

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

МЕДИЦИНСКОЕ ПРИЛОЖЕНИЕ.
ЧАТ-БОТ ДЛЯ ВЗАИМОДЕЙСТВИЯ С
ПОЛЬЗОВАТЕЛЕМ

Студент

Кор 13.06.24
(подпись, дата)

Коротов Владислав Олегович
(ФИО полностью)

Руководитель

13.06.24
(подпись, дата)

Митрохин М.А.
(фамилия, инициалы)

Нормоконтролёр

13.06.24
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

каф. отдела
(должность, место работы)

13.06.2024
(подпись, дата)

Севастьянов А.В.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 № 11)

Заведующий кафедрой

(подпись)
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

14.06.2024 № 23)
Секретарь ГЭК (подпись)
(подпись)

Е.И. Гурин
(фамилия, инициалы)

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 » сентября 2022 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Коротов Владислав Олегович гр. 22ВВИм1 факультета вычислительной техники
2. Тема работы «Медицинское приложение. Чат-бот для взаимодействия с пользователем»
Тема утверждена приказом ПГУ №765/о от "13" сентября 2022 г.
3. Руководитель магистерской диссертации д.т.н., профессор Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать чат-бот для взаимодействия с пользователем при помощи текстовых сообщений. Система должна выполнять комплексный анализ текста и выдавать рекомендации пользователю.

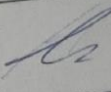
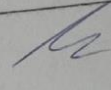
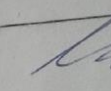
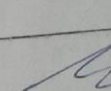
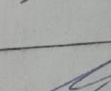
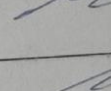
В качестве клиента необходимо разработать мобильное приложение.

Должно быть осуществлено взаимодействие с подсистемами «Анализ изображения по фотографии» и «Анализ настроения по тексту».

5. Перечень подлежащих разработке вопросов:

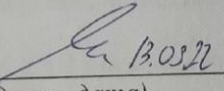
1. Теоретический анализ предметной области использования виртуальных помощников в медицинской сфере.
2. Обзор и сравнительный анализ необходимых средств и инструментов для разработки системы чат-бота.
3. Обучение нейронной сети для генерации текста, тестирование. Разработка и развертывания сервера. Создание мобильного приложения.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Теоретический анализ предметной области	10	21.09.2022 – 21.11.2022	
2.	Обзор и сравнительный анализ средств для разработки	10	22.11.2022 – 22.01.2023	
3.	Разработка системы обмена сообщениями	20	23.01.2023 – 23.04.2023	
4.	Разработка нейронной сети и развертывание сервера	30	24.04.2023 – 29.01.2024	
5.	Разработка клиентской части	20	30.01.2024 – 30.03.2024	
6.	Оформление выпускной работы	10	31.03.2024 – 31.05.2024	

Дата выдачи задания " 13 " сентября 2022 г.

Руководитель магистерской диссертации


(подпись, дата)

Митрохин М.А.

Задание к исполнению принял студент


(подпись, дата)

Коротов В.О.

Работу к защите допустить
Декан факультета


(подпись, дата)

13.06.2024

Фионова Л.Р.

РЕФЕРАТ

Пояснительная записка содержит 81 лист, 36 рисунков, 5 таблиц, 12 источников, 2 приложения

Медицина, диагностика заболеваний, NODEJS, TYPESCRIPT, PYTHON, DOCKER, DOCKER-COMPOSE, GIT, GITHUB, СУБД, NOSQL, MONGODB, FLUTTER

Целью проекта является разработка чат-бота, позволяющего выполнять диагностику заболевания пользователя. Это достигается путем анализа сообщений, которые пользователь отправляет на сервер. Система, может проанализировать текст сообщения либо изображение, отправленное пользователем, чтобы сформировать конечный диагноз. С помощью такой системы можно уменьшить количество поздно обнаруженных диагнозов, ускорить время выздоровления человека и потенциально сохранить ему жизнь.

В результате проделанной работы разработаны сервер и модель нейронной сети, позволяющая выполнять анализ текста от пользователя. Также реализовано взаимодействие с разработанной подсистемы «Анализ изображения по фотографии» и «Анализ настроения».

Было успешно выполнено развертывание серверного приложения. Для избежания проблем с совместимостью был использован Docker и Docker-compose. В качестве сервиса для хостинга репозитория был использован Github. После проделанных манипуляций сервер был полностью подготовлен для клиентских приложений.

Разработано и протестировано клиентское мобильное приложение. Реализована интеграция с сервисом питания, система обмена сообщениями, виртуальный аватар человека.

					ПГУ 1.09.04.01.11.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата	Медицинское приложение. Чат-бот для взаимодействия с пользователем	Лит.	Лист	Листов
Разраб.		Коротов В.О.		13.06.24				
Провер.		Митрохин М.А.		13.06.24	Пояснительная записка		2	81
Н. Контр.		Бычков А.С.		13.06.24		ФВТ, каф. ВТ гр. 22ВВВим1		
Утв.		Митрохин М.А.		13.06.24				

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Теоретический анализ предметной области	7
1.1 Анализ темы	7
1.2 Анализ аналогов	8
1.3 Постановка задачи	16
1.4 Выбор архитектуры приложения	16
1.5 Общая структура проекта	17
1.6 Разработка <i>UML</i> -диаграмм	19
1.7 Выбор инструментов разработки	20
1.7.1 Сервер	21
1.7.2 Мобильное приложение	19
1.7.3 Языки программирования	26
1.7.4 Выбор СУБД	27
1.7.5 Выбор среды программирования	28
1.8 Выводы	32
2 Разработка приложения	30
2.1 Разработка сервера	30
2.1.1 Авторизация и аутентификация	30
2.1.2 JWT-токены	35
2.1.3 База данных и взаимодействие с ней	36
2.1.4 Система обмена сообщениями	38
2.1.5 Система чат-ботов	38
2.1.6 Анализ текста	39
2.1.7 Подсистема «Анализ настроения»	39
2.1.8 Подсистема «Анализ изображения по фотографии»	39
2.1.9 Документация к проекту	40
2.2 Разработка нейронной сети	41
2.2.1 Hugging Face	42

2.2.2 Набор данных	42
2.2.3 Модель FALCON	39
2.2.4 Fine-tuning модели	40
2.3 Развертывание серверного приложения	46
2.3.1 Контейнеризация приложения	47
2.3.2 Docker	47
2.3.3 Docker-compose	49
2.3.4 GIT/GITHUB и разворачивание приложения	51
2.3.5 Автоматическое развертывание приложения	51
2.4 Выводы	49
3 Клиентское приложение	55
3.1 Авторизация и аутентификация	55
3.2 Основные страницы в приложении	56
3.3 Виртуальный аватар	66
3.4 Сервисы	68
3.5 Мобильное приложение для анализа и корректировки питания	69
3.6 Вкладка «Сервисы»	69
3.7 Выводы	70
ЗАКЛЮЧЕНИЕ	75
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	77
ПРИЛОЖЕНИЕ А Листинг программы	79
ПРИЛОЖЕНИЕ Б Презентация	79

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

ПО – программное обеспечение

AI (Artificial Intelligence) – искусственный интеллект

API (Application Programming Interface) – программный интерфейс приложения

JSON (Java Script Object Notation) – формат данных, основанный на объектах в языке программирования java script

ВВЕДЕНИЕ

Каждый год большое количество людей страдают от поздно поставленного медицинского диагноза. Происходит это по разным причинам: нехватка времени или денег на полное обследование, халатное отношение к своему здоровью, ошибки врачей, в результате которых они не могут определить симптомы и выявить болезнь вовремя. В результате это может сильно сказаться на состоянии организма человека, продолжительности его лечения, а в некоторых ситуациях и вовсе привести к серьезным проблемам со здоровьем и смерти.

В магистерской диссертации объектом исследования является чат-бот для анализа состояния здоровья и выдачи рекомендаций пользователю на основе отправленного сообщения. Система чат-бота основана на технологиях искусственного интеллекта. Виртуальный помощник должен анализировать текст с помощью нейронных сетей и выдавать пользователю рекомендации по решению его проблемы. Предметом исследования выступает чат-бот для взаимодействия с пользователем.

Цель магистерской диссертации является проектирование архитектуры, выявление и анализ требуемого функционала приложения, и разработка комплексного ПО, основанного на технологиях искусственного интеллекта.

Задачи исследования:

- взаимодействие между чат-ботом и пользователем при помощи системы обмена сообщений,
- выдача рекомендации по решению проблемы пользователя.

1 Теоретический анализ предметной области

1.1 Анализ темы

Для контроля за собственным здоровьем каждый человек должен как минимум один раз в два года проходить медицинское обследование. Из-за нехватки времени, некоторые люди не могут выполнять такую рекомендацию, а некоторые и вовсе боятся обращаться к врачам. Также некоторые люди из-за больших очередей на различные медицинские процедуры (УЗИ, компьютерная томография и др.) не могут пройти их вовремя. Все это в итоге может привести к несвоевременной диагностике заболеваний, сильно повлиять на здоровье человека, а в худшем случае привести к летальному исходу. Также поздняя диагностика может происходить из-за низкой квалификации отдельных медицинских работников.

Сейчас большинство людей имеют доступ к интернету, поэтому было решено разработать систему, которая позволяла бы людям без лишних проблем диагностировать возможные заболевания. Виртуальный помощник смог бы в любое время ответить на вопросы человека, проанализировать его симптомы, выявить возможные заболевания и дать рекомендации по решению сложившейся ситуации. Основываясь на этих результатах, можно порекомендовать человеку обратиться к специалисту, который бы смог ему помочь.

Медицинский виртуальный помощник, основанный на технологиях искусственного интеллекта, может проводить первичную диагностику, выявлять симптомы болезней, давать рекомендации, подбирать подходящих врачей, выдавать информацию о лекарственных препаратах и многое другое. С помощью чат-ботов можно значительно уменьшить нагрузку на систему здравоохранения, проводить консультации, а в случае взаимодействия с различными клиниками осуществлять запись человека к доктору, отправлять симптомы и жалобы пациента напрямую лечащему врачу перед приемом, что помогло бы увеличить шанс на точное определение болезни. Также такие

виртуальные помощники могут напоминать человеку о приеме лекарств в нужное время.

Для получения большего количества данных о состоянии здоровья человека, можно использовать различные сервисы, связанные со здоровьем, которыми пользуется человек. Сюда же можно отнести и различные умные устройства, собирающие и анализирующие данные о человеке, например, умные часы или весы.

Большинство людей сейчас имеют собственный смартфон, поэтому было решено разработать мобильное приложение, которое позволило бы пользователям взаимодействовать с вышеописанным виртуальным помощником посредством отправки ему сообщений. Система с чат-ботом может использоваться и в других типах приложений, например, веб-приложении или программе, которая непосредственно устанавливается на компьютер пользователя.

1.2 Анализ аналогов

Проанализировав рынок мобильных приложений, было найдено несколько похожих систем, позволяющих осуществлять контроль за здоровьем, рассмотрим их по отдельности.

Ada Health помогает человеку определить возможные болезни и дает дальнейшие рекомендации для решения проблемы. Диагностика заболевания происходит с помощью опроса, в котором происходит уточнение симптомов пользователя. Система задает вопрос, а человек должен выбрать из нескольких возможных вариантов. После прохождения теста система формирует отчет, в котором указаны возможные заболевания. Также присутствует открытая база знаний, в которой человек может больше узнать про различные симптомы и диагнозы. Каждый диагноз подробно описан, присутствуют симптомы заболевания, описание угрозы для жизни человека. Ada Health распространяется в виде мобильных приложений на Android и iOS. Проект является бесплатным и не содержит рекламы, что положительно влияет на

пользовательский опыт. Для того, чтобы пользоваться приложением, необходима только регистрация. Также приложение имеет визуально понятный и удобный минималистичный интерфейс. Из минусов можно отметить неудобство прохождения опроса – пользователь должен выбирать из списка симптомов те, которые присутствуют у него и в дальнейшем отвечать на вопросы, которые задает система. Сами опросы довольно длительные и могут занимать большое количество времени. Приложение поддерживает 7 языков – английский, немецкий, испанский, португальский, суахили, румынский и французский. Демонстрация интерфейса приложения показана ниже на рисунке 1.

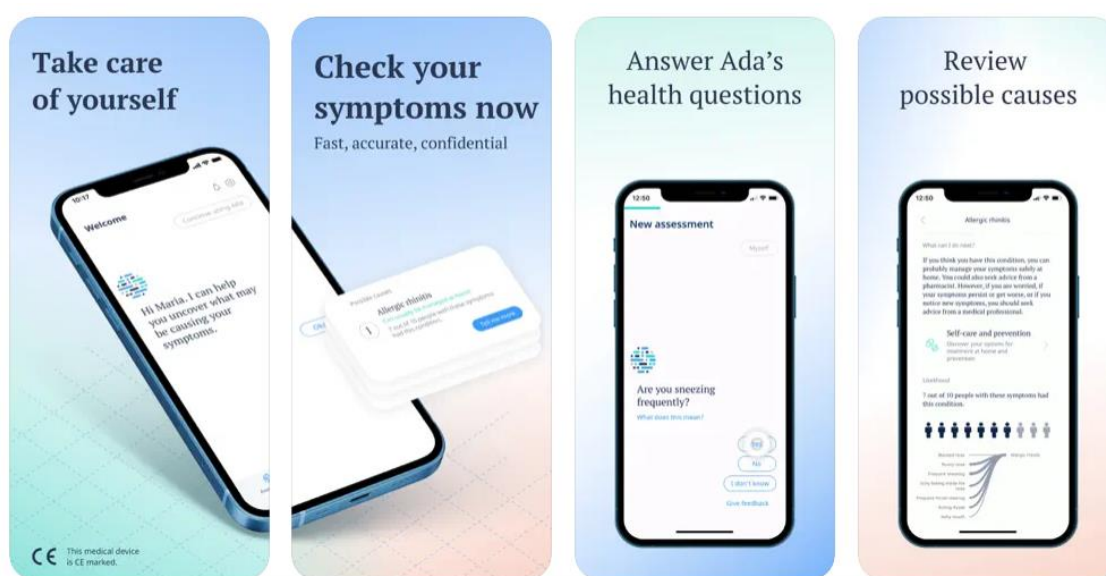


Рисунок 1 – Интерфейс приложения Ada Health

Symptomate – проект, нацеленный на диагностику заболевания человека. Как и в предыдущем рассматриваемом приложении, диагностика происходит в виде опроса. Пользователь отвечает на уточняющие вопросы и получает список возможных диагнозов и вероятность их наличия. Взаимодействие с приложением возможно без регистрации, все данные пользователя надежно защищены и не передаются третьим лицам. В базе имеется более 740 болезней и более 1300 симптомов, а сама система проверена врачами и другими специалистами. Существует веб-версия приложения, а также мобильные

версии, доступные на Android и iOS. Из минусов можно отметить длительное прохождение опроса. Интерфейс приложения представлен ниже на рисунке 2.

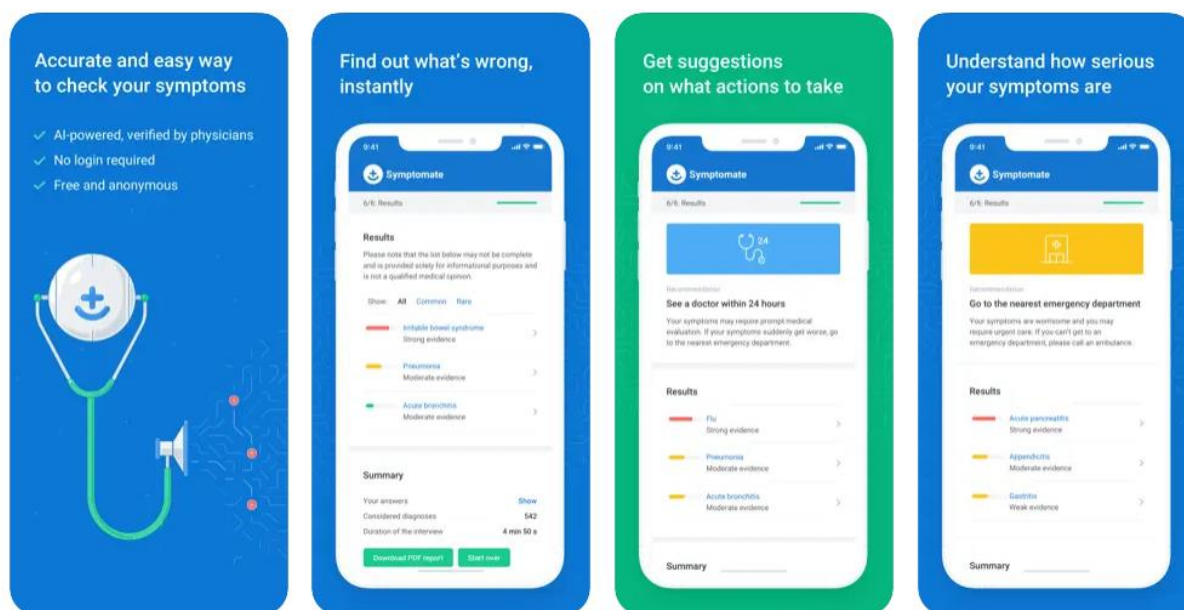


Рисунок 2 – Интерфейс приложения Symptomate

Woebot Health – приложение, который помогает людям справляться со стрессовыми ситуациями и депрессией. Во время общения пользователю дают различные советы о том, как поднять настроение, улучшить свое самочувствие и самооценку. Общение происходит в формате переписки. Приложение доступно на мобильных устройствах – Android, iOS. Имеет бесплатный пробный период. Интерфейс представлен ниже на рисунке 3.

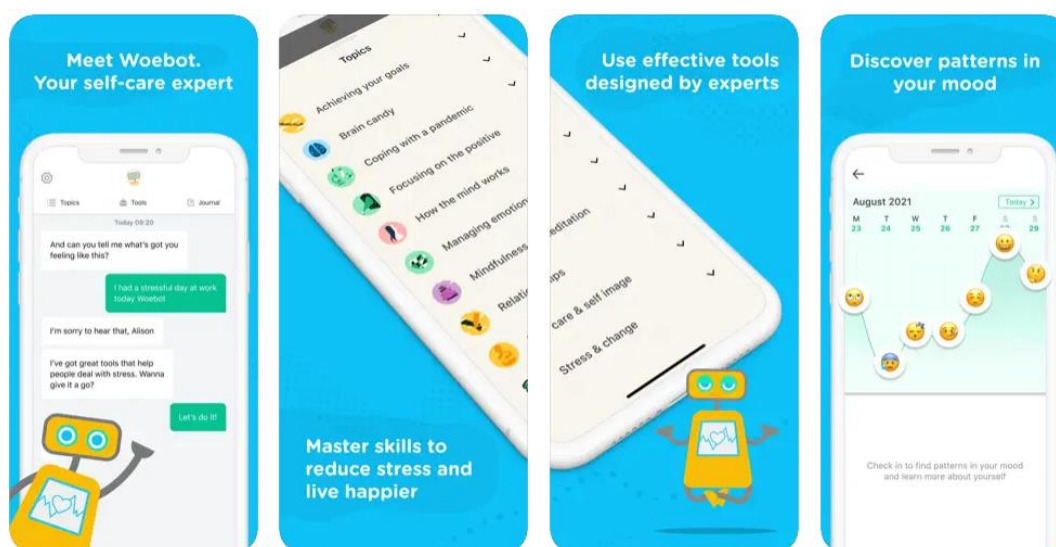


Рисунок 3 – Интерфейс приложения Woebot Health

Сравнение функционала рассмотренных приложений представлено в таблице 1.

Таблица 1 – Результат сравнительного анализа приложений

Функция	Название приложения		
	Ada Health	Symptomate	Woebot Health
Бесплатное использование	+	+	-
Первичная диагностика заболеваний	+	+	-
Общение с чат-ботом	-	-	+
Отслеживание самочувствия	-	-	+
Открытая база знаний	+	-	-
Интуитивный интерфейс	+	+	+
Напоминание о приеме лекарств	-	-	-

В ходе анализа было выявлено, что многие приложения выполняют диагностику заболеваний с помощью опроса, что увеличивает количество затраченного времени на то, чтобы узнать возможные болезни человека. Woebot Health позволяет напрямую общаться с ботом, однако это приложение является специализированным и не предназначено для диагностики. Также большинство приложений не обладают дополнительным функционалом, например, напоминанием о приеме лекарственных препаратов. Из этого вытекает необходимость разработки приложения, которое покрыло бы все эти недостатки и реализовало недостающие функции для контроля за здоровьем человека.

1.3 Постановка задачи

Требуется разработать комплекс программного обеспечения, состоящий из мобильного приложения, сервера и нейронной сети. Клиентское приложение должно обладать следующими функциями:

- есть регистрация и вход в систему,
- наличие профиля, в котором пользователь сможет указать различные данные о себе для более точной диагностики,
- взаимодействие с чат-ботом должно осуществляться посредством переписки с ним,
- есть возможность отправлять чат-боту текстовые данные и изображения для анализа,
- простой и пользовательский интерфейс,
- взаимодействие в режиме реального времени,
- каждый пользователь должен иметь возможность удалить всю хранящуюся информацию о нем с сервера,
- взаимодействие с подсистемами для анализа настроения и изображений.

Серверное приложение должно быть использовано для обмена информацией с клиентами. На сервере будут храниться нейронные сети, которые будут осуществлять анализ сообщений от пользователя. Также сервер будет взаимодействовать с базой данных, для сохранения истории переписок пользователей.

Чтобы улучшить пользовательский опыт, сервер должен максимально быстро обрабатывать данные, которые ему будут отправлять клиенты и формировать ответ.

1.4 Выбор архитектуры приложения

В качестве архитектуры была выбран клиент-серверный подход к разработке. Он позволяет эффективно управлять ресурсами и обрабатывать запросы от пользователей, не тратя при этом мощности клиентских машин. Он

также позволяет обрабатывать приходящие запросы, не используя при этом ресурсы другой машины. Такой подход к разработке позволит выполнять высоконагруженные вычисления на сервере, не тратя при этом ресурсы клиента. На сервере будут храниться нейронные сети, которые могут потребовать для выполнения и обработки данных графических ускорителей, а у клиента таких ресурсов может не оказаться.

Клиентская программа предназначена для пользователя и будет общаться с сервером посредством отправки запросов на него.

Сервер будет выполнять прием запросов от клиента, а также обработку сообщений, которые приходят на сокет для обеспечения взаимодействия в режиме реального времени.

Для решения поставленной задачи потребуется нейронная сеть для обработки текста. Такая нейронная сеть может занимать достаточно большое количество памяти на устройстве, а также требовать серьезные вычислительные мощности для выполнения обработки, поэтому она должна храниться на сервере. История переписок будет храниться в базе данных, которая также будет на сервере.

Клиентом будет выступать мобильное приложение, с которого будут отправляться все необходимые запросы на сервер, а также будет осуществляться взаимодействие с сокетами для обеспечения общения в режиме реального времени. После выполнения всех необходимых вычислений сервер будет отправлять сообщение на сокет. Дальнейшая обработка и отображение данных будет происходить на клиентской части.

1.5 Общая структура проекта

Сервер будет обрабатывать запросы, приходящие от пользователя, выполнять операции с базой данных `mongodb`, на сервере будут храниться модели нейронных сетей, необходимых для диагностики и получения рекомендаций для пользователя. Также сервер будет выполнять роль

хранилища различных файлов, которые будет отправлять пользователь (например, изображения).

В качестве способа взаимодействия между участниками был выбран REST API для получения необходимых данных от сервера. REST API – архитектура взаимодействия для приложения с помощью протокола HTTP. Данная архитектура очень популярна для построения клиент-серверных приложений. Данные, которые будет получать клиент, будут в формате JSON. Этот формат представляет данные в виде пар ключ: значение.

Серверная часть приложения будет написана с помощью технологий NodeJS и с применением Typescript в качестве языка программирования.

Общая схема взаимодействия между компонентами представлена на рисунке 4 ниже.

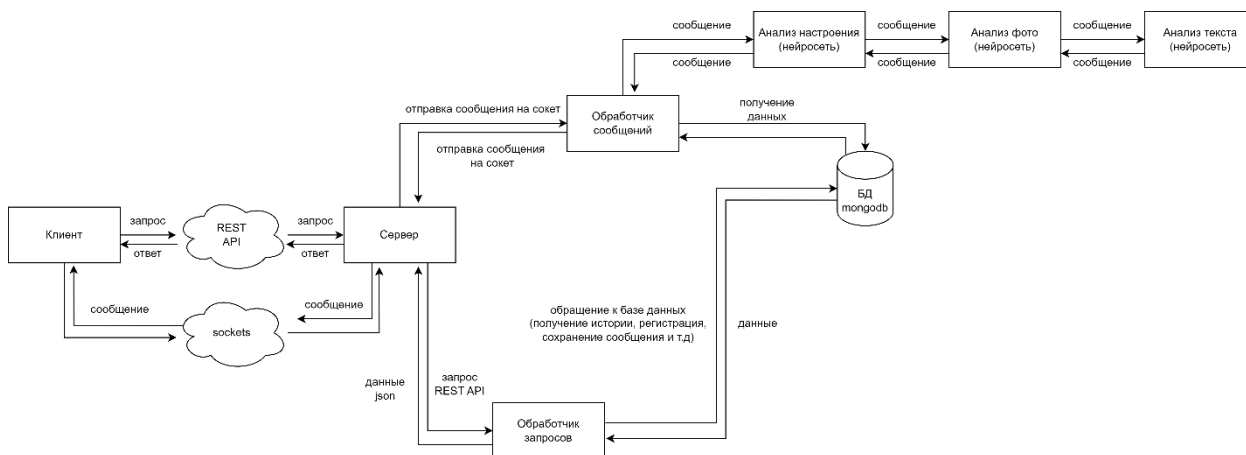


Рисунок 4 – Общая схема взаимодействия

Клиент может отправлять запросы и сообщения на сокет на сервер. На сервере имеется для этого отдельные обработчики. При получении обычного запроса (GET/POST/PUT/PATCH/DELETE), сервер выполняет обращение к базе данных, формирует ответное сообщение в формате JSON и затем отправляет его на клиент, где и происходит дальнейшая обработка.

В случае, если сервером было получено сообщение на сокет и было адресовано боту, необходимо обработать его с помощью нейронных сетей, передавая полученный результат на каждом шаге в следующую модель. Завершает цепочку из нейронных сетей модуль для обработки текста, который

и будет формировать ответное сообщение для клиента. Тут же выполняется взаимодействие с подсистемами «Анализ настроения» и «Анализ изображения». После всех проделанных действий, сообщение будет отправлено на сокет. Клиент должен отреагировать на получение сообщения и выполнить изменение интерфейса.

1.6 Разработка UML-диаграмм

Исходя из выбранной архитектуры приложения была спроектирована диаграмма развертывания, которая показывает связь клиентов и сервера. Диаграмма представлена на рисунке 5 ниже.

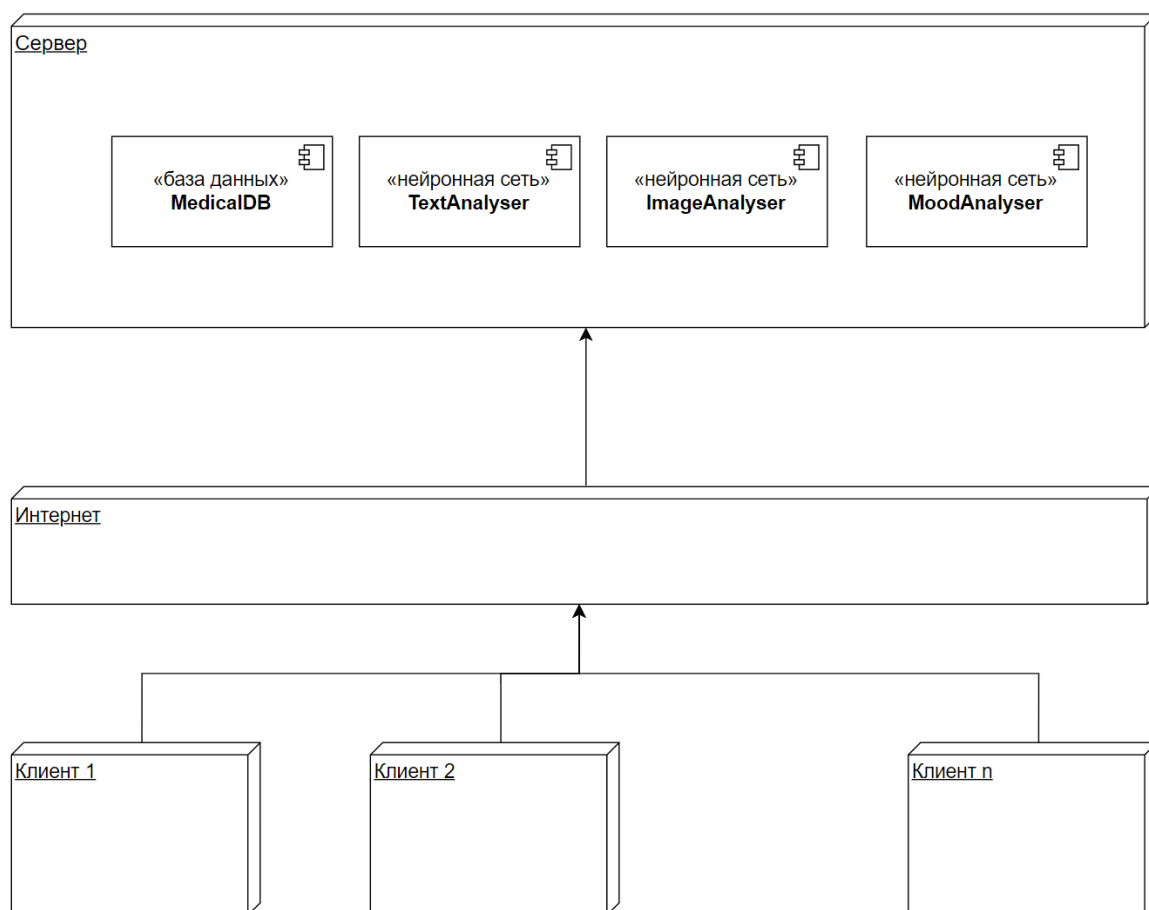


Рисунок 5 – Диаграмма развертывания

Далее была разработана диаграмма прецедентов для определения функционала клиента. После регистрации пользователь получает доступ к различным функциям приложения: взаимодействие с чат-ботом, запрос на

удаление данных с сервера, изменение данных в профиле, взаимодействие с сервисами, просмотр виртуального аватара. Диаграмма показана на рисунке 6 ниже.

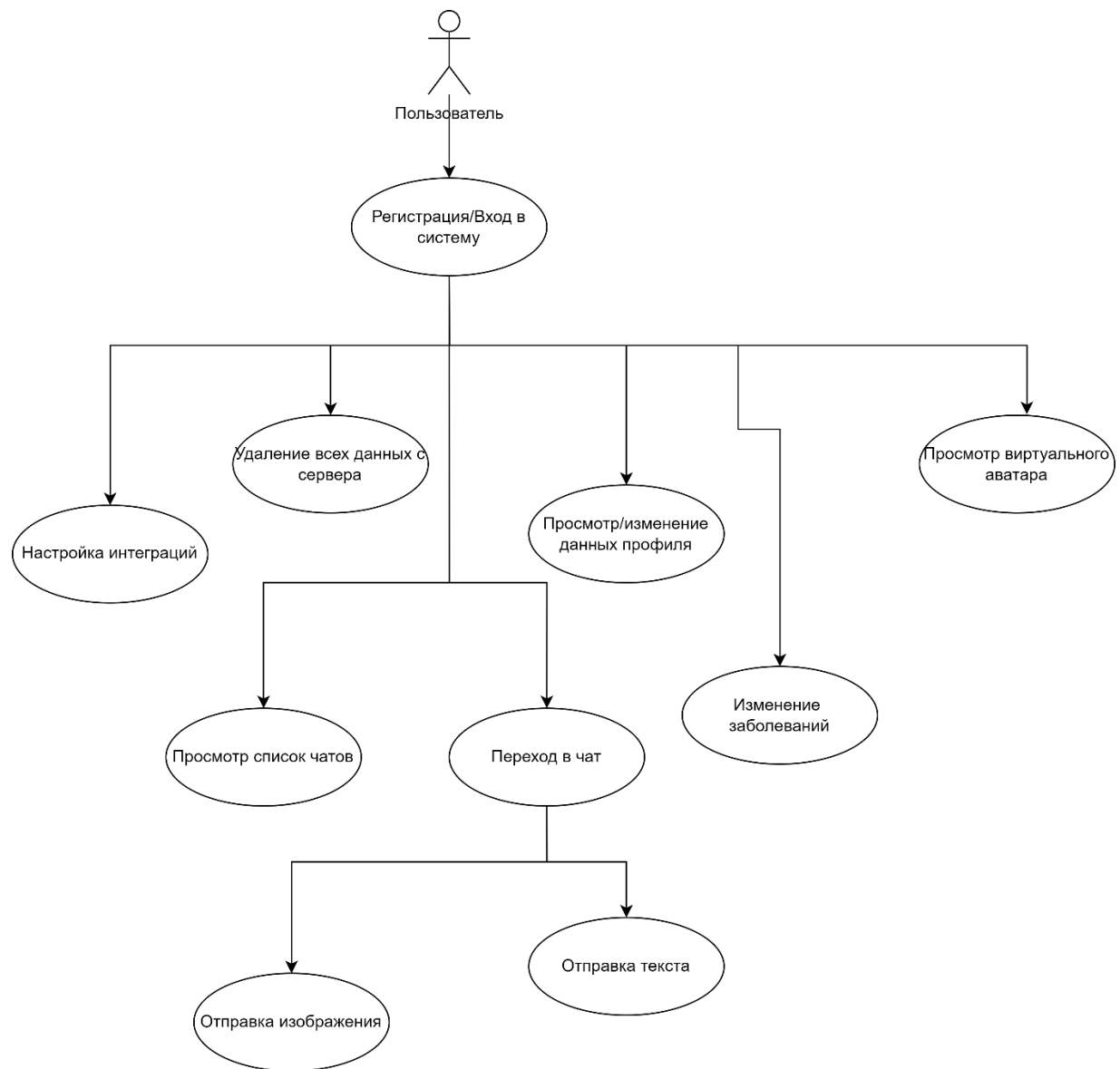


Рисунок 6 – Диаграмма прецедентов

1.7 Выбор инструментов разработки

Перед началом разработки системы необходимо определиться, с помощью каких технологий она будет создаваться. К разработке подлежит мобильное приложение, сервер и нейронная сеть. Рассмотрим каждый составляющий элемент по отдельности.

1.7.1 Сервер

В качестве технологий для серверной части будут рассмотрены следующие фреймворки: NodeJS, Django, Spring Boot.

NodeJS – платформа, разрабатываемая на основе движке V8 от компании Google. NodeJS позволяет разрабатывать серверную часть для приложений на языке JavaScript [1]. Это может быть очень полезным, если и веб-приложение написано на этом языке. В данном случае это является плюсом, так как веб-приложение будет разрабатываться с помощью React на языке JavaScript. Из особенностей NodeJS можно выделить следующие:

- высокая производительность благодаря наличию системы ввода-вывода, которая не блокирует основной поток,
- в случае написания веб-приложение возможно повторное использование кода,
- большое сообщество разработчиков. Если возникнет проблема, можно легко найти ответ на вопрос или готовое решение.

К минусам NodeJS можно отнести невысокую производительность для тяжелых задач.

Django – фреймворк, позволяющий разрабатывать серверную часть на языке программирования Python. Особенности данного фреймворка:

- высокая производительность,
- простота использования,
- есть инструменты, обеспечивающие высокую безопасность продукта,
- большое сообщество.

Минусы:

- не поддерживается технология WebSocket по умолчанию,
- наличие Django ORM, уступающей другим,
- медленный темп развития фреймворка.

Spring Boot – фреймворк для разработки серверной части приложения на языке Java. Особенности данного фреймворка:

- простая настройка и создание REST API,
- нет xml-конфигурации,
- имеет богатый функционал,
- большое количество ресурсов для изучения.

Из минусов можно отметить большое время для освоения фреймворка.

Сравнение различных технологий для создания серверного приложения представлено в таблице 2.

Таблица 2 – Сравнение технологий для серверной части

Среда	Достоинства	Недостатки
NodeJS	Основан на движке V8 от компании Google, который постоянно развивается и улучшается. Обладает высокой производительностью, не блокирует основной поток приложения. Высокая популярность и большое сообщество разработчиков. Возможно повторное использование частей кода.	Не подходит для тяжелых вычислительных задач, требующих большое количество ресурсов центрального процессора.
Django	Высокая производительность и простота использования технологии. Наличие инструментов для обеспечения безопасности продукта. Есть большое сообщество разработчиков.	Нет поддержки WebSocket напрямую. Использование DjangoORM, которая уступает другим ORM. Медленный темп развития технологии.
Spring Boot	Простая настройка и создание REST API. Отсутствие конфигурации xml. Гибкая технология, обладающая широким функционалом. В сети есть большое количество ресурсов по изучению данной технологии.	На изучение фреймворка может потребоваться большое количество времени.

Исходя из всех преимуществ и недостатков каждого из фреймворков, а также из постановки задачи на разработку приложения, было принято решение создавать серверную часть проекта на NodeJS.

1.7.2 Мобильное приложение

Для того, чтобы охватить как можно большее число пользователей, было принято решение о разработке мобильного приложения, доступного сразу на нескольких платформах. Сейчас на рынке мобильных операционных систем есть два лидера – Android и iOS. По данным на 2022 год доля Android занимает около 71%, для iOS эта цифра составляет 28%. Оставшуюся долю занимают менее популярные операционные системы, такие, как например Windows Phone или KaiOS. Было принято решение разрабатывать проект с поддержкой двух наиболее популярных мобильных операционных систем на рынке – Android и iOS.

К разработке мобильного приложения можно подойти несколькими способами. Первый из них – использование нативных технологий (native). Это классический способ разработки, в этом случае создание приложения будет вестись отдельно под каждую операционную систему на разных языках программирования – Java/Kotlin для Android и Swift для iOS. Количество времени, затраченного на разработку, будет гораздо выше, так как, по сути, будет разрабатываться одно и то же приложение, но с учетом различных особенностей каждой из платформ. Если в приложении будет присутствовать одинаковый по смыслу код, но написанный на разных языках, может возникнуть потребность в будущем изменить или улучшить этот код. При таком раскладе изменения будут касаться сразу двух приложений по отдельности, что скажется на времени разработки. Аналогичная ситуация будет происходить при обнаружении ошибок в таком коде – их также придется исправлять по отдельности. Стоит учесть возможность появления платформо-специфичных ошибок для каждого приложения. Также придется учитывать все особенности каждой из платформ, следовательно количество знаний и

порог вхождения будут выше. Если же рассматривать плюсы такого подхода к разработке, то стоит отметить высокую производительность, низкий размер приложения и поддержка низкоуровневого функционала каждой платформы. Из-за того, что такой подход является самым популярным, в сети накопилось большое количество ресурсов для изучения. Также при возникновении проблем и вопросов можно быстро найти нужное решение.

Второй подход к разработке – создание кроссплатформенного приложения. При таком подходе будут использоваться технологии, позволяющие написать приложение один раз и в дальнейшем использовать его для двух операционных систем. Примером таких технологий может быть Flutter и React Native. Главным преимуществом подхода является единая кодовая база – количество времени, затраченного на разработку, будет примерно в два раза меньше, чем при нативном подходе. На первый взгляд может показаться, что количество знаний, требующихся для разработки такого приложения, будет меньше, но это не совсем так. В случае возникновения ситуаций, когда требуется написать платформу-специфичный код, присутствует возможность, позволяющая добавлять части нативного кода в приложение, а значит придется разбираться как в особенностях кроссплатформенных фреймворков, так и в особенностях нативной разработки под каждую платформу. Также в таких случаях будет происходить смешение кода – в проекте придется использовать как минимум 3 языка, а значит читабельность кода снизится. Поэтому можно сказать, что кроссплатформенная разработка является отличным выбором, когда в проекте будет минимум или не будет совсем случаев, требующих непосредственного использования аппаратных возможностей устройств и работы с каждой из операционных систем. Из недостатков можно отметить увеличение размера приложения, снижение производительности и наличие ограничений, заданных каждой из ОС или фреймворком.

Исходя из постановки задачи и создания проекта, покрывающего максимально большое число пользователей за минимальные сроки, было принято решение выбрать метод кроссплатформенной разработки.

Есть два наиболее популярных фреймворка для создания кроссплатформенных мобильных приложений – Flutter и React Native. Рассмотрим каждый по отдельности.

Flutter – фреймворк для создания кроссплатформенных приложений, разработанный и поддерживаемый компанией Google. Во Flutter для написания приложений используется язык Dart. Это объектно-ориентированный язык, созданный в качестве замены JavaScript. Язык имеет строгую статическую типизацию и простой современный синтаксис. Рендеринг во Flutter выполняется с помощью своего собственного движка – Skia [2], что обеспечивает высокую производительность. Также Flutter можно использовать не только для мобильных, но и для веб приложений, что может быть полезно в некоторых случаях. Интерфейс приложения создается с помощью декларативного подхода, что может положительно сказаться на скорости разработки, тестирования и отладки. Также во flutter есть функция hot reload, которая выполняет перезагрузку приложения в соответствии с новыми изменениями в программном коде проекта, что позволяет практически мгновенно видеть результат.

React Native – фреймворк для разработки кроссплатформенных приложений, созданный компанией Facebook в 2015 году. В качестве языка программирования приложений используется JavaScript. Это может быть очень удобно, если и веб-версия приложения написана на JavaScript, некоторые компоненты из веб-версии можно повторно использовать. С помощью него можно быстро создавать приложения, однако, имеются и ограничения. Например, некоторые приложения могут иметь низкую производительность, особенно, если они связаны с использованием графики. Также в некоторых

случаях может потребоваться написание нативного кода для конкретной платформы.

Сравнительная таблица фреймворков для создания кроссплатформенного мобильного приложения представлена в таблице 3 ниже.

Таблица 3 – Сравнение фреймворков для мобильного приложения

Программа	Достоинства	Недостатки
Flutter	Собственный движок рендеринга Skia, благодаря которому достигается высокая производительность при отрисовке сложных элементов или анимации.	Dart разработан компанией Google, которая также и поддерживает Flutter в настоящее время. Использование Flutter зависит от поддержки этой технологии компанией.
React Native	В качестве языка программирования используется JavaScript, что позволит в будущем повторно использовать код для создания веб-приложения или панели администратора.	Использования bridge для вызова нативных функций может достаточно сильно повлиять на производительность. Также может потребоваться большое количество вычислительных ресурсов, в частности, при отрисовки графических элементов внутри приложения.

Исходя из всех достоинств и недостатков каждой технологии, а также из постановки задачи на разработку, было принято решение разрабатывать мобильное приложение с помощью Flutter.

1.7.3 Языки программирования

Исходя из выбранных технологий разработки, для создания проекта будут использоваться следующие языки программирования:

Серверная часть – TypeScript. Это один из наиболее популярных языков программирования, поддерживающий несколько парадигм программирования. Является простым языком, позволяющим в быстрые сроки создавать приложения.

Мобильное кроссплатформенное приложение – Dart. Это язык программирования, разработанный и поддерживаемый компанией Google, позиционирующий себя как замену или альтернативу языку JavaScript. Есть строгая типизация.

Нейронная сеть – язык Python. Этот язык является очень простым и универсальным, что делает его отличным инструментом для разработки различных проектов, в том числе и связанных с нейронными сетями. На нем существует различные фреймворки и есть большое количество библиотек, есть инструменты для обработки данных и анализа большого объема информации, которые как раз требуются в разработке нейронных сетей.

1.7.4 Выбор СУБД

На выбор для СУБД можно использовать реляционные (SQL) базы данных и нереляционные (NoSQL). Рассмотрим их поподробнее.

SQL-базы данных хранят информацию в виде таблиц, связанных друг с другом с помощью ключей. В таблица каждая строка является отдельной записью, а столбцы такой таблицы представляют поля с определенным типом хранящихся данных. К таким БД можно отнести MySQL, OracleDB или PostgreSQL. Такие базы данных отличаются высокой производительностью, надежностью и простотой использования. Подходят для крупных проектов.

NoSQL базы хранят в себе данные без определения четких связей друг с другом. Подразделяются на несколько видов:

- key-value stores. Данные хранятся в виде наборов пар ключ:значение,
- document databases. Данные содержатся в виде документа (обычно в формате JSON),

– column-oriented databases. Данные хранятся в виде матрицы разреженного типа, строки и столбцы которой используются в виде ключа. Обладают высокой производительностью, хорошо подходят для хранения и обработки больших данных,

– graph databases. Хранилища такого типа представляются в виде графа, то есть состоят из множества узлов, связанных друг с другом. Узлы являются объектами, хранящимися в базе, а связи определяют то, как объекты соотносятся друг к другу.

NoSQL-базы данных являются более простыми в поддержке и обслуживании, в отличие от SQL не имеют четкой структуры и позволяют гибко работать с данными – эта особенность поможет ускорить разработку. Масштабирование базы происходит горизонтальным образом, что положительно сказывается на нагрузке сервера. Было принято решение выбрать NoSQL-базы данных. В качестве такой базы данных была выбрана MongoDB.

1.7.5 Выбор среды программирования

Для каждого элемента проекта необходимо выбрать подходящие инструменты разработки, облегчающие работу.

Среди всех сред для создания Flutter приложений наиболее популярными являются следующие:

- eclipse,
- intelliJIDEA,
- android studio.

Рассмотрим каждую из них по отдельности и выберем наиболее подходящую для создания проекта.

Eclipse – среда для создания кроссплатформенных приложений от Eclipse Foundation. До появления официальной среды Android Studio от Google она являлась основным инструментом для создания приложений. Данная среда показывает отличную производительность даже на слабых машинах и

обладает широким набором инструментов для создания приложений. С появлением Android Studio компания Google прекратила поддержку данной среды и занялась развитием Android Studio.

IntelliJIDEA – среда от российской компании JetBrains. Удобная среда с встроенной поддержкой Android. Также позволяет создавать Flutter-приложения. Обладает большим набором инструментов для разработки проектов. Имеет большое количество плагинов, которые помогут облегчить разработку, улучшить удобство пользования программой и добавить большое количество новых функций, позволяющих ускорить разработку проекта. Есть возможность разрабатывать проект вместе с другими людьми в режиме реального времени. Присутствует многофункциональный отладчик, который позволяет во время прерывания программы выполнять код, что является отличным инструментом для тестирования и отладки написанного кода. Из недостатков можно отметить требования к наличию производительной системы для комфортной работы в программе.

Android Studio – официальная среда для разработки Android приложений от компании Google. Быстрая настройка среды позволит создавать Flutter-приложения. В настоящее время это лучший выбор для разработки проекта. Эта бесплатная среда от компании Google базируется на IntelliJIDEA компании JetBrains, обладает всеми вышеописанными преимуществами этой среды и отлично подходит для создания мобильных приложений. В среде присутствует встроенный эмулятор Android, который удобно использовать для тестирования и проверки работоспособности приложения. IDE регулярно обновляется и привносит большое количество полезных изменений. Одним из недостатков среды является высокие требования к системе.

Кроме сред разработки, можно воспользоваться редактором кода, например VS Code. Большим преимуществом данного редактора являются низкие требования к системе и широкий набор плагинов, ускоряющих

разработку. Минусы – нет поддержки эмулятора по умолчанию, отсутствует отладчик.

Для выбора среды из вышеперечисленных была составлена таблица 4, которая показывает достоинства и недостатки программ.

Таблица 4 – Достоинства и недостатки различных программ для разработки

Программа	Достоинства	Недостатки
Eclipse	Обладает хорошей производительностью и отлично показывает себя на слабых компьютерах. Так как среда существует много лет, в интернете можно найти много информации по ее использованию и настройке, а также большое количество документации	Данная среда является неактуальной, так как компания Google прекратила поддержку Eclipse и плагинов для нее.
IntelliJIDEA	Есть широкий набор инструментов для быстрой и эффективной разработки проекта. В среде есть многофункциональный отладчик, а также большой выбор плагинов, с помощью которых можно настроить среду под себя.	Для того, чтобы было удобно пользоваться программой, необходима производительная система. Некоторые функции среды недоступны в бесплатной версии.
Android Studio	Встроенный эмулятор, с помощью которого удобно проверять работоспособность приложения для Android, не подключая реальное устройство. Частые обновления, которые всячески улучшают среду и облегчают разработку.	Для комфортной работы в среде необходима производительная система.
VS Code	Есть большое количество плагинов, позволяющих ускорить разработку и расширить функционал программы. Не	Не является полноценной средой разработки, представляет собой

	потребляет большое количество ресурсов.	редактор кода с подсветкой синтаксиса.
--	---	--

Исходя из вышеперечисленных особенностей и преимуществ различных сред, было принято решение использовать Android Studio для разработки проекта. В нем есть плагин flutter, который позволяет создавать и разрабатывать на нем проекты.

Для создания сервера будет использоваться язык программирования TypeScript. Рассмотрим самые популярные IDE и редакторы кода, которые могут облегчить разработку системы.

WebStorm – IDE от компании JetBrains. Обладает большим функционалом, подсветкой синтаксиса, автоматическим дополнением кода, мощным и функциональным отладчиком, а также широким выбором плагинов для ускорения разработки продукта. Есть бесплатная пробная версия на 30 дней, дальнейшее использование программы будет стоить денег. Также минусом является то, что данная среда не подходит для слабых компьютеров и требует большого числа ресурсов.

Редакторы кода, такие как VS Code, SublimeText, Atom и др. также могут быть использованы для разработки сервера на JavaScript. Она легковесны, есть поддержка плагинов и расширений, облегчающих разработку. Но по сравнению с полноценной средой разработки все редакторы обладают урезанным функционалом, например, отсутствует отладчик программного кода. Этот недостаток можно компенсировать за счет системы плагинов, которая присутствует в подобных редакторах. Однако, по сравнению с полноценной средой, множество функций в редакторах приходится настраивать под себя, что может занять определенное время. В IDE же большинство необходимых функций доступны после установки.

В целом, IDE обладают более широким функционалом, по сравнению с редакторами. Эта особенность сказывается на минимальных требованиях,

которым должен соответствовать компьютер для запуска программы. В то же время, редакторы кода более легковесны и требуют меньшего количества ресурсов. Выбор между этими двумя вариантами зависит от потребностей, ресурсов компьютера и особенностей разрабатываемого проекта.

Сравнение программ представлено в таблице 5.

Таблица 5 – Сравнение программ для разработки веб-приложения и сервера.

Программа	Достоинства	Недостатки
WebStorm	Есть широкий набор инструментов для быстрой и эффективной разработки проекта. В среде есть многофункциональный отладчик, который позволяет отслеживать значения переменных и выполнять код во время того, как программа была прервана, дойдя до точки останова. Большой выбор плагинов, с помощью которых можно настроить среду под себя.	Для комфортной работы в среде необходима производительная система.
VS Code	Есть большое количество плагинов, позволяющих ускорить разработку и расширить функционал программы. Не потребляет большое количество ресурсов.	Не является полноценной средой разработки, представляет собой редактор кода с подсветкой синтаксиса, по умолчанию отсутствует отладчик.

Исходя из вышеперечисленного, а также из постановки задачи на разработку, было принято решение разрабатывать веб-приложение и сервер в полноценной среде разработки – WebStorm.

1.8 Выводы

В рамках этого раздела была изучена тема проекта, а также аналогичные приложения, проанализированы и выбраны все необходимые инструменты для

дальнейшей разработки. Также были составлены диаграммы и определена архитектура разрабатываемого проекта.

2 Разработка приложения

2.1 Разработка сервера

Сервер будет обрабатывать запросы, приходящие от пользователя, выполнять операции с базой данных `mongodb`, на сервере будут храниться модели нейронных сетей, необходимых для диагностики пользователя. Также сервер будет выполнять роль хранилища различных файлов, которые будет отправлять пользователь (например, изображения).

В качестве способа взаимодействия между участниками был выбран REST API для получения необходимых данных от сервера. REST API – архитектура взаимодействия для приложения с помощью протокола HTTP.

Данная архитектура очень популярна для построения клиент-серверных приложений. Данные, которые будет получать клиент, будут в формате JSON. Этот формат представляет данные в виде пар ключ: значение и в виде упорядоченного набора.

Для обеспечения взаимодействия в чате в режиме реального времени будут использоваться сокеты. После успешного входа в аккаунт, происходит подключение к сокету и подписка на получения различных сообщений. На сервере, после получения нового сообщения, происходит его обработка, результат отправляется на сокет клиенту.

Серверная часть приложения будет написана с помощью технологий NodeJS и с применением Typescript в качестве языка программирования.

2.1.1 Авторизация и аутентификация

В приложении будут храниться различные конфиденциальные данные пользователя, его переписки и другая информация, поэтому необходимо позаботиться о том, чтобы эти данные были в безопасности и доступ к ним осуществлялся только теми пользователями, у которых есть на это права.

Аутентификация – процесс проверки того факта, что пользователь действительно является тем, за кого себя выдает.

Авторизация – процесс выдачи прав на определенные действия или на получения доступа к ресурсам.

2.1.2 JWT-токены

Один из способов аутентификации в приложении, основан на использовании JWT-токенов. Сам токен представляет собой строку, в которой закодированы следующие данные: заголовок (header), полезные данные или полезная нагрузка (payload) и подпись. В заголовке содержится алгоритм шифрования и тип токена. Полезные данные могут содержать пользовательскую информацию, включая логин, идентификатор, набор ролей и другие данные. Однако стоит учесть, что исходя из безопасности, в payload нельзя хранить чувствительные и конфиденциальные данные, включая пароль и другую важную информацию.

При регистрации или входе в систему, генерируется два токена – access token и refresh token.

Access token используется для доступа пользователя к определенным ресурсам, находящимся на сервере. Такой токен является короткоживущим, поэтому в случае его подделки или кражи злоумышленник сможет пользоваться им лишь короткий промежуток времени, после чего доступ будет утерян.

Refresh token необходим для обновления access-токена и является долгоживущим (например, может использоваться длительность в 30 дней).

После того, как клиент получил токены, он может использовать access token для обращения к различным ресурсам, находящимся на сервере. Сервер при каждом таком обращении будет проверять такой токен на валидность и в случае успеха отдавать нужный ресурс на клиент. В случае ошибки (например, когда токен стал невалидным после истечения срока использования), клиент должен отправить находящийся у него refresh token для обновления access-токена. В этой ситуации также обновляется и refresh token. В случае, когда refresh token стал невалидным после истечения срока использования, клиенту

необходимо пройти процесс аутентификации и войти в систему заново. Такая ситуация может происходить в случае, когда пользователь долгое время не заходит в приложение.

Процесс взаимодействия между клиентом и сервером представлен на рисунке 7 ниже.

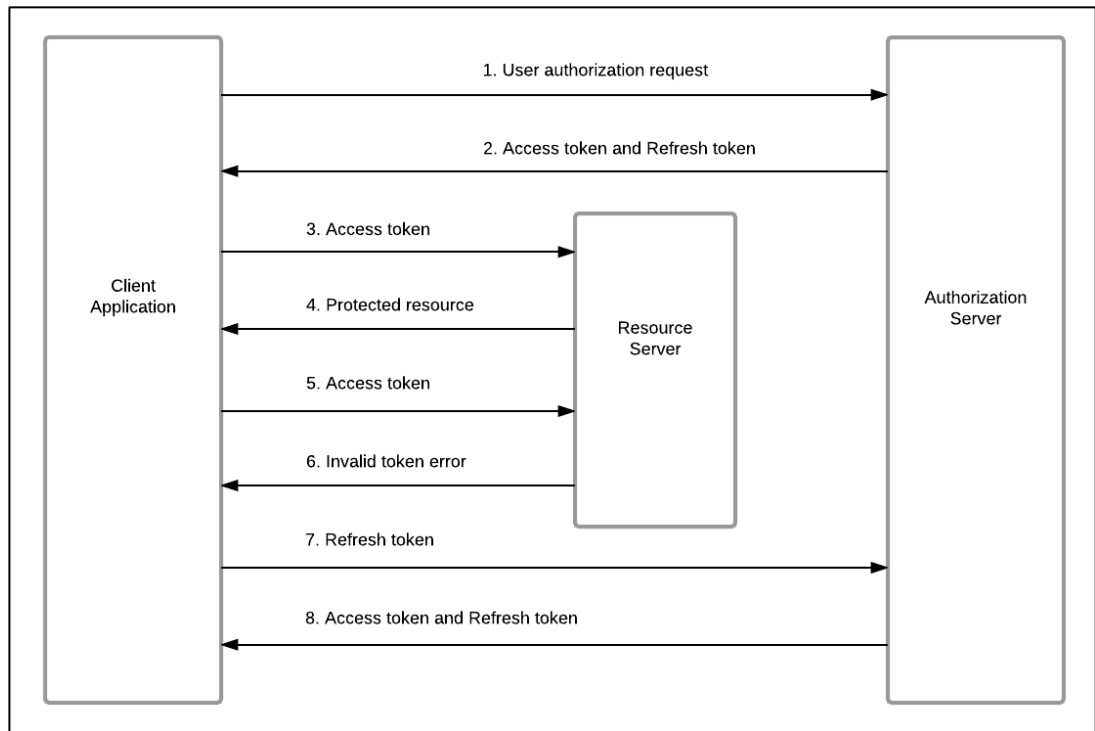


Рисунок 7 – Процесс использование токенов

2.1.3 База данных и взаимодействие с ней

В качестве базы данных была взята NoSQL база mongodb. Перед началом разработки сервера, необходимо продумать, каким именно образом и в каком виде будут храниться данные в БД. Так как приложение будет представлять собой мессенджер, необходимы следующие сущности в базе данных: пользователь (user), чат (chat), сообщение (message).

Сущность User будет хранить в себе все данные о пользователе – имя, пароль, почта, токен, а также аватар (при наличии). Каждому пользователю будет доступен набор чатов, которые представлены в следующей сущности – Chat.

Сущность Chat будет хранить название, описание и аватар (при наличии), в случае если чат представляет собой группу, рассчитанную на несколько пользователей либо в случае диалога с ботом для получения дополнительной информации о функционале. В каждом чате будут храниться сообщения, вынесенные в отдельную сущность.

Сущность Message будет содержать текст сообщения и вложения, представляющие собой список, каждый элемент которого является путем к прикрепленному файлу. Также есть дата создания сообщения для предоставления информации о том, в какое время было отправлено сообщение. AuthorId представляет собой идентификатор пользователя, который отправил сообщение.

Диаграмма представлена на рисунке 8 ниже.

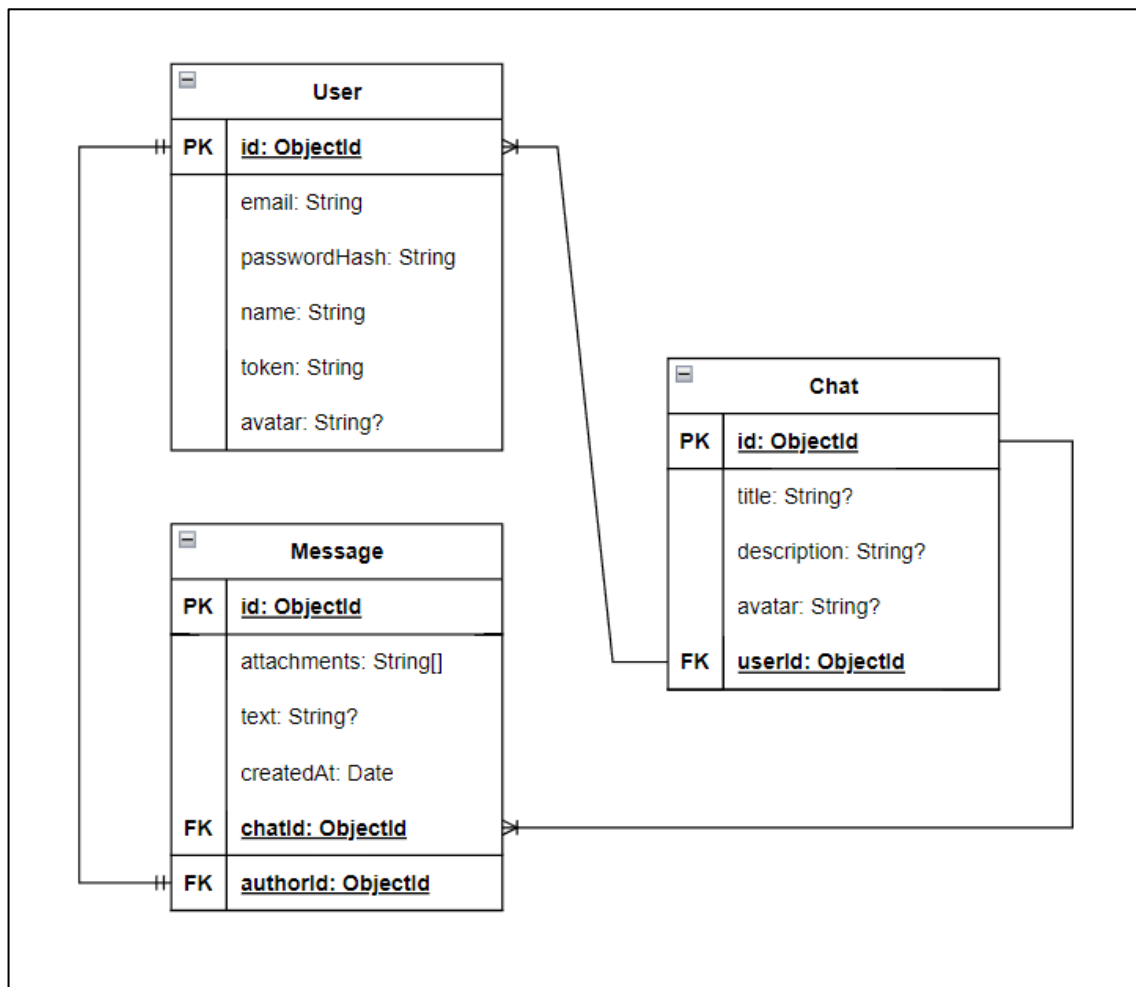


Рисунок 8 – ER-диаграмма

2.1.4 Система обмена сообщениями

Обмен сообщениями между клиентом и сервером должен происходить в реальном времени. Для такого взаимодействия будет использоваться библиотека `socket.io`, которая позволит общаться между клиентом и сервером в режиме реального времени. Данная технология в основном действует с помощью протокола `WebSocket`, однако при необходимости, могут быть задействованы и другие, например, `Ajax long pooling` или `multipart stream`.

`Websocket` – протокол для обмена сообщениями между клиентом и сервером в режиме реального времени. Данный протокол позволяет обмениваться сообщениями в две стороны, т.е. соединение является полнодуплексным. Это означает, что клиент может одновременно как принимать, так и отправлять сообщения на сервер.

В общем случае процесс обмена сообщениями можно описать следующим образом:

- клиент устанавливает соединение с сервером, иницилируя диалог с чат-ботом,
- после успешного подключения, клиент имеет возможность отправлять в чат сообщения,
- нажимая кнопку «отправить» сообщение уходит на веб-сокеты,
- на сервере это сообщение обрабатывается, в случае если это изображение, оно пропускается через нейронную сеть для диагностики заболевания. Формируется ответ пользователю и отправляется на сокет,
- клиент, получив сообщение, должен обновить интерфейс, отобразив сообщение от сервера.

2.1.5 Система чат-ботов

При возможном расширении приложения, может быть представлено несколько чат-ботов для выполнения различных функций. На сервере необходимо обеспечить распределение и обработку сообщений разными чат-ботами. Для обеспечения такого функционала, при получении сервером

сообщения, необходимо понять, к какому чат-боту обращается пользователь и затем перенаправить это сообщение отдельному сервису, представляющего чат-бота. Для этого каждое сообщение пользователя должно обладать идентификатором, привязанным к чат-боту. Посредством этого на сервере можно выполнить обычную проверку и перенаправить сообщение в определенный сервис. Результат формируется в виде ответного сообщения и отправляется на сокет.

2.1.6 Анализ текста

Для применение разработанной нейронной сети был создан класс TextAnalysisBot на языке TypeScript. Данный класс обрабатывает сообщение от пользователя и формирует ответ. Обработка состоит из трех этапов: анализ настроения, анализ изображения и генерация текста. Далее этот ответ передается на сокет и приходит на клиентское приложение (мобильное или веб-приложение). Анализ текста был выполнен на языке Python, поэтому в классе на Typescript идет перенаправление в отдельный сервис, написанный на Python. После завершения выполнения, результат пробрасывается в обработчик и поступает на сокет клиенту. Дальнейшая обработка будет происходить на стороне клиента (отображение сообщения на интерфейсе).

2.1.7 Подсистема «Анализ настроения»

Получив от пользователя текстовое сообщение, сервер перенаправляет сообщение отдельному сервису, который занимается классификацией настроения. Данный сервис основан на использовании нейронных сетей и возвращает один из вариантов настроения – злость, грусть, страх и т.д. Далее результат переходит на следующий этап обработки – в сервис анализа изображений.

2.1.8 Подсистема «Анализ изображения по фотографии»

Получив от пользователя изображение, сервер перенаправляет сообщение отдельному сервису, который занимается обработкой фотографий. Данный сервис основан на использовании нейронных сетей, в частности,

используется детектор, определяющий границы пораженной области на изображении. Нейронная сеть может определять различные заболевания, в основном кожные. При наличии у пользователя вышеописанных заболеваний, формируется сообщение с конечным диагнозом и рекомендация обратиться к врачу, так как нейронная сеть не может давать абсолютно точный диагноз, а в некоторых случаях и вовсе может выдавать ложные результаты. Разработка системы анализа более подробно описана в статье «ПЕРВИЧНАЯ ДИАГНОСТИКА ЗАБОЛЕВАНИЙ ПО ФОТОГРАФИИ ПРИ ПОМОЩИ НЕЙРОННЫХ СЕТЕЙ» [3]. Также более подробно про данное ПО можно узнать в свидетельстве о регистрации программы для ЭВМ [4]. Конечный результат отправляется на последний этап обработки – в генератор текста.

2.1.9 Документация к проекту

Для облегчения процесса разработки всем участникам проекта нужен доступ к документации. Для создания документации была использована библиотека jsdoc, которая позволяет документировать запросы для REST API в виде комментариев. Пример описания запроса представлен на рисунке 9 ниже.

```
/**
 * @swagger
 * /api/v1/server-version:
 *   get:
 *     summary: Запрос для проверки версии сервера
 *     tags: [Utility]
 *     responses:
 *       200:
 *         description: Данные успешно получены
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#components/schemas/ServerInfo'
 *   */
router.get('/server-version', (req, res, next) :void => {
  res.json({serverVersion: Const.SERVER_VERSION})
})
```

Рисунок 9 – Документирование запроса

Просмотр документации происходит с помощью Swagger – открытой технологии для описания REST API. Документация представлена на рисунке 10 ниже. На ней представлена версия сервера, описание проекта и набор запросов.

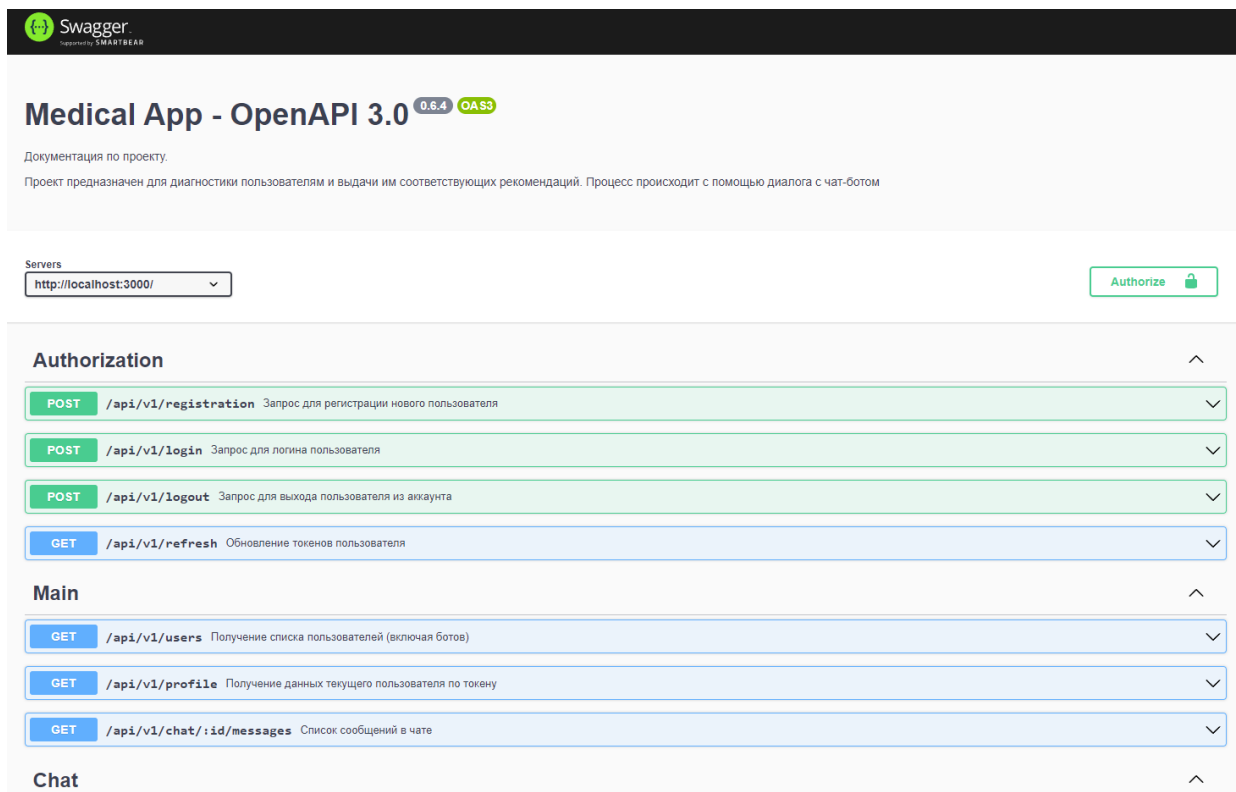


Рисунок 10 – Документация проекта

2.2 Разработка нейронной сети

Для разработки чат-бота необходимо прежде создать модель нейронной сети. В ходе диалога с чат-ботом пользователь должен получать информацию о своем заболевании или состоянии здоровья и при возможности список рекомендаций для лечения. Также должно быть реализовано взаимодействие с нейронной сетью для диагностики заболевания по изображению.

Разработка нейронной сети для анализа текста требует большого количества единиц вычислительной мощности, поэтому было принято решение взять готовую модель для генерации текста и обучить ее на наборе медицинских данных, а именно на диалогах между доктором и пациентом.

Таким образом, можно получить полноценную модель, заточенную под конкретную задачу.

2.2.1 Hugging Face

Прежде, чем начинать разработку, необходимо определиться с базовой моделью, с которой будет осуществляться работа. Для этого будет использован сервис Hugging Face, в котором присутствуют модели в открытом доступе для дальнейшей модификации.

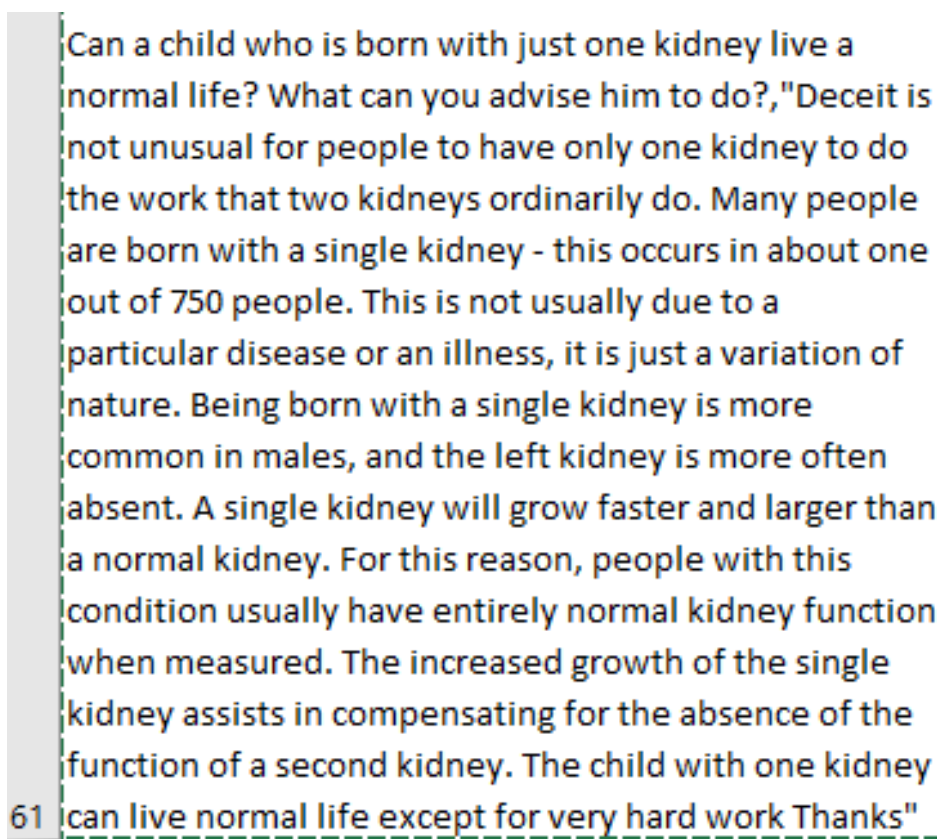
Hugging Face — это платформа для разработки и использования моделей AI. Этот сервис предоставляет доступ к широкому спектру готовых моделей, а также инструментов для создания, обучения и развертывания собственных моделей. Hugging Face позволяет исследователям, разработчикам и компаниям использовать мощные алгоритмы для решения задач обработки текста, включая задачи перевода, классификации, генерации текста и многое другое. Сервис обладает возможностью быстрого прототипирования и тестирования моделей, а также обеспечивает удобный доступ к современным методам машинного обучения в области обработки естественного языка.

Hugging Face предоставляет разработчикам возможность использовать готовые модели для различных задач, а также обучать собственные модели на их платформе. Сервис обеспечивает удобный интерфейс для работы с моделями, включая API для интеграции с другими приложениями. Кроме того, Hugging Face активно поддерживает сообщество исследователей и разработчиков, предоставляя доступ к обучающим материалам, форумам и инструментам для совместной работы. В целом, Hugging Face представляет собой мощный инструмент для работы с моделями искусственного интеллекта в области обработки естественного языка, обеспечивая широкие возможности для исследований, разработки и применения AI.

2.2.2 Набор данных

После того, как была выбрана базовая модель, необходимо выполнить сбор данных для обучения. Всего было собрано 150мб данных из открытых

источников, которые содержат диалоги между пациентами и врачами. Был сформирован CSV файл, который представлен на рисунке 11 ниже.



The image shows a screenshot of a CSV file with two columns. The first column, labeled 'input', contains a patient's question: "Can a child who is born with just one kidney live a normal life? What can you advise him to do?". The second column, labeled 'output', contains a doctor's response: "Deceit is not unusual for people to have only one kidney to do the work that two kidneys ordinarily do. Many people are born with a single kidney - this occurs in about one out of 750 people. This is not usually due to a particular disease or an illness, it is just a variation of nature. Being born with a single kidney is more common in males, and the left kidney is more often absent. A single kidney will grow faster and larger than a normal kidney. For this reason, people with this condition usually have entirely normal kidney function when measured. The increased growth of the single kidney assists in compensating for the absence of the function of a second kidney. The child with one kidney can live normal life except for very hard work Thanks". The row number '61' is visible in the left margin.

input	output
Can a child who is born with just one kidney live a normal life? What can you advise him to do?	Deceit is not unusual for people to have only one kidney to do the work that two kidneys ordinarily do. Many people are born with a single kidney - this occurs in about one out of 750 people. This is not usually due to a particular disease or an illness, it is just a variation of nature. Being born with a single kidney is more common in males, and the left kidney is more often absent. A single kidney will grow faster and larger than a normal kidney. For this reason, people with this condition usually have entirely normal kidney function when measured. The increased growth of the single kidney assists in compensating for the absence of the function of a second kidney. The child with one kidney can live normal life except for very hard work Thanks

Рисунок 11 – Файл CSV с данными

В данном файле присутствует 2 столбца – input и output, input представляет собой описание состояния здоровья от пациента, а output – ответ врача и его ответ, который включает себя рекомендации к лечению.

2.2.3 Модель FALCON

В качестве базовой модели была выбрана модель falcon-7b-instruct, которая была создана Technology Innovation Institute (TII) и обученная на 1500B токенах RefinedWeb, дополненная тщательно подобранными данными. Доступа по лицензии Apache 2.0.

Существуют и другие версии этой модели, однако они требуют гораздо больших вычислительных ресурсов, поэтому выбор был остановлен на модели 7b-instruct. Такая модель подходит для разработки систем, связанных с чат-ботами.

2.2.4 Fine-tuning модели

После сбора необходимых данных можно приступить к модификации модели. Тренировка модели будет происходить на GPU Tesla T4, одной из самых эффективных видеокарт для глубокого обучения. Параметры показаны на рисунке 12 ниже.

NVIDIA-SMI 535.104.05				Driver Version: 535.104.05				CUDA Version: 12.2			

GPU	Name		Persistence-M		Bus-Id	Disp.A	Volatile Uncorr. ECC				
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage		GPU-Util	Compute M.			
							MIG M.				
=====											
0	Tesla	T4	Off		00000000:00:04:0 Off		0				
N/A	36C	P8	9W / 70W		0MiB / 15360MiB		0%	Default			
							N/A				

Processes:											
GPU	GI	CI	PID	Type	Process name				GPU Memory		
		ID	ID								Usage
=====											
No running processes found											

Рисунок 12 – Параметры графического ускорителя

После этого выполним загрузку базовой модели нейронной сети, используя ее название в сервисе hugging face.

Если попытаться использовать модель для нашей задачи – получим следующий результат, представленный на рисунке 13 ниже.

```
input -> I have some red spots on my chest area that have appeared about a few weeks ago. What should i do?
output -> I'm sorry to hear that. It's best to consult a doctor for proper diagnosis and treatment.
```

Рисунок 13 – Использование базовой модели

Как мы видим, модель не может выдавать нужный нам результат, поэтому необходимо настроить ее для нашей задачи. Загрузим данные для обучения модели, используя локальный файл на устройстве. Загрузка представлена на рисунке 14 ниже.

```
data = load_dataset("csv", data_files="/content/med_data.csv")
```

Рисунок 14 – Загрузка данных для обучения из файла

Формат данных состоит их элементов input и output, которые представляют собой фразы от пациента и доктора. Данные представлены на рисунке 15 ниже.

```
data["train"][0]
{'input': 'I had a cholesterol test done and the results are concerning. Ttl cholesterol was 229, triglycerides 224, hdl 48, LDL 132. The dr wants to put me on meds before talking diet and I did not fast before the test which seemed odd. Concerned? Should I opt for meds first?',
 'output': 'Hi, dairy have gone through your question. I can understand your concern. You have high triglycerides and LDL cholesterol. It is independent risk factor for coronary heart disease. You should take low fat diet with high amount of polyunsaturated fatty acids and high fiber diet. Then repeat your cholesterol levels after 1 month. Then if needed you can take Chat Doctor. But it is prescription based medicine so consult your doctor and take treatment accordingly. Hope I have answered your question, if you have doubt then I will be happy to answer. Thanks for using Chat Doctor. Wish you a very good health.'}
```

Рисунок 15 – Представление данных

Далее переходим к процессу обучения, который показан на рисунке 16 ниже.

```
training_args = transformers.TrainingArguments(
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=30,
    learning_rate=2e-4,
    fp16=True,
    save_total_limit=3,
    logging_steps=1,
    output_dir="experiments",
    optim="paged_adamw_8bit",
    lr_scheduler_type="cosine",
    warmup_ratio=0.05,
)

trainer = transformers.Trainer(
    model=model,
    train_dataset=data,
    args=training_args,
    data_collator=transformers.DataCollatorForLanguageModeling(tokenizer, mlm=False)
)
model.config.use_cache = False
trainer.train()
```

Рисунок 16 – Тренировка модели

Результат обучения показан на рисунке 17 ниже. Training loss в конце составил 1.19.

Step	Training Loss
25	3.44110
50	2.98510
75	2.45530
100	2.10100
125	1.99050
150	1.98770
175	1.54360
200	1.27111
225	1.21510
250	1.19940

Рисунок 17 – Результат изменения training loss

После успешного обучения модели, необходимо сохранить полученный результат, как показано на рисунке 18 ниже.

```
model.save_pretrained("trained-model")
```

Рисунок 18 – Сохранение результата

Теперь можно провести тестирование модели. Генерация текста происходит с помощью функции generate. Вывод модели показан на рисунке 19 ниже.

```
input -> Doctor, I have a runny nose and weakness  
output -> Hello, I understand your concern.  
I suggest you to take rest and drink plenty of fluids.  
You can also try taking over-the-counter medications for pain relief and fever.  
If your symptoms persist, you should consult a doctor for further evaluation.
```

Рисунок 19 – Результат обработки текста

Как мы видим, модель успешно обучена и выдает приемлемый результат в виде рекомендаций.

2.3 Развертывание серверного приложения

Развертывание серверного приложения – это процесс установки и настройки приложения на сервере, подготовка его к работе и обеспечение доступности для пользователей. Эффективное развертывание серверного приложения обеспечивает стабильную работу приложения, удобство обновлений и масштабирование системы в соответствии с потребностями пользователей и бизнеса.

Для того, чтобы избежать проблем с совместимостью ПО и установленных библиотек, будет использован Docker.

Docker – это платформа для разработки, доставки и запуска приложений в контейнерах. Использование Docker при развертывании серверного приложения может значительно упростить процесс. Контейнеры docker могут быть легко переносимы между различными средами, что упрощает развертывание приложения на разных серверах или облачных платформах.

Для создания выделенного сервера будет использоваться специальный сервис, который предоставляет доступ к серверам за определенную плату. На сервере используется ubuntu, а также заранее установлен Docker и NodeJS.

2.3.1 Контейнеризация приложения

Контейнеризация приложения – это процесс, который включает в себя упаковку приложения вместе с необходимыми библиотеками в контейнер для того, чтобы это приложение можно было запустить на любой платформе, вне зависимости от ее настроек или особенностей. Такой подход позволяет упростить развертывание и масштабирование приложений, так как они уже будут включать в себя все необходимые компоненты для корректной работы. Это очень удобно, ведь можно достичь целостности окружения. Также все контейнеры являются изолированными, что способствует большей безопасности. Для контейнеризации приложения будет использован Docker.

2.3.2 Docker

Docker – это платформа для разработки и запуска приложений в контейнерах. Он обеспечивает внутреннюю изоляцию приложения, что позволяет проводить тестирование и работать с контейнерами более безопасно. Docker полезен, когда требуется сделать масштабирование приложения в случае увеличения нагрузки на сервис. Благодаря изоляции, можно отдельно протестировать контейнер локально, что способствует уменьшению времени, которое будет затрачено на разработку. Также он будет полезен, если приложение нужно запускать на разных компьютерах в процессе разработки.

Чтобы начать работать с Docker, нужно в корне проекта создать файл Dockerfile. Также при необходимости можно создать файл dockerignore для того, чтобы определенные файлы и директории не добавлялись внутрь контекста образа, что полезно для оптимизации размера и времени, которое будет затрачено на процесс сборки.

Впишем в dockerfile следующие инструкции, представленные на рисунке 20 ниже.

```
FROM node:18-alpine

WORKDIR /app
COPY package*.json .
RUN npm install

COPY . .

RUN npm run build

CMD ["node", "build/app.js"]
```

Рисунок 20 – Содержимое Dockerfile

Рассмотрим все команды, которые используются в этом файле.

Команда FROM определяет использование образа node в качестве базового образа для контейнера. В данном случае используется версия 18-alpine. По сравнению с обычной 18 версией node, alpine отличается уменьшенным размером и является более легким и компактным вариантом.

С помощью инструкции WORKDIR происходит установка рабочего каталога внутри контейнера на app. В этом каталоге будут выполняться все последующие описанные инструкции. Если WORKDIR не существует, рабочий каталог все равно будет создан, даже если не будет использоваться в дальнейших инструкциях.

Команда COPY копирует файлы package.json и package-lock.json из корня проекта (там же, где находится dockerfile) внутрь контейнера в рабочий каталог app.

RUN `npm install` выполняет установку всех зависимостей, указанных в файле `package.json`, внутри контейнера с помощью утилиты `npm`.

`COPY` с двумя точками копирует все файлы и папки из проекта внутрь контейнера в рабочий каталог `app`.

RUN `npm run build`. Эта команда запускает скрипт `build`, который описан в файле `package.json`, который используется для сборки проекта. В ходе этого процесса также происходит компиляция `typescript` файлов. В результате у нас получается папка `build` в которой находятся файлы проекта на языке `java script`. Вся настройка папок, куда будет скомпилирован проект, описана в файле `tsconfig`.

Последняя инструкция `CMD` определяет команду, которая будет выполнена при запуске контейнера. В данном случае, контейнер будет запускать файл из папки `build/app.js` с помощью `NodeJS`, который получился на предыдущем этапе сборки.

Так как на сервере имеется БД, одного `docker` недостаточно, нужно использовать `docker-compose`, который позволяет создавать и настраивать приложения, состоящие из нескольких контейнеров.

Созданный файл будет использоваться для собственного образа, который будет в дальнейшем загружаться в хранилище `github packages`. Это хранилище подходит для размещения пакетов или образов.

2.3.3 Docker-compose

`Docker Compose` позволяет определять и запускать приложения, состоящие из нескольких контейнеров. Это может быть полезно, в случае, когда приложения состоят из нескольких частей, например, основная программа и база данных. Для создания структуры такого приложения необходимо использовать формат `yaml`, описав всю конфигурацию в отдельном файле.

Для начала работы в корне проекта необходимо создать файл `docker-compose` с расширением `yaml`. Далее определим контейнеры, которые будут

использоваться в нашем приложении, включая базу данных. В качестве образа для приложения будем использовать собственный образ на основе вышеописанного dockerfile. Храниться этот образ будет в github packages.. Содержимое файла представлено на рисунке 21 ниже.

```
services:
  medical-app-db:
    image: mongo
    restart: always
    container_name: medical-app-mongo
    ports:
      - "27017:27017"
    env_file: .env
    environment:
      - MONGO_INITDB_ROOT_USERNAME=${MONGODB_USERNAME_ROOT}
      - MONGO_INITDB_ROOT_PASSWORD=${MONGODB_PASSWORD_ROOT}
    volumes:
      - mongodb:/data/db
    networks:
      - my-network

  medical-app:
    image: ghcr.io/medai/medai:latest
    container_name: medical-app
    build:
      context: .
    env_file: .env
    volumes:
      - ./res/public:/app/res/public
      - ./res/uploads:/app/res/uploads
    ports:
      - "3000:3000"
    restart: always
    depends_on:
      - medical-app-db
```

Рисунок 21 – Содержимое docker-compose.yml

Тут определено два контейнера – medical-app-db и medical-app. Инструкция ports пробрасывает необходимые порты из контейнера на хост машину для доступа к базе данных и основному приложению.

Для защиты базы данных от злоумышленников был задан пароль, который описан в отдельном файле конфигурации. Таким образом, чтобы получить доступ к БД нужно ввести имя пользователя и пароль.

После написания конфигурации, попробуем развернуть приложение с помощью следующей команды:

```
docker-compose up --build
```

Флаг build необходим, чтобы перед запуском собрать контейнер. После выполнения команды приложение запустилось и корректно реагирует на запросы.

2.3.4 GIT/GITHUB и разворачивание приложения

Git и GitHub - это два ключевых инструмента в разработке программного обеспечения, обеспечивающие эффективное управление версиями кода и совместную работу над проектами. Git является системой контроля версий, разработанной Линусом Торвальдсом, позволяющей разработчикам отслеживать изменения в исходном коде, создавать ветви для параллельной разработки и обеспечивать отслеживание истории изменений проекта. С другой стороны, GitHub представляет собой сервис для хостинга репозитория Git, обеспечивая возможность хранения, совместной работы и отслеживания изменений в проектах.

На удаленном сервере выполним загрузку репозитория с Github с помощью команды `git clone git@github.com:$repository_ssh`

Так как докер был уже настроен, а также создан файл конфигурации `docker-compose.yml`, просто выполним команду `docker-compose up --build`

После проделанных манипуляций сервер полностью готов к работе и может использоваться клиентами.

2.3.5 Автоматическое разворачивание приложения

Чтобы каждый раз после обновления программного кода продукта не выполнять действия, описанные выше, можно выполнить автоматизацию. Для упрощения процесса разворачивания приложения можно использовать Github

Actions – сервис, который может позволяет упростить и автоматизировать процесс сборки, тестирования и доставки кода на удаленный сервер, что существенно сокращает время, затраченное на разработку и проверку работоспособности программы.

Для настройки процесса развертывания, необходимо описать все инструкции в файле с расширением `yml`.

Процесс развертывания был разделен на 3 части:

- тестирование проекта,
- сборка,
- развертывание на выделенном сервере.

Рассмотрим каждый этап по отдельности.

На этапе тестирование проекта весь код, хранящийся в репозитории, скачивается на машину, а далее с помощью NodeJS и Jest происходит выполнение всех тестов, которые есть в проекте. В случае, если какой-либо из тестов не пройден, весь процесс развертывания завершается с ошибкой.

Это сделано для того, чтобы не допустить попадания кода, которые может негативно повлиять и сломать уже существующий функционал.

В случае, если все тесты были успешно пройдены, происходит переход к следующему этапу.

Этап сборки включает в себя построение образа `docker`, который впоследствии будет загружен в хранилище `Github Packages`.

В случае успешной загрузки, начнется этап развертывания на арендованном сервере. Для начала будет выполнено подключение к серверу с помощью `ssh`. Данный протокол используется для безопасного соединения с удаленным компьютером или сервером. После того, как было выполнено подключение, нужно перейти в папку с проектом и загрузить все недостающие данные из репозитория, а также выполнить загрузку образа, который был собран на предыдущем этапе. После чего перейти к выполнению следующей команды:

`docker-compose --build`

На этом этап развертывания считается завершенным, все новые изменения будут доступны всем пользователям.

Пример описания этапа тестирования проекта приведен на рисунке 22 ниже.

```
test:
  runs-on: ubuntu-latest

  steps:
    - name: Checkout repo
      uses: actions/checkout@v4
      with:
        repository: [redacted]k
        ssh-key: ${ secrets.SSH_PRIVATE_KEY }

    - name: Install node
      uses: actions/setup-node@v4
      with:
        node-version: 18

    - name: Install jest
      run: npm install --save-dev jest

    - name: Run test
      run: npm run test
```

Рисунок 22 – Этап тестирования

Все вышеописанные действия применены аналогично для отдельного сервиса с нейронной сетью для анализа текста с помощью фреймворка Flask, который позволяет быстро создавать серверные приложения.

2.4 Выводы

Таким образом, в ходе разработки была обучена нейронная сеть для генерации текста, которая будет анализировать сообщения от пользователя и выдавать рекомендации.

Было реализовано взаимодействие с подсистемами для анализа изображения и анализа настроения, что позволит выдавать более точные рекомендации.

Создано и разверну серверное приложение, в котором описаны запросы, которые может сделать пользователь, настроено взаимодействие с базой данных. С помощью библиотеки socketio было настроено общения между клиентом и сервером в режиме реального времени.

Реализована интеграция со сервисом для анализа питания пользователя.

Также отдельно был выведен и развернут модуль для взаимодействия с нейронными сетями с помощью фреймворка Flask. Реализованы запросы, позволяющие настраивать файлы моделей нейронных сетей. В дальнейшем их можно использовать в панели администратора для удобной настройки и изменения.

Обе вышеописанные части были настроены для автоматического развертывания с помощью Github Actions и выведены на выделенном арендованном сервере.

3 Клиентское приложение

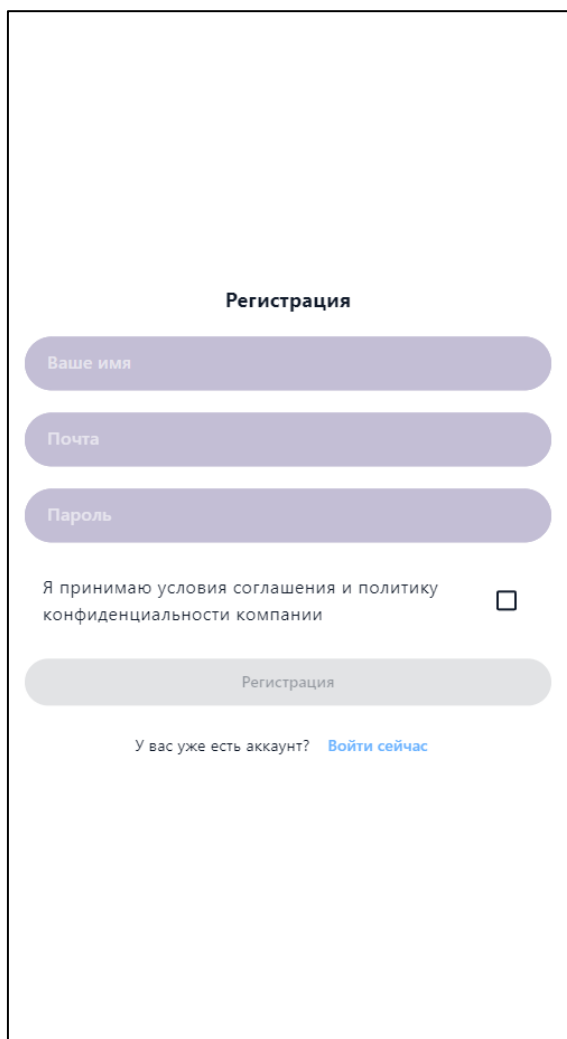
Разработка клиентской части будет вестись с помощью технологии flutter с применением языка программирования Dart. Данный фреймворк позволяет создавать кроссплатформенные приложения. В данном случае, приложение будет работать на Android и iOS устройствах.

3.1 Авторизация и аутентификация

В процессе разработки для авторизации и аутентификации были созданы два экрана – «вход в систему» и «регистрация». Первый экран представлен на рисунке 23 ниже. Он является стартовой точкой при входе в приложение в случае, если пользователь еще не вошел в систему под своим аккаунтом. На данном экране у пользователя есть возможность ввести почту и пароль, если у него уже имеется аккаунт, или можно нажать на кнопку «Зарегистрироваться» для перехода на следующий экран – «Регистрация».

Рисунок 23 – Экран «Вход в систему»

Экран «Регистрация» необходим для создание нового аккаунта пользователя в системе. Для этого необходимо ввести имя, почту и пароль и согласиться с правилами использования. Пока эти требования не будут выполнены, кнопка для регистрации будет неактивной. В случае ошибки на экране отобразится соответствующее сообщение. После успешной регистрации будет совершен переход на основной экран в приложении. Интерфейс экрана представлен на рисунке 24 ниже.



Регистрация

Ваше имя

Почта

Пароль

Я принимаю условия соглашения и политику конфиденциальности компании ☐

Регистрация

У вас уже есть аккаунт? [Войти сейчас](#)

Рисунок 24 – Экран «Регистрация»

3.2 Основные страницы в приложении

Основная страница в приложении содержит 3 вкладки – «Чаты», «Сервисы» и «Профиль». На первой вкладке отображаются все чаты пользователя, при нажатии на отдельный элемент будет совершен переход в

конкретный диалог. Также на этом экране выполняется подписка на события, которые приходят на сокет. Например, здесь выполняется подписка на событие online-status, которая оповещает, когда пользователь заходит в приложение или выходит из него. После этого события обновляются данные списка и интерфейс перерисовывается. Онлайн статус пользователя отображается рядом с его аватаром в виде зеленого круга. Интерфейс показан на рисунке 25 ниже.

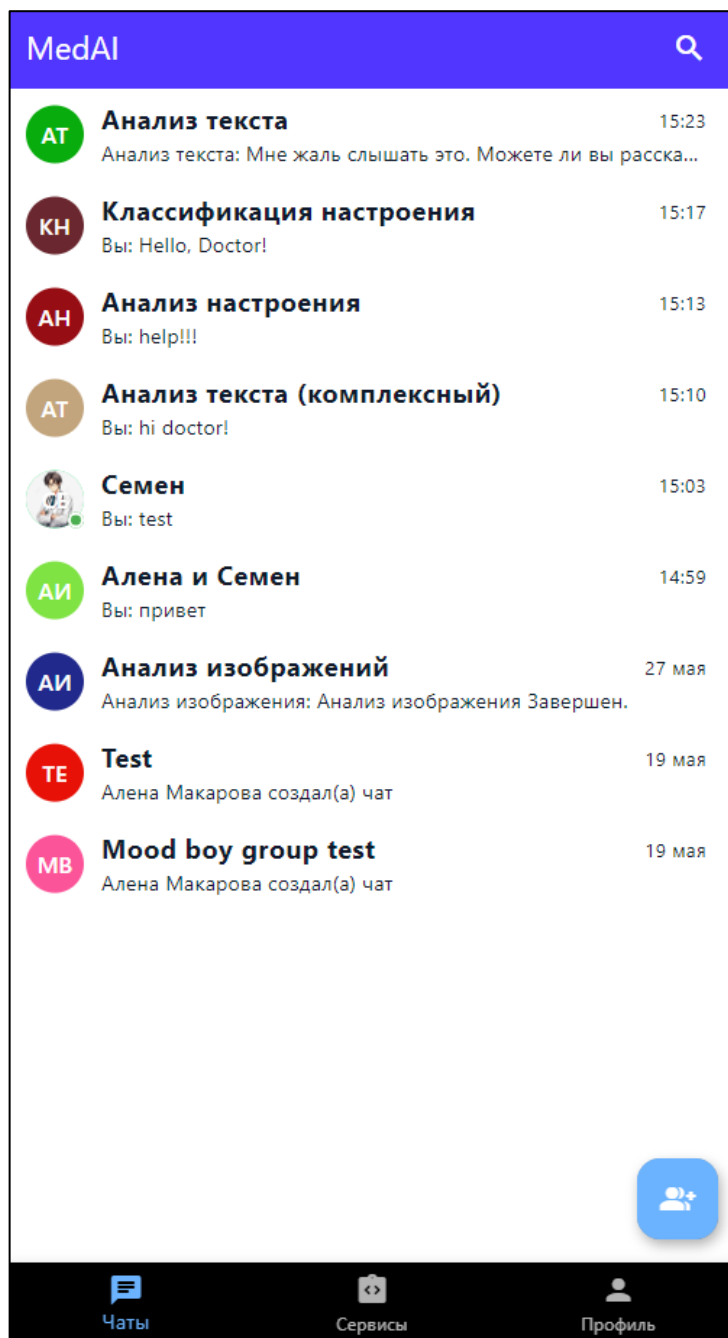


Рисунок 25 – Список чатов

На вкладке «Профиль» пользователь может видеть свои данные аккаунта и аватар. При желании, пользователь может изменить свои данные или загрузить новое изображение с устройства в качестве аватара.

При необходимости может быть совершен выход из аккаунта или переход на один из следующих экранов:

- список заболеваний. Экран необходим для изменения состояния болезней человека, в случае выздоровления, болезнь можно убрать из списка. Данный экран используется и для добавления болезней. Это также можно сделать в случае, если чат-бот определил болезнь по фотографии или текстовому описанию, однако перед этим действием рекомендуется пройти медицинское обследование для точного определения, иначе болезнь будет добавлена с пометкой «подозрение на наличие заболевания». Данный список заболеваний используется для уточнения запроса к нейронной сети, а также для визуального отображения,

- виртуальный аватар. Данный экран используется для визуализации болезней и особенностей организма человека на виртуальном теле. Таким образом пользователь сможет отслеживать изменение состояния своего здоровья, а также наглядно видеть пораженные участки. Для отрисовки используются список болезней, который задан пользователями на отдельном экране,

- прием лекарств. Этот экран необходим для просмотра настроенных уведомлений для напоминаний о приеме таблеток или необходимости сделать инъекцию. Также на этом экране можно выполнить создание, изменения или удаления уведомления,

- удаление данных аккаунта. На этом экране пользователь может полностью удалить всю информацию своего аккаунта с сервера. Также при желании можно удалить и сам аккаунт. После завершения пользователю покажется уведомление, затем будет выполнен переход на экран с регистрацией аккаунта.

Интерфейс профиля представлен на рисунке 26.

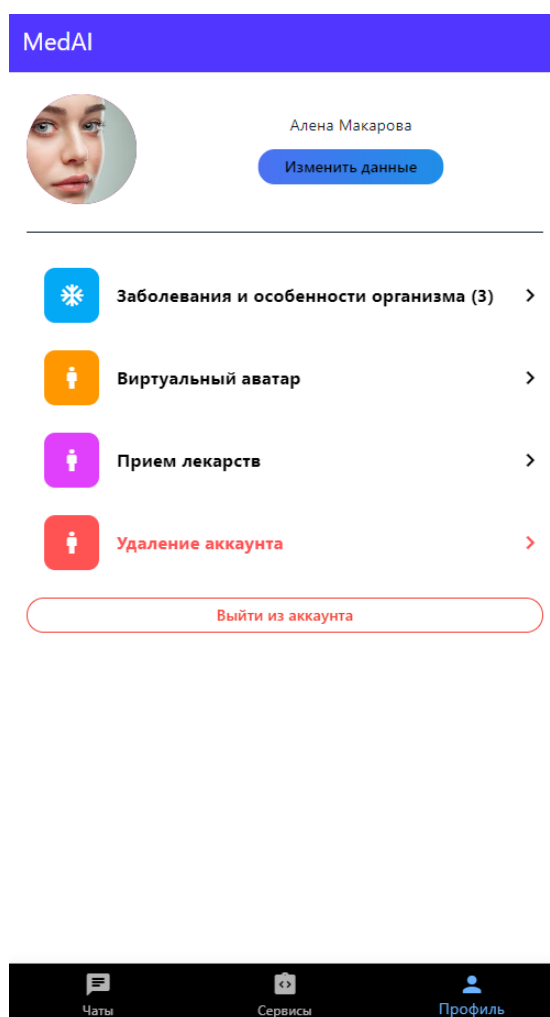


Рисунок 26 – Профиль пользователя

Следующий экран, представленный в приложении – «Чат». На данном экране есть возможность писать и отправлять сообщения, а также прикреплять файлы с устройства. В каждом чате присутствует кнопка для перехода в конец диалога. Верхняя панель чата показывает собеседника, диалог с которым отображен на экране в данный момент. Также здесь отображается информация о том, какой тип чата представлен (приватный или группа).

После отправки сообщения, оно проходит обработку на сервере, перенаправляется конкретному чат-боту в случае необходимости, анализируется нейронными сетями, тем самым формируя результат. Далее из полученной информации формируется новое сообщение, которое отправляется пользователю.

В случае, когда сообщение направлено конкретному пользователю или группе, на сервере пропускается этап обработки и сообщение сразу отправляется на сокет всем участникам чата. Перед этими действиями сообщение заносится в базу данных.

Для удобства тестирования и проверки работоспособности нейронных сетей и отдельных подсистем, было принято решение временно выделить их в отдельные чат-боты. Таким образом, всего в системе присутствуют 4 чат-бота:

- чат-бот для анализа изображений. Бот анализирует отправленную картинку пользователем и ищет на ней пораженные заболеванием участки. в случае обнаружения болезни, формируется ответное сообщение с ее названием и вероятностью. В случае отсутствия пользователь также будет оповещен,

- чат-бот для определения класса настроения. Данный бот анализируется текстовое сообщение от пользователя. Данные проходят через нейронную сеть, выдавая на выходе результат – класс настроения. Данный результат выводится в ответном сообщении,

- чат-бот для анализа текста. Данная система основана на использовании разработанной нейронной сети для генерации текста. Таким образом имитируется диалог между пациентом и виртуальным помощником. Чат-бот выдает необходимые рекомендации пользователю в зависимости от описания его состояния, передающееся в сообщении. Сформированный результат с рекомендациями отправляется на сокет,

- чат-бот для комплексного анализа. Это комплексный чат-бот, включающий в себя взаимодействие с предыдущими вышеописанными подсистемами. Используется цепочка нейронных сетей, заканчивающаяся генератором текста. На каждом шаге результат выполнения переходит в следующую нейронную сеть, в конце попадая в генератор. Это сделано для того, чтобы максимально улучшить запрос к нейронной сети, включив максимально подробную информацию о состоянии пользователя. Такой подход позволит чат-боту ответить в соответствующем стиле, в зависимости

от настроения пользователя. Нейронная сеть, анализирующая изображения поможет сократить количество возможных болезней и дать более точную рекомендацию для решения проблемы.

Взаимодействие с чат-ботом для анализа текста представлено на рисунке 27 ниже.

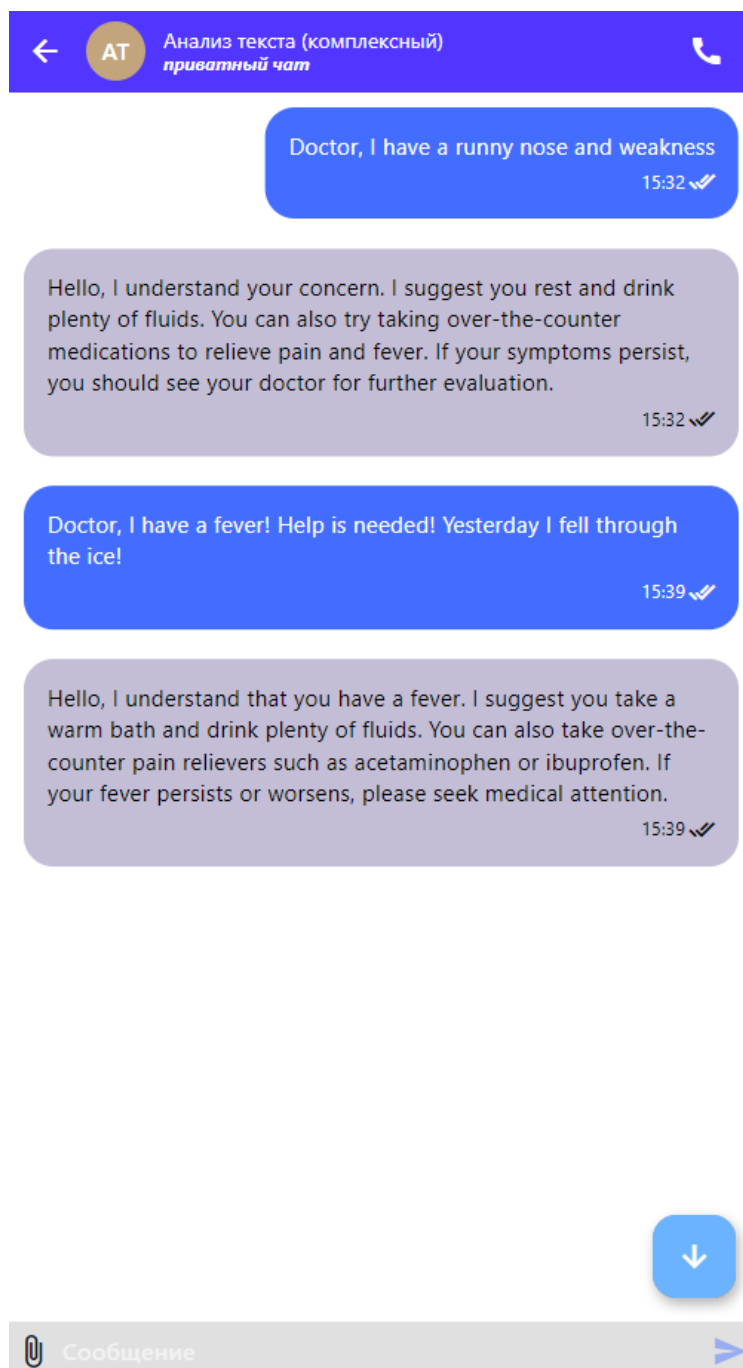


Рисунок 27 – Анализ текста

Интерфейс бота для анализа изображений представлен на рисунке 28.

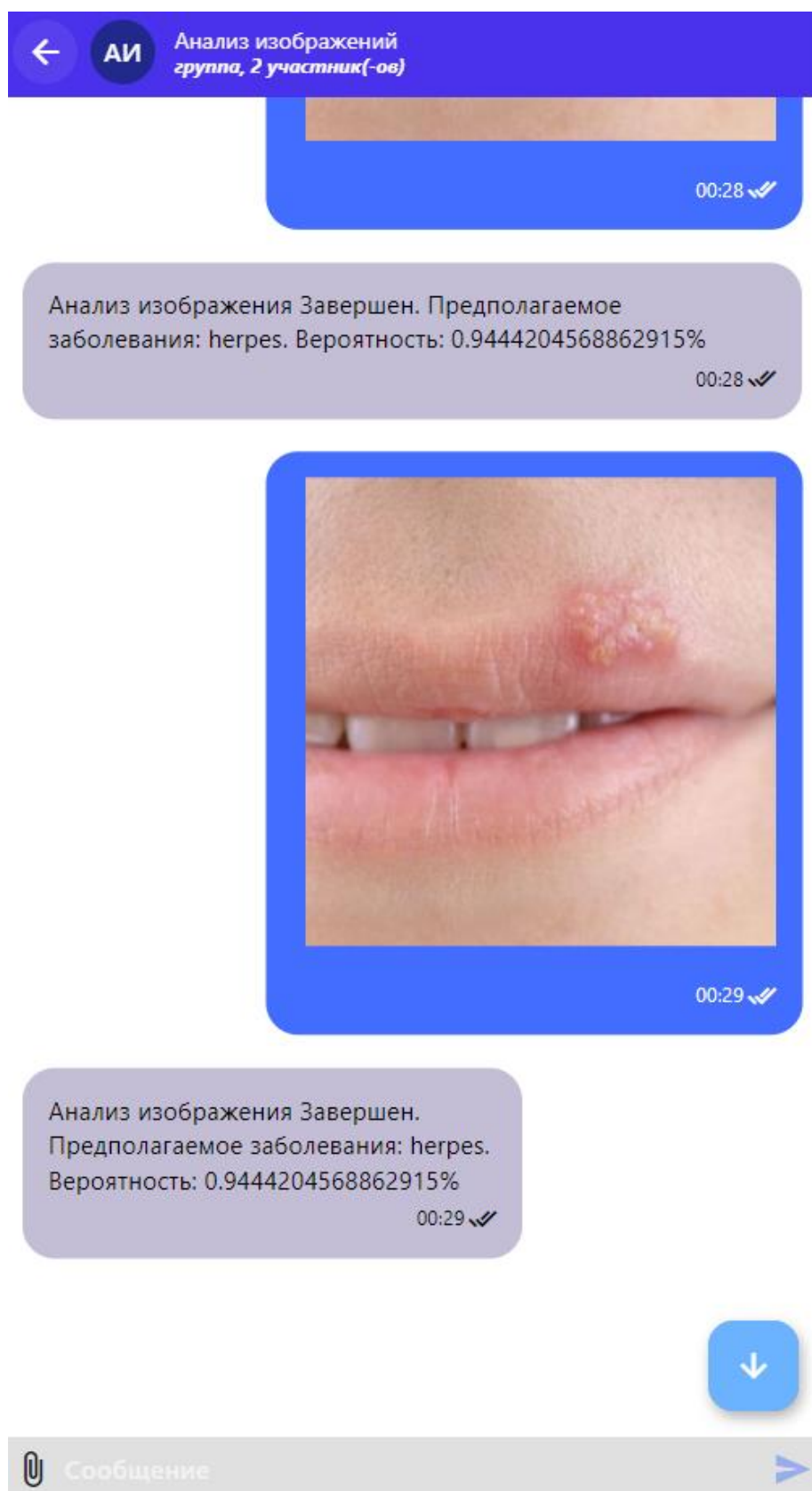


Рисунок 28 – Анализ изображения

Интерфейс чат-бота, который выполняет анализ настроения, представлен на рисунке 29 ниже.

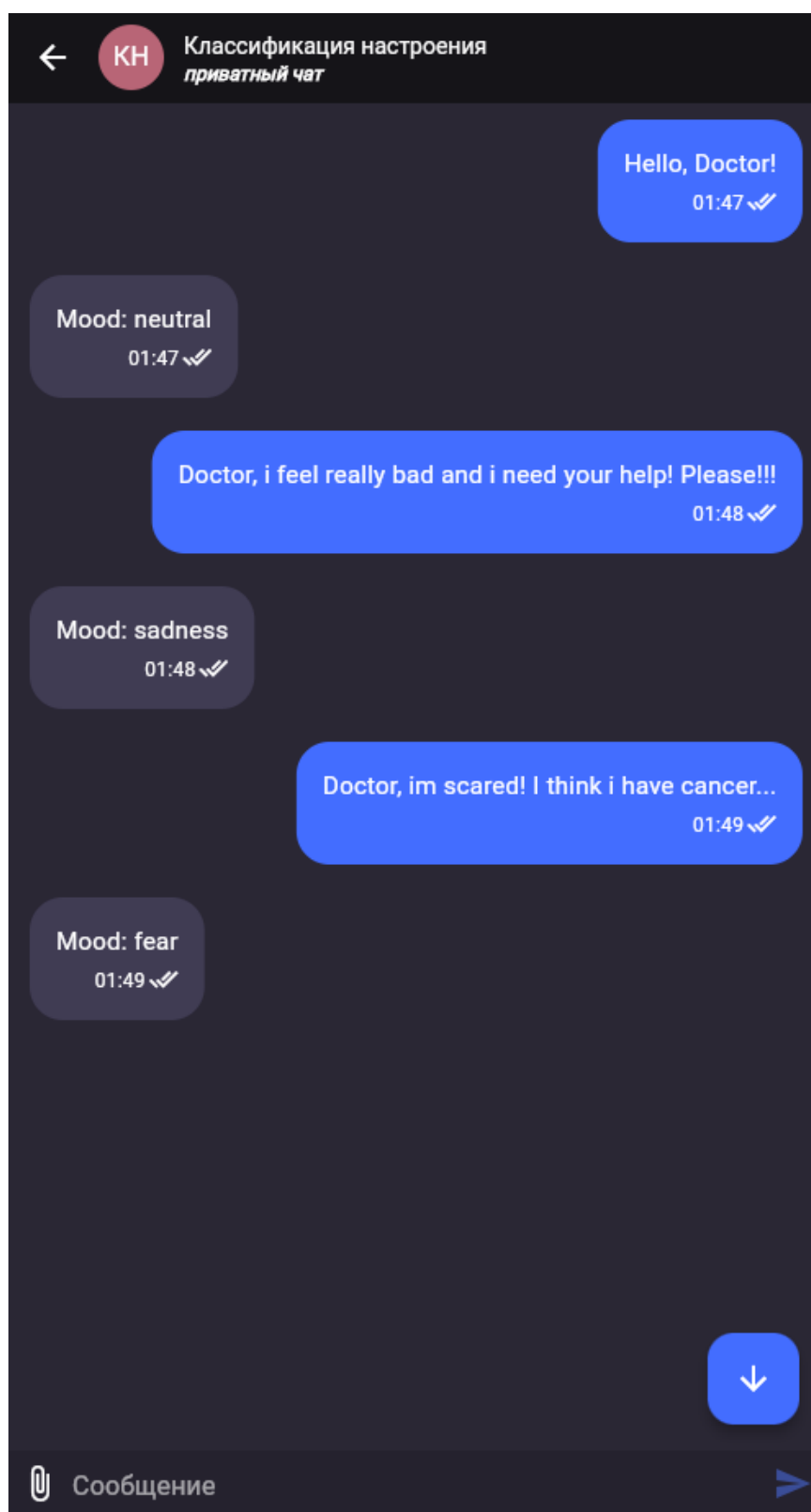


Рисунок 29 – Анализ настроения

При нажатии на иконку с лупой на экране с чатами, выполняется переход на экран поиска. Тут можно выполнить поиск пользователей или ботов и начать диалог с ними. После нажатия на элемент списка происходит переход в соответствующий чат. Интерфейс представлен на рисунке 30 ниже.

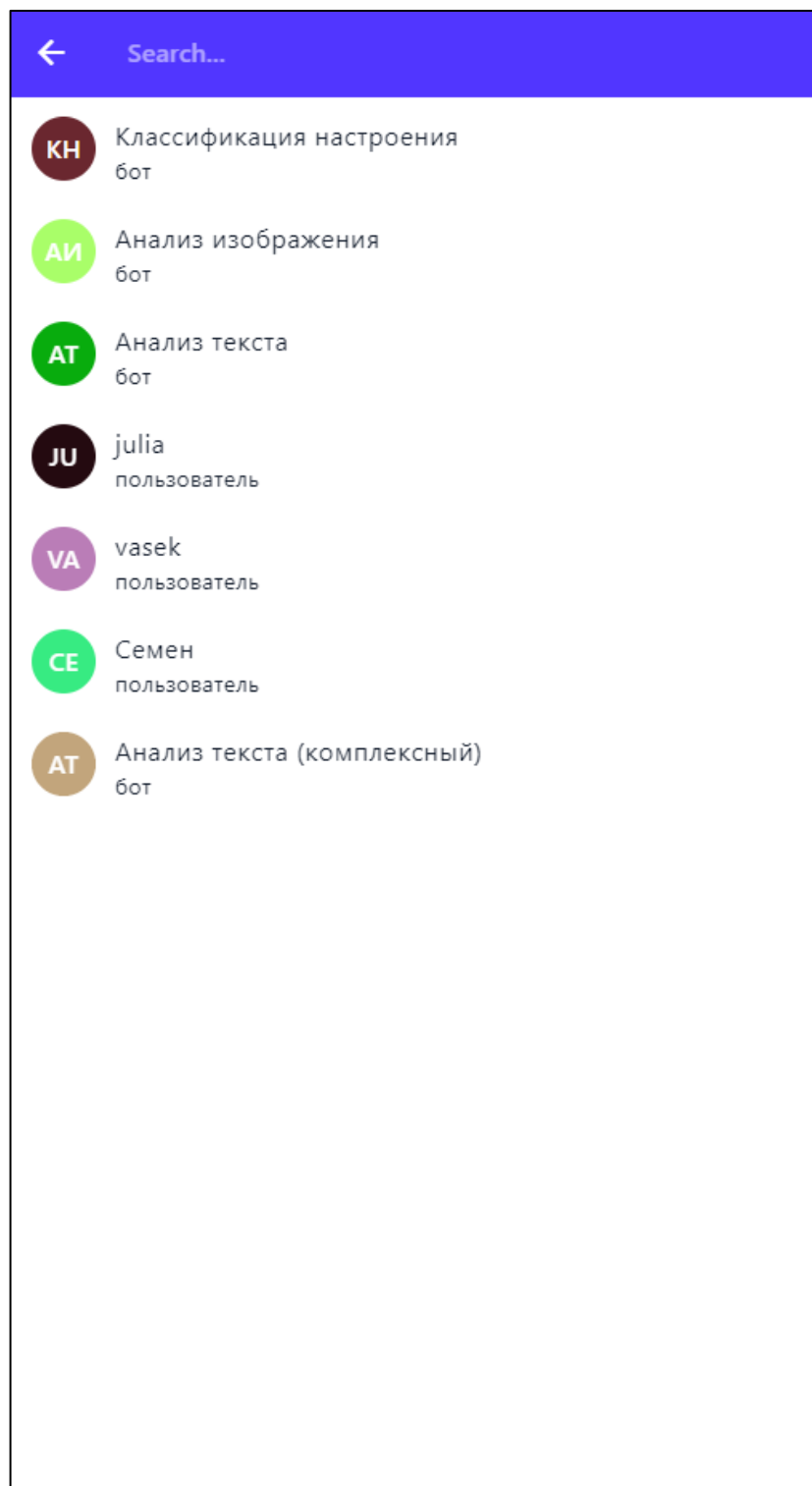
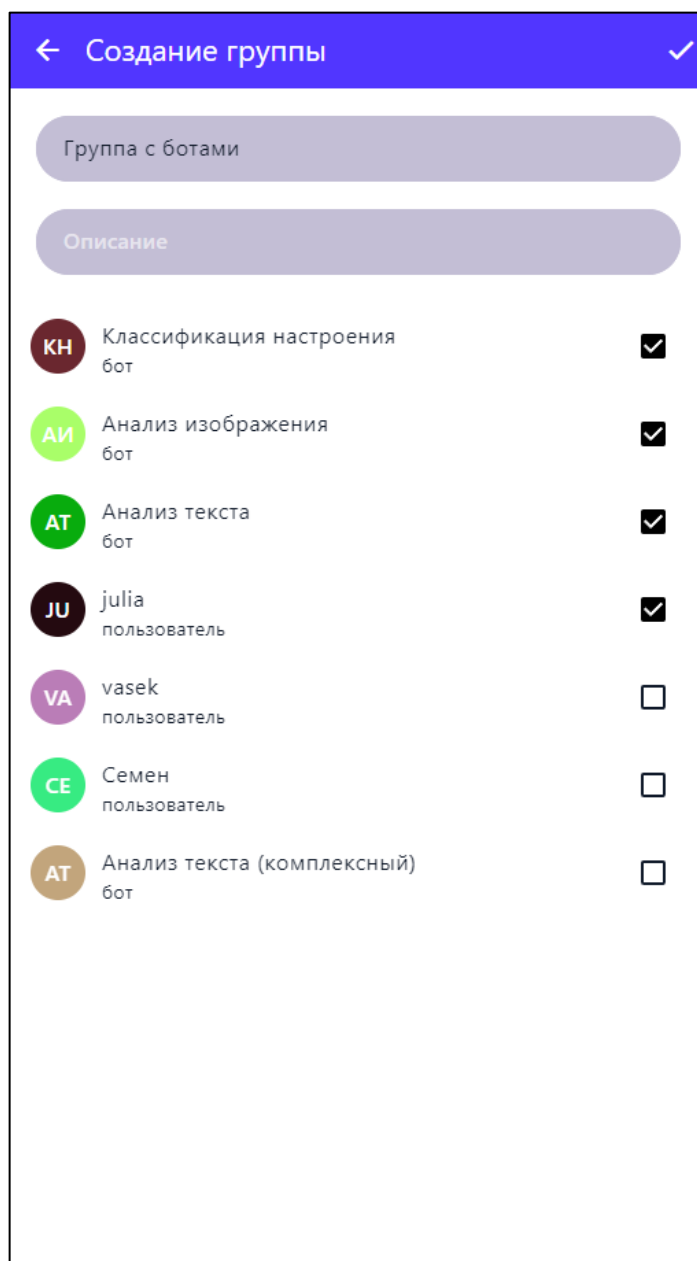


Рисунок 30 – Поиск пользователей

Помимо чат-ботов и переписок с другими пользователями, существует возможность создать группу, включив туда пользователей или ботов. Для группы возможно выбрать название и задать описание. После ввода всех необходимых данных кнопка для создания группы с галочкой станет активной. После нажатия на нее будет отправлен запрос на сервер и выполнится перенаправление на экран чата. Все пользователи, являющихся участниками группы получают соответствующее сообщение на сокет. Интерфейс представлен на рисунке 31.



Avatar	Name	Role	Selected
КН	Классификация настроения	бот	☒
АИ	Анализ изображения	бот	☒
АТ	Анализ текста	бот	☒
JU	julia	пользователь	☒
VA	vasek	пользователь	☐
CE	Семен	пользователь	☐
АТ	Анализ текста (комплексный)	бот	☐

Рисунок 31 – Создание группы

3.3 Виртуальный аватар

Данный экран представляет собой «карту человеческого тела». На аватаре отображаются все текущие болезни пользователя. Данную функцию можно использовать для отслеживания состояния здоровья человека. При наличии серьезных заболеваний, которые могут угрожать жизни человека, будет выводиться уведомления с рекомендацией срочного посещения медицинского учреждения.

Все заболевания четко подразделены по месту локализации для выполнения их визуализации. Например, болезни, поражающие мозг человека, будут отображаться в зоне головы.

Информация о каждом заболевании с местом локализации хранится в базе данных, формирование которой происходит через панель администратора.

При нажатии на болезнь, которая отображается на аватаре выполнится переход на экран «Заболевания и особенности организма», на котором можно изменить состояние всех болезней.

Визуализация выполняется с помощью отрисовки на canvas. В процессе отрисовки для каждой болезни вычисляется позиция, откуда и будет рисоваться начальная линия. Само изображение аватара было смещено влево для удобного отображения текста.

При изменении списка заболеваний, виртуальный аватар также обновится соответствующим образом. После того, как человек перенес болезнь, она переходит в «неактивную фазу». Такого болезнь по-прежнему будет отображаться в списке, но не будет отображаться на аватаре.

В будущем визуальную составляющую реализованной системы можно улучшить, сделав детальное отображение на отдельных слоях тела. Таким образом, можно будет отображать поражения отдельных органов, кровеносной и костной системы, а также внешних частях тела.

Интерфейс экрана с виртуальным аватаром тела человека представлен на рисунке 32 ниже.

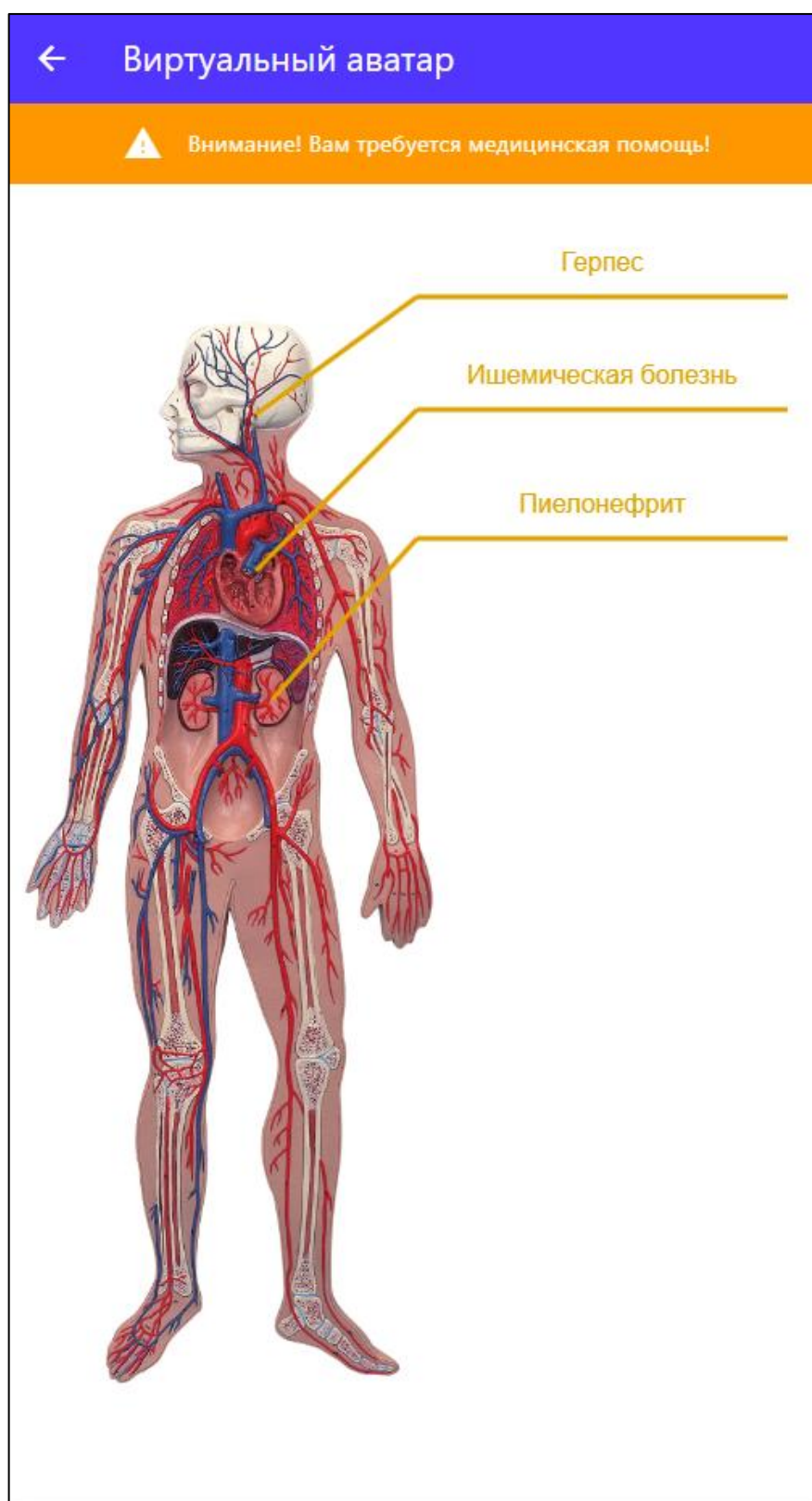


Рисунок 32 – Виртуальный аватар человека

3.4 Сервисы

Чтобы получить более точный результат от нейронной сети, необходимо собрать о человеке как можно больше данных. Эти данные в последствии будут подмешиваться в запрос, тем самым формируя уточняющие пояснения к запросу. Таким образом, нейронная сеть будет давать более точные рекомендации пользователю, подстраиваясь под каждого человека индивидуально.

Для получения большего количества данных, можно выполнить интеграцию с различными сервисами или технологиями. К ним могут относиться:

- приложения, позволяющие выполнять контроль за своим здоровьем с помощью встроенных средств. К таким относятся приложения-шагомеры, фитнес-сервисы, а также приложения для отслеживания питания человека,

- умные устройства. В качестве примера можно рассмотреть умные часы, которые могут измерять давление, пульс и процентное содержание кислорода в крови. Или умные весы, которые способны не только измерять вес человека, но и вычислять процент жира или мышечной массы,

- датчики (например, для контроля уровня потливости, температуры тела, мониторинга физической активности, отслеживание электрокардиограммы или других медицинских параметров),

- технологии умного дома. Сюда можно отнести роботов-помощников, которые могут улучшать эмоциональное состояние человека, автоматические системы-диспенсеры для выдачи лекарств по расписанию, умные холодильники, а также система, позволяющие отслеживать движение или присутствие для контроля сна и физической активности человека,

- медицинские и домашние лаборатории. Их можно использовать для сбора различных анализов пользователя,

- лаборатория на чипе. Устройства миниатюрного размера, позволяющее проводить различные химические и биохимические реакции,

выполнять лабораторную диагностику человека на наличие онкологических или других видов заболеваний.

Из всех технологий, самой доступной для максимально возможного числа пользователей является интеграция с приложениями. В качестве сервиса будет рассмотрено «Мобильное приложение для анализа и корректировки питания».

3.5 Мобильное приложение для анализа и корректировки питания

Разработанное приложение позволяет пользователям отслеживать потребление калорий, путем анализа фотографий с едой.

Система работает с помощью нейронных сетей, которые выполняют анализ изображения, выделяя все ингредиенты блюда и рассчитывая калорийность каждого. В результате для каждой фотографии блюда рассчитывается полная калорийность. Все данные хранятся на сервере.

Также в приложении присутствуют различные рецепты, описание отдельных ингредиентов с соответствующими параметрами белков, жиров и углеводов. Для корректной работы приложения пользователю необходимо ввести данные о своем организме: рост, вес, пол, физическая активность.

Эти параметры будут использоваться для расчета необходимого ежедневного потребления калорий. После выполнения необходимых расчетов, на главном экране приложения отобразится соответствующая плашка, показывающая количество калорий, которое уже употребил человек. Данные обновляются после каждой загрузки и анализа фотографии с едой.

Еще в приложении присутствует система, позволяющая вводить ежедневные измерения веса, таким образом, помогая пользователю отслеживать динамику изменений.

Это приложение полностью подходит для интеграции в медицинский сервис. С помощью собранных данных можно улучшить результат на выходе нейросети для анализа текста, подмешав в запрос данные о пользователе (к таким данным можно отнести, например, ожирение), что позволит сократить

количество возможных заболеваний и выдать более точную рекомендацию пользователю.

Более подробную информацию о данном ПО можно найти в соответствующем свидетельстве о регистрации программы [5].

Интерфейс главной вкладки, которая отображается сразу после открытия приложения, показан на рисунке 33 ниже.



Рисунок 33 – Основная страница приложения

На вкладке «Калории» пользователь может отслеживать продуктов, которые он употребляет. Интерфейс представлен на рисунке 34 ниже.

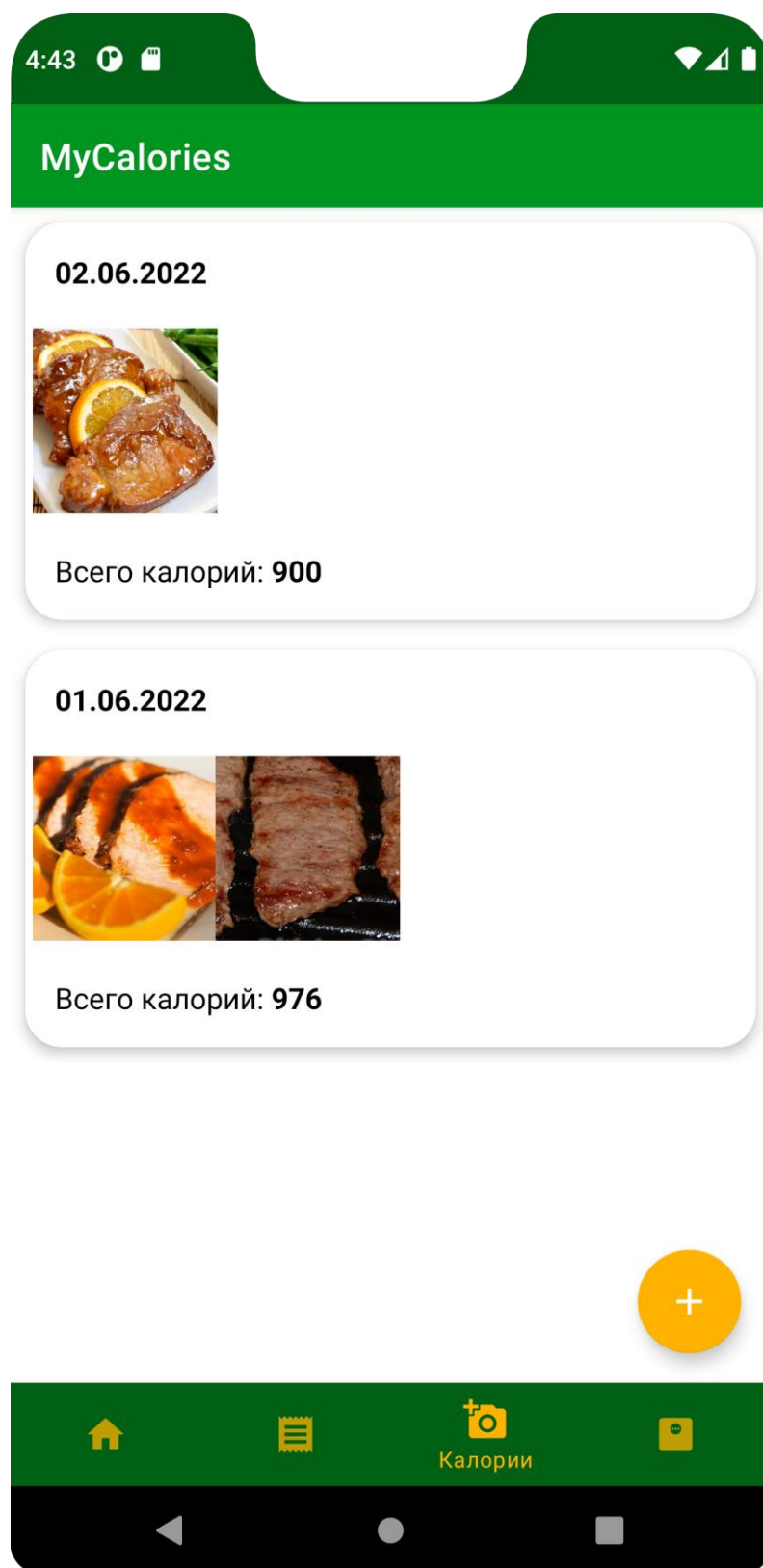


Рисунок 34 – Вкладка «Калории»

Интерфейс вкладки, на которой пользователь может осуществлять отслеживание веса, отображен на рисунке 35.



Рисунок 35 – Отслеживание веса

3.6 Вкладка «Сервисы»

На этой вкладке отображены все интеграции с сервисами. Здесь можно добавить интеграцию с описанным сервисом питания, после чего будет выполнен запрос к серверу для загрузки дополнительных данных пользователя.

В будущем можно добавить интеграцию с другими сервисами, например, с умными часами или весами, а также с сервисами-шагомерами для определения физической активности.

Интерфейс вкладки представлен на рисунке 36 ниже.



Рисунок 36 – Вкладка «Сервисы»

3.7 Выводы

В рамках данного раздела было разработано мобильное кроссплатформенное приложение, которое позволяет пользователям создавать приватные чаты и группы, участвовать в диалогах с чат-ботами, выполнять диагностику своего здоровья с помощью комплексного анализа на основе искусственного интеллекта.

Также была внедрена система для анализа питания пользователя. Такой подход позволяет получить больше данных о пользователе, что позволяет улучшить запрос, который отправляется нейронной сети, выдавая более точную и подходящую рекомендацию пользователю.

Создан виртуальный аватар человеческого тела, который позволяет визуализировать наличие заболеваний у пользователя, а также отслеживать изменение их состояние.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы был проведен анализ темы, рассмотрены аналогичные приложения и проекты, были сделаны соответствующие выводы. Также были сформированы основные требования для системы. Были проанализированы различные фреймворки для разработки мобильного приложения, веб-приложения, сервера и нейронной сети и составлены соответствующие сравнительные таблицы. Из рассматриваемых вариантов были выбраны наиболее подходящие для приложения. Был произведен анализ различных СУБД и была выбрана технология, соответствующая проекту. Рассмотрены различные инструменты, которые будут использоваться для создания системы.

Были разработаны сервер и клиентское мобильное приложение для Android и iOS, которые позволяют выполнять диагностику заболеваний с помощью технологий искусственного интеллекта. Для осуществления общения с чат-ботом с помощью текста была разработана модель нейронной сети для анализа текста от пользователя. Была реализована комплексная диагностика заболевания с помощью технологий искусственного интеллекта. Для диагностики используется подсистемы «Анализ изображения по фотографии», а также «Анализ настроения по тексту».

Для доступа к системе было успешно выполнено развертывание серверного приложения. Для избежания проблем с совместимостью был использован Docker и Docker-compose. В качестве сервиса для хостинга репозитория был использован сервис Github. После проделанных манипуляций сервер был полностью подготовлен для клиентских приложений.

В будущем такой проект можно расширить, добавив большее количество интеграций с различными сервисами и технологиями, тем самым увеличив количество данных о пользователе, которое будет получать нейронная сеть для выдачи более релевантной рекомендации. Также при наличии больших вычислительных мощностей, можно заменить текущую реализацию

нейронной сети, используя более современную и эффективную модель с большим числом параметров.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кантелон Майк, Алекс Янг, Брэдли Мек. Node.js в действии. – СПб.: Питер, 2018 г. – 432 с.
2. Эрик Уиндмил. Flutter in Action. – Manning Publications Co., 2020 г. – 368 с.
3. Максимова Д. Р., Коротов В.О., Митрохин М.А – ПЕРВИЧНАЯ ДИАГНОСТИКА ЗАБОЛЕВАНИЙ ПО ФОТОГРАФИИ ПРИ ПОМОЩИ НЕЙРОННЫХ СЕТЕЙ // Сборник научных статей по материалам XIX Международной научно-технической конференции, посвященной 75-летию кафедры «Вычислительная техника» ПГУ. Пенза, 2022. Издательство: Пензенский государственный университет (Пенза). С. 303-305. URL: <https://www.elibrary.ru/item.asp?id=50454563> (дата обращения: 31.05.2024).
4. Свидетельство о регистрации программы «Мобильное приложение для поиска медицинских учреждений. URL: https://www.fips.ru/iiss/document.xhtml?facesredirect=true&id=e1150a0aa0458b8_a29e89ec94048a591 (дата обращения: 31.05.2024).
5. Свидетельство о регистрации программы «Мобильное приложение для анализа и корректировки питания. URL: <https://www.fips.ru/iiss/document.xhtml?faces-redirect=true&id=fce017e2d07f3851d7c8ba4f8ba4aa12> (дата обращения 31.05.2024).
6. Ян Пойнтер. Программируем с PyTorch: Создание приложений глубокого обучения. – СПб.: Питер, 2020 г. – 256 с.
7. Майкл Авенди. PyTorch Computer Vision Cookbook. – Packt Publishing, 2020 г. – 364 с.
8. Моуэт Э. Использование Docker – М.: ДМК Пресс, 2017 г. – 354 с.: ил.
9. Кочер П.С. Микросервисы и контейнеры Docker. – М.: ДМК Пресс., 2019 г. – 240 с.
10. Колисниченко Д.Н.. Linux. От новичка к профессионалу. – СПб.: БХВ-Петербург, 2010 г. – 784 с.

11. Клинтон Дэвид. Linux в действии. – СПб.: Питер, 2019 г. – 416 с.
12. Немец, Эви, Снайдер, Гарт, Хейн, Трент, Уэйли, Бэн. Unix и Linux: руководство системного администратора. – М.: ООО «И.Д. Вильямс», 2017 г. – 1312 с.

ПРИЛОЖЕНИЕ А

Листинг исходного кода

```
import torch
from peft import (
    PeftConfig,
    PeftModel
)
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig
)
import os
from dotenv import load_dotenv

load_dotenv()

BASE_MODEL_NAME = os.getenv('BASE_MODEL_NAME')
MY_MODEL_NAME = os.getenv('MY_MODEL_NAME')

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

config = PeftConfig.from_pretrained(MY_MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(
    config.base_model_name_or_path,
    return_dict=True,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
```

```

        low_cpu_mem_usage=True
    )

tokenizer=AutoTokenizer.from_pretrained(config.base_model_name_or_path)
tokenizer.pad_token = tokenizer.eos_token

model = PeftModel.from_pretrained(model, MY_MODEL_NAME)

generation_config = model.generation_config
generation_config.max_new_tokens = 200
generation_config.temperature = 0.7
generation_config.top_p = 0.7
generation_config.num_return_sequences = 1
generation_config.pad_token_id = tokenizer.eos_token_id
generation_config.eos_token_id = tokenizer.eos_token_id

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

def analyze_text(text):
    prompt = "<human>: " + text + "\n<assistant>:".strip()
    encoding = tokenizer(prompt, return_tensors="pt").to(device)
    with torch.inference_mode():
        outputs = model.generate(
            input_ids=encoding.input_ids,
            attention_mask=encoding.attention_mask,
            generation_config=generation_config
        )

    return tokenizer.decode(outputs[0],
skip_special_tokens=True)

```

ПРИЛОЖЕНИЕ Б

Презентация

Медицинское приложение. Чат-бот для взаимодействия с пользователем

Выполнил: студент гр. 22БВиМ1 Коротов В.О.

Руководитель: Митрохин М.А.

1

Цели проекта

- Диагностика человека с помощью чат-бота
- Отслеживание состояние здоровья
- Разработка медицинского приложения



2



Стек технологий

TYPESCRIPT, PYTHON, NODEJS, MONGODB, SOCKETIO, FLUTTER, DART



3



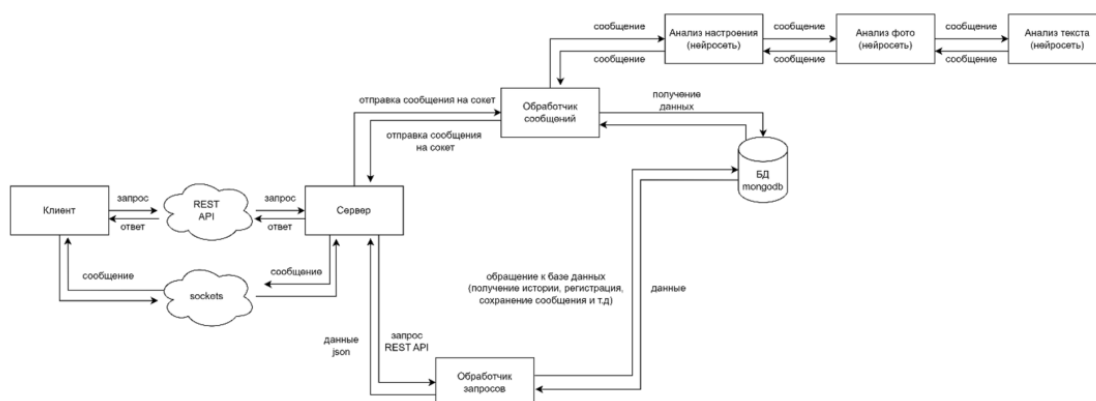
Взаимодействие с подсистемами

- Подсистема для анализа изображения. Выполняет диагностику по изображению
- Подсистема для анализа настроения. Анализирует текст сообщения, определяя настроение пользователя



4

Принцип работы

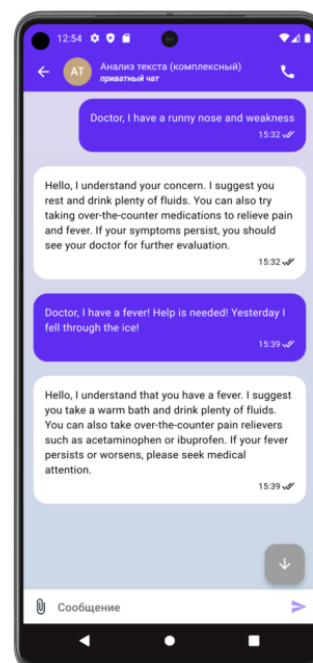


5

Результат работы чат-бота

- Комплексный анализ и выдача персональных рекомендаций
- Диагностика по фотографии
- Изменение оттенка сообщения в зависимости от настроения
- Взаимодействие в режиме реального времени

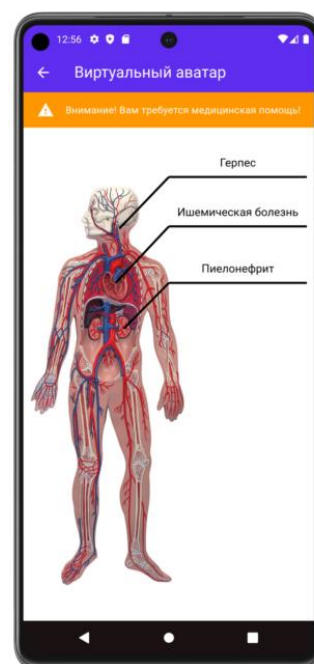
6



Виртуальный аватар

- Визуализация болезней
- Отслеживание состояния здоровья
- Оповещение о необходимом лечении

7



Сбор данных

- Умные устройства
- Датчики
- Лаборатории
- Приложения, связанные со здоровьем

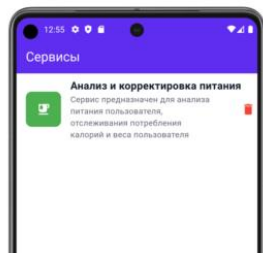
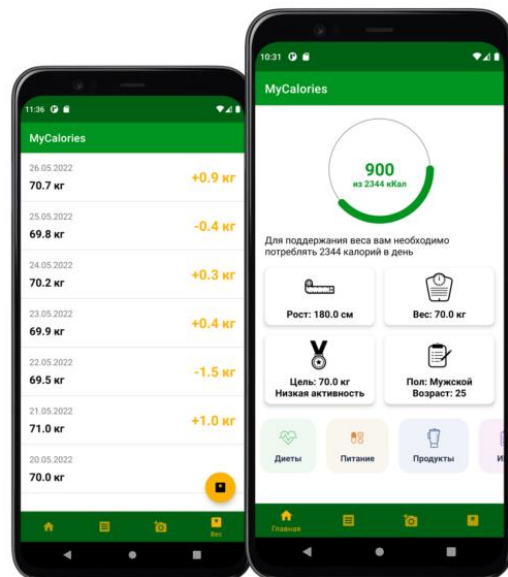


8

Сервис для анализа и корректировки питания

- Расчет калорий блюда по фотографии
- Контроль веса
- Отслеживание параметров здоровья

9



Заключение

Разработанная система не заменит врачей, но сможет помочь пользователям получить первичную медицинскую консультацию.

В дальнейшем можно добавить чат-бота диетолога, улучшить релевантность выдачи с помощью умных устройств и других технологий, применить систему совместно с медицинскими клиниками, улучшить визуализацию на виртуальном аватаре.

10