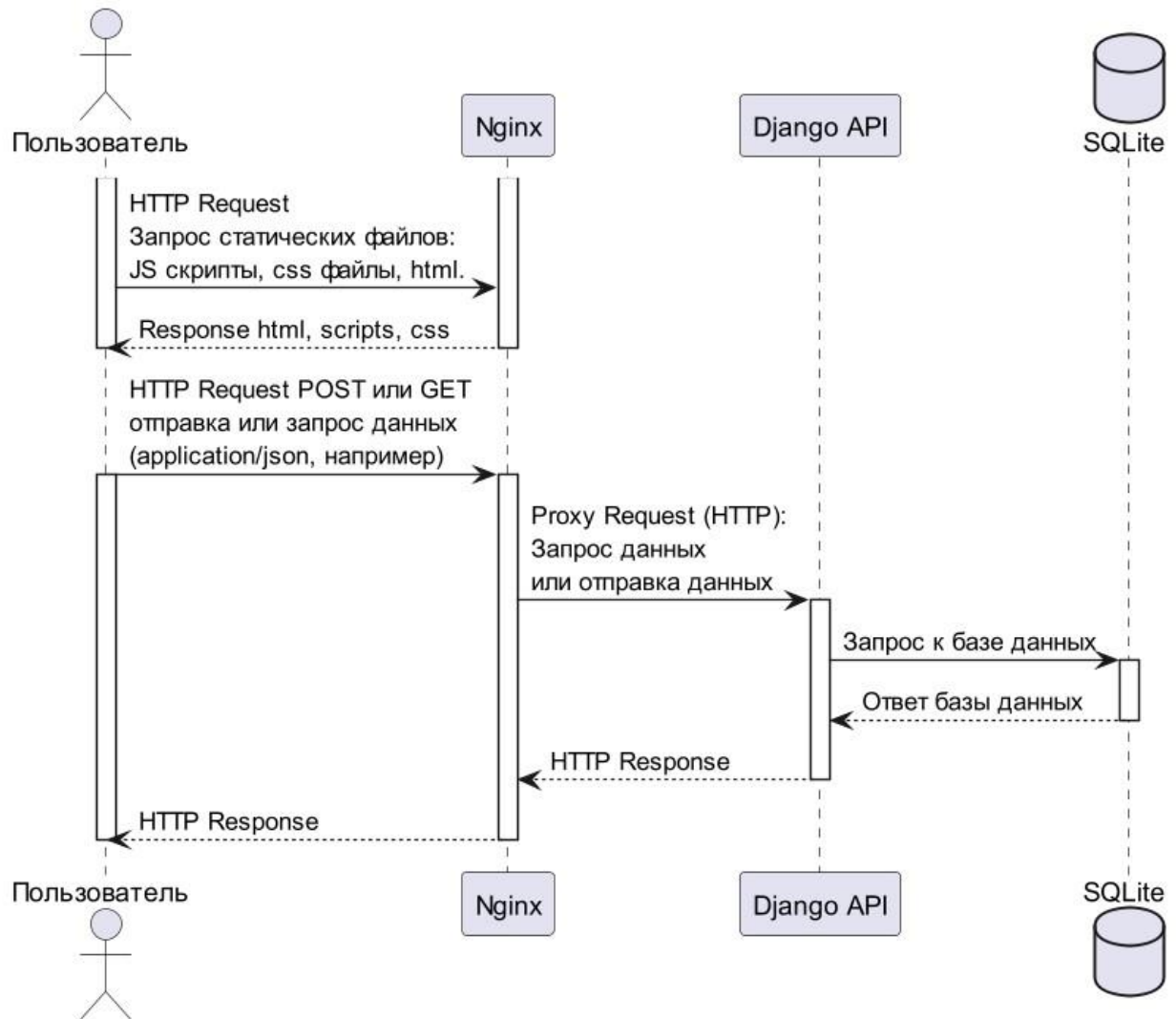
ПРИЛОЖЕНИЕ Б.1

Структура приложения на верхнем уровне



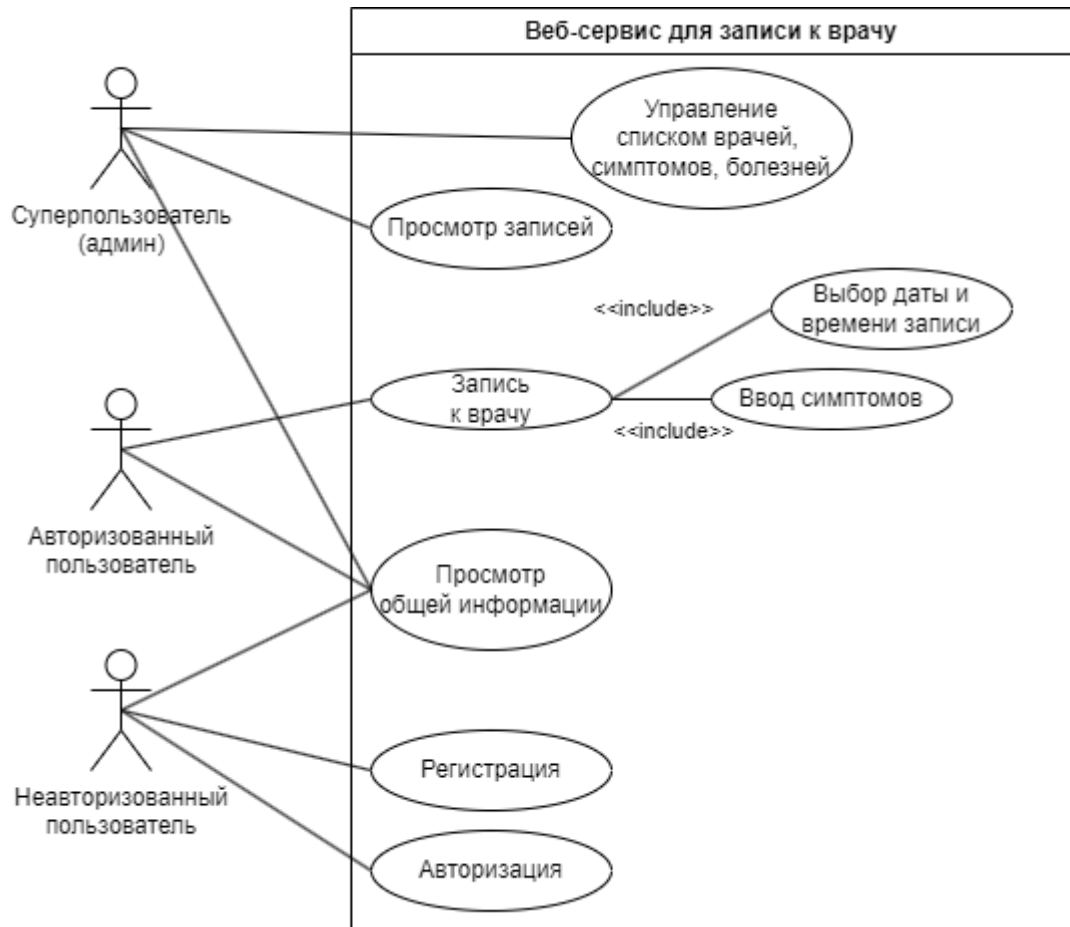
ПРИЛОЖЕНИЕ Б.2

Диаграмма последовательности обобщенная для записи к врачу по симптомам



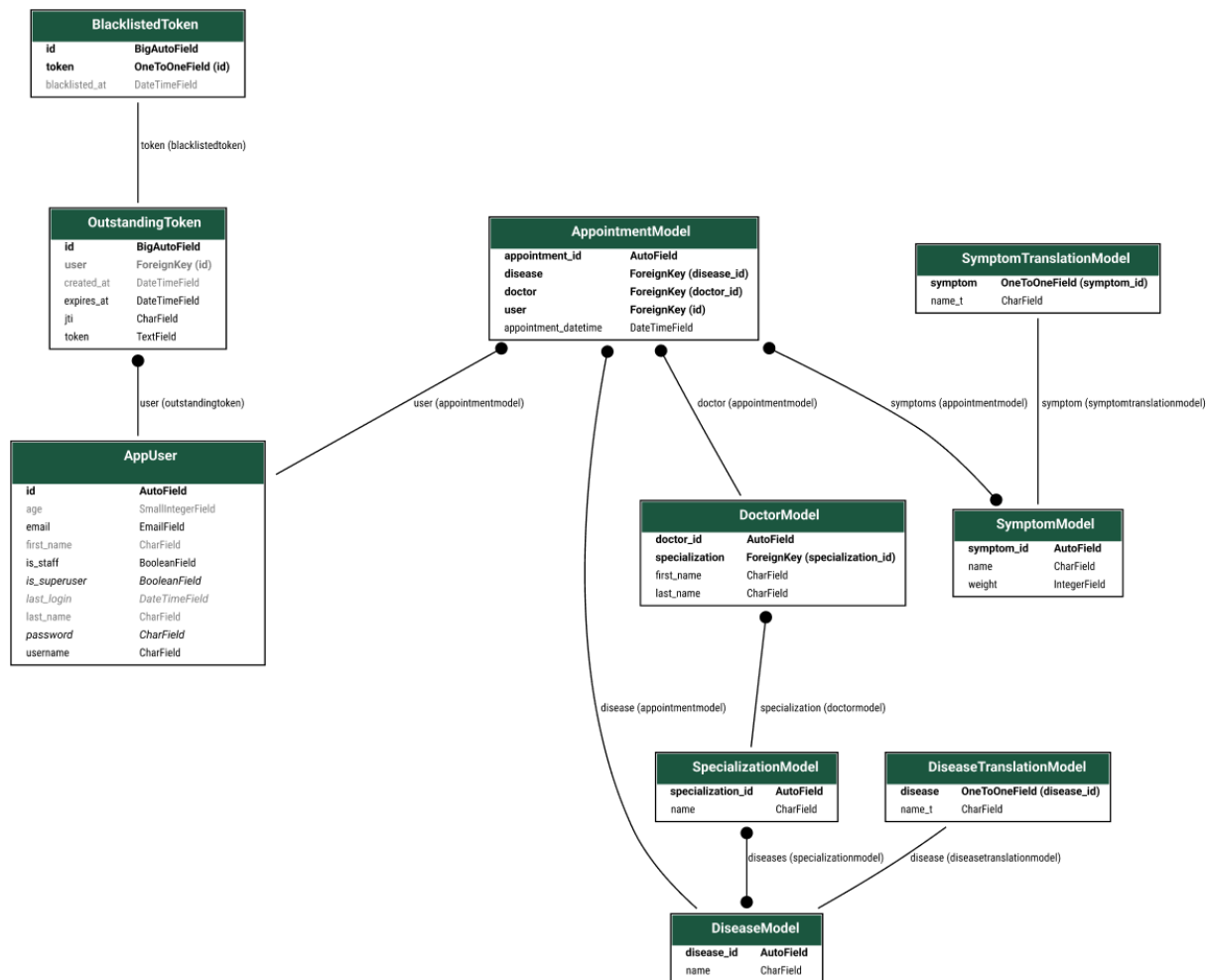
ПРИЛОЖЕНИЕ Б.3

Диаграмма последовательностей



ПРИЛОЖЕНИЕ Б.4

Структура базы данных



ПРИЛОЖЕНИЕ В

Интеллектуальная экспертная система диагностики и учета пациентов

медицинского центра

Презентация

ИНТЕЛЛЕКТУАЛЬНАЯ ЭКСПЕРТНАЯ СИСТЕМА ДИАГНОСТИКИ И УЧЕТА ПАЦИЕНТОВ МЕДИЦИНСКОГО ЦЕНТРА

ВЫПОЛНИЛ: ПРОНЬКИН А.А.
РУКОВОДИТЕЛЬ: ФИНОГЕЕВ А.Г.

Задачи и цели работы

Цель работы – разработка интеллектуальной экспертной системы, которая будет автоматизировать процесс диагностики и учета пациентов в медицинском центре.

Для достижения цели необходимо выполнение следующих задач:

- Анализ предметной области,
- Обзор аналогов разрабатываемой системы,
- Выбор платформы
- Выбор языка программирования
- Обучение модели
- Проектирование системы
- Разработка системы

Актуальность работы

Экспертные системы с использованием искусственного интеллекта повышают точность и эффективность лечения. Поэтому разработка таких систем является актуальной задачей для улучшения качества медицинского обслуживания.

3

Обзор аналогов



Логотип системы IBM Watson



Логотип системы CADUCEUS

4

Обзор аналогов



Логотип системы EPIC

5

Достоинства и недостатки аналогов

Основные преимущества рассмотренных аналогов:

- 1) Высокая точность диагностики и лечения.
- 2) Автоматизация анализа клинических данных.
- 3) Поддержка принятия обоснованных решений врачами.

Недостатки:

- 1) Сложность использования для медицинского персонала.
- 2) Требование специальных навыков.
- 3) Необходимость больших вычислительных ресурсов и доступа к данным.

6

Обучение модели

Опорные векторы (SVM) - это алгоритм машинного обучения, который находит оптимальную гиперплоскость для разделения данных различных классов в пространстве признаков.

SVM имеет преимущества:

- 1) Хорошую обобщающую способность;
- 2) Эффективность в пространствах высокой размерности;
- 3) Возможность работы с различными типами данных;

Недостатки:

- 1) Сложность выбора и настройки ядра;
- 2) Чувствительность к шуму;
- 3) Длительное время обучения на больших наборах данных;

7

Основные заболевания, которые способна распознать модель

- | | | |
|---|--------------------------|-------------------------------------|
| 1. Грибковая инфекция; | 15. Желтуха; | 29. Диморфные геморроидальные узлы; |
| 2. Аллергия; | 16. Малярия; | 30. Сердечный приступ; |
| 3. ГЭРБ; | 17. Ветряная оспа; | 31. Варикозное расширение вен; |
| 4. Хронический холестаз; | 18. Лихорадка Денге; | 32. Гипотиреоз ; |
| 5. Лекарственная реакция; | 19. Брюшной тиф; | 33. Гипертиреоз; |
| 6. Язвенная болезнь желудка и двенадцатиперстной кишки; | 20. Гепатит А; | 34. Гипогликемия; |
| 7. СПИД; | 21. Гепатит В; | 35. Остеоартрит; |
| 8. Сахарный диабет; | 22. Гепатит С; | 36. Артрит; |
| 9. Гастроэнтерит; | 23. Гепатит D; | 37. (Головокружение) парампульсное |
| 10. Бронхиальная астма; | 24. Гепатит E; | позиционное головокружение; |
| 11. Гипертония; | 25. Алкогольный гепатит; | 38. Угревая сыпь; |
| 12. Мигрень; | 26. Туберкулез; | 39. Инфекция мочевыводящих путей; |
| 13. Шейный спондилез; | 27. Обычная простуда; | 40. Псориаз; |
| 14. Паралич (кровоизлияние в мозг); | 28. Пневмония; | 41. Импетиго; |

8

Симптомы:

- | | | |
|---------------------------------|--|---------------------------|
| 1. Зуд; | 15. Усталость; | 29. Потливость; |
| 2. Кожная сыпь; | 16. Прибавка в весе; | 30. Обезвоживание; |
| 3. Узловые образования на коже; | 17. Беспокойство; | 31. Расстройство желудка; |
| 4. Постоянный похмель; | 18. Холодные руки и ноги; | 32. Головная боль; |
| 5. Дрожь; | 19. Перепады настроения; | 33. Желтоватая кожа; |
| 6. Озноб; | 20. Потеря веса; | 34. Темная моча; |
| 7. Боль в суставах; | 21. Беспокойство; | 35. Тошнота; |
| 8. Боль в желудке; | 22. Вялость; | 36. Потеря аппетита; |
| 9. Повышенная кислотность; | 23. Першение в горле; | 37. Боль в глазах; |
| 10. Язвы на языке; | 24. Повышенный уровень сахара в крови; | 38. Боль в спине; |
| 11. Мышечная слабость; | 25. Кашель; | 39. Запор; |
| 12. Рвота; | 26. Высокая температура; | 40. Абдоминальная боль; |
| 13. Жжение на лице; | 27. Потемневшие глаза; | 41. Диарея; |
| 14. Мажущее мочеиспускание; | 28. Одышка; | 42. Легкая лихорадка; |

9

Симптомы:

- | | | | |
|-----------------------------------|-------------------------------------|---------------------------|-------------------------------|
| 43. Желтая моча; | 57. Боль в груди; | 71. Опухшие лицо и глаза; | 81. Мышечная слабость; |
| 44. Пожелтение глаз; | 58. Слабость в конечностях; | 72. Увеличенные | 82. Затекшая шея; |
| 45. Острый недостаток жидкости; | 59. Учащенное сердцебиение; | щитовидные железы; | 83. Вздутие суставов; |
| 46. Перегрузка жидкостью; | 60. Боль во время движения мышцами; | 73. Ломкие ногти; | 84. Жесткость движений; |
| 47. Вздутие желудка; | 61. Боль в анальной области; | 74. Опухлости | 85. Вращательные движения; |
| 48. Опухание лимфатических узлов; | 62. Кровянистый стул; | экстремальные; | 86. Потеря равновесия; |
| 49. Недомогание; | 63. Раздражение в анусе; | 75. Чрезмерный желтый; | 87. Неустойчивость; |
| 50. Нечеткое и искаженное зрение; | 64. Боль в шее; | 76. Экстра-маритальные | 88. Слабость одной стороны |
| 51. Мокрота; | 65. Головокружение; | контакты; | тела; |
| 52. Першение в гортани; | 66. Судороги; | 77. Сухие и двигающие | 89. Потеря запаха; |
| 53. Покраснение глаз; | 67. Кровоподтеки; | жимы; | 90. Дискомфорт в мочевом |
| 54. Синусовое давление; | 68. Ожирение; | 78. Невнятная речь; | пузыре; |
| 55. Насморк; | 69. Опухшие ноги; | 79. Боль в коленях; | 91. Неприятный запах мочи; |
| 56. Застойные явления; | 70. Опухшие кровеносные сосуды; | 80. Боль в тазобедренных | 92. Постоянное ощущение мочи; |
| | | суставах; | |

10

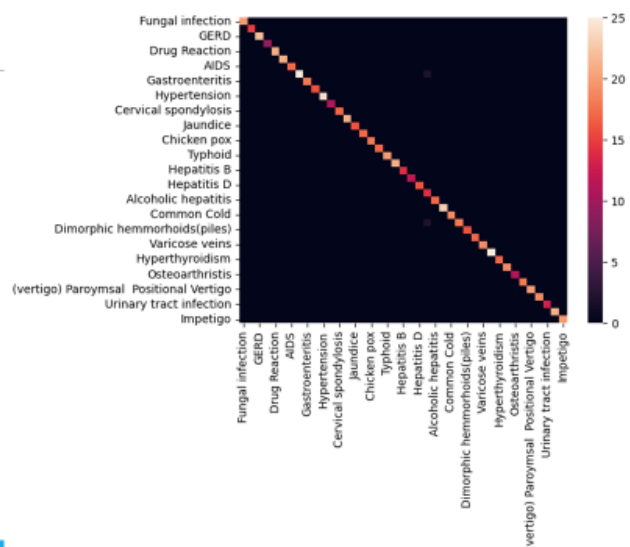
Симптомы:

93. Выделение газов;	107. Семейная история;	121. Болезненная ходьба;
94. Внутренний зуд;	108. Слизистая оболочка;	122. Гнойные прыщи;
95. Токсический вид (брюшной тиф);	109. Ржавая оболочка;	123. Угри;
96. Депрессия;	110. Недостаточная концентрация;	124. Зуд;
97. Раздражительность;	111. Визуальные нарушения;	125. Шелушение кожи;
98. Мышечные боли;	112. Получение крови трансфузии;	126. Серебристая пылеватость;
99. Измененная сенсорика;	113. Получение стерильных инъекций;	127. Маленькие зубцы на ногтях;
100. Красные пятна на теле;	114. Кома;	128. Воспалющиеся ногти;
101. Боль в животе;	115. Желудочное кровотечение;	129. Волдырь;
102. Ненормальная менструация;	116. Вздутие живота;	130. Покраснение вокруг носа;
103. Дисхромные пятна;	117. История употребления алкоголя;	131. Желтая корочка;
104. Слезотечение;	118. Кровь в мокроте;	
105. Повышенный аппетит;	119. Выступающие желудки на икре;	
106. Полиурия;	120. Учащенное сердцебиение;	

11

Матрица ошибок

Для визуализации точности работы модели была построена матрица ошибок или матрица смещения.



12

Выбор средств разработки

В качестве языка программирования был выбран Python, ключевой особенностью которого является большое количество встраиваемых библиотек, что значительно ускоряет разработку ПО.



Python

13

Выбор средств разработки

Для реализации клиентской части веб-приложения был выбран фреймворк React на языке программирования JavaScript. React позволяет создавать динамичные пользовательские интерфейсы из отдельных компонентов, что способствует повышению модульности и переиспользованию кода.



React

14

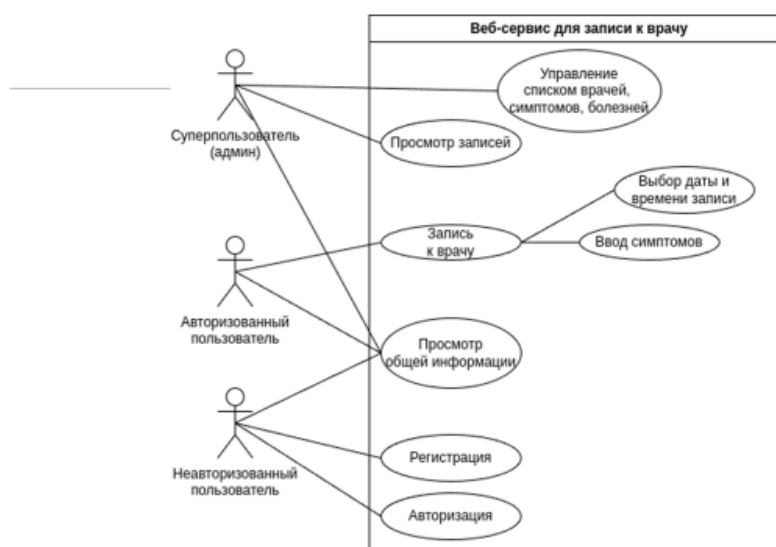
Выбор средств разработки

Для серверной части веб-приложения используется фреймворк Django, известный своей гибкостью и высокой производительностью. В паре с Django REST framework, который позволяет быстро создавать RESTful API, данный фреймворк обеспечивает надежное и масштабируемое серверное приложение.



15

Схема процесса записи к врачу



16

Видеоработоспособность программы.

17

Заключение

В результате выпускной квалификационной работы разработана интеллектуальная экспертная система диагностики и учета пациентов медицинского.

Система включает в себя следующие модули:

1. Модуль ввода данных и описания симптомов болезни;
2. Модуль распознавания симптомов;
3. Модуль классификации вероятных симптомов по заболеванию;
4. Модуль предварительного диагностирования с определением вероятного заболевания;
5. Модуль определения требуемых врачей;
6. Модуль рекомендательного выбора конкретного специалиста;
7. Модуль записи к врачу;
8. Модуль формирования медицинской карты пациента с перечнем симптомов, вероятных заболеваний и рекомендованных врачей;
9. База данных пациентов;
10. Модуль управления базой данных;

18

Спасибо за внимание!