МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

| *Факультет* | *Кафедра* |
|---|---|
| вычислительной техники | Вычислительная техника |

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль «Вычислительные машины, комплексы, системы и сети»

# БАКАЛАВРСКАЯ РАБОТА

на тему

# «ГОЛОСОВОЙ АССИСТЕНТ ДЛЯ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА»

**Студент** 30.05.25 Кирьянов В.Е.
(подпись, дата)

**Руководитель** 30.05.25 Юрова О.В.
(подпись, дата)

**Нормоконтролёр** 30.05.25 Бычков А.С.
(подпись, дата)

*Работа допущена к защите* (протокол заседания кафедры от 28.05.2025 г. № 16)

**Заведующий кафедрой** _____ Митрохин М.А.
(подпись)

*Работа защищена с отметкой* отлично (протокол заседания ГЭК от 02.06.25 № 3)

**Секретарь ГЭК** _____ Гурин Е.И.
(подпись)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

| *Факультет* | *Кафедра* |
| --- | --- |
| вычислительной техники | «Вычислительная техника» |

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«6» 05 2025 г.

# ЗАДАНИЕ
# НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Кирьянов Владислав Евгеньевич гр. 21ВВВ1 факультета вычислительной техники.

2. Тема работы: «Голосовой ассистент для персонального компьютера»

Тема утверждена приказом ПГУ № 601/0 от "6" 05 2025 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Юрова О.В.

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

Необходимо создать голосового ассистента для персонального компьютера, который предназначен для повышения эффективности работы за компьютером. Программа должна выполнять голосовые команды пользователя.

Программа должна получать на вход речь пользователя, а после обрабатывать её и выполнять определённую команду.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ следующих предметных областей: изучить существующие методы и алгоритмы создания голосовых ассистентов, изучить существующие среды разработки и выбрать наиболее подходящую из них, а также изучить существующие модели распознавания. Необходимо рассмотреть вопросы создания архитектуры программы, реализации системы распознавания речи, а также системы синтеза речи.

6. Календарный график выполнения работы

| № п/п | Наименование этапов работы | Объем работы | Срок выполнения | Подпись руководителя |
|---|---|---|---|---|
| 1. | Изучение предметной области | 52 | 06.05.2025-09.05.2025 | |
| 2. | Анализ требований | 13 | 10.05.2025 | |
| 3. | Выбор технологий, применяемых при проектировании и разработке | 52 | 11.05.2025-14.05.2025 | |
| 4. | Описание алгоритмов | 39 | 15.05.2025-17.05.2025 | |
| 5. | Разработка голосового ассистента для персонального компьютера | 104 | 18.05.2025 - 25.05.2025 | |
| 6. | Оформление пояснительной записки | 64 | 26.05.2025 - 30.05.2025 | |

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы 30.05.25 Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент 6.05.25 Кирьянов В.Е.
(подпись, дата)

Работу к защите допустить
Декан факультета 30.05.2025 Фионова Л.Р.
(подпись, дата)

## Реферат

Бакалаврская работа содержит 76 листов, 17 рисунков, 4 таблицы, 14 источников и 3 приложения.

ГОЛОСОВОЙ АССИСТЕНТ, РАСПОЗНАВАНИЕ РЕЧИ, СИНТЕЗ РЕЧИ, КОМАНДНОЕ УПРАВЛЕНИЕ, МОДЕЛЬ РАСПОЗНАВАНИЯ

Цель работы заключается в разработке голосового ассистента, который распознаёт и синтезирует речь, выполняет одну из перечисленных команд, которую задаёт пользователь голосовым вводом: вывод информации о прогнозе погоды, открытие внешнего приложения, поиск запроса в браузере, поиск термина в википедии, перевод фразы и генерацию случайного числа в заданном диапазоне.

Объектом исследования в работе являются системы распознавания и синтеза речи, для дальнейшей реализации распознанного текста.

В работе применены методы разработки голосовых ассистентов с использованием языка Python, а также программная реализация командного управления.

В работе рассмотрены существующие системы распознавания и синтеза речи, проведён обзор аналогов голосового ассистента, протестированы различные среды разработки и готовые решения систем, а также определен набор наилучших свойств, необходимых подобной системе, для применения в разрабатываемой программе.

**Содержание**

Перечень сокращений и обозначений ..... 4
Введение ..... 5
1 Анализ предметной области ..... 6
1.1 Анализ аналогов ..... 6
1.2 Выбор инструментов разработки ..... 9
1.3 Выбор среды разработки ..... 12
1.4 Постановка задачи ..... 13
2 Проектирование и разработка ..... 16
2.1 Архитектура приложения ..... 16
2.2 Выбор модели распознавания ..... 19
2.3 Разработка пользовательского интерфейса ..... 23
2.4 Разработка ядра программы ..... 25
2.5 Разработка команд голосового ассистента ..... 29
3 Тестирование и анализ результатов ..... 33
4 Охрана труда ..... 39
4.1 Характеристика потенциальных рисков при работе с ПК ..... 39
4.2 Оборудование рабочего места ..... 39
4.3 Освещённость рабочего места ..... 40
4.4 Электробезопасность ..... 41
4.5 Пожарная безопасность ..... 41
4.6 Профилактические меры для снижения утомления глаз ..... 42
Заключение ..... 43
Список используемых источников ..... 44
Приложение А - UML – диаграммы программы ..... 46
Приложение Б - Листинг программы ..... 52
Приложение В - Презентация ..... 72

## Перечень сокращений и обозначений

ПК – персональный компьютер

API (Application Programming Interface) – программный интерфейс приложения

ASR (Automatic Speech Recognition) – автоматическое распознавание речи

CPU (Central Processing Unit) – центральный процессор

GPU (Graphics Processing Unit) – графический процессор

OpenAI (Open Artificial Intelligence) – открытый искусственный интеллект

TTS (Text-to-Speech) – преобразование текста в речь

## Введение

Голосовые ассистенты упрощают взаимодействие пользователя с компьютером или телефоном. Они позволяют управлять программами, искать информацию в интернете и выполнять повседневные задачи при помощи голосовых команд. Использование таких ассистентов в повседневной работе за персональным компьютером даёт возможности автоматизировать рутинные задачи, тем самым экономя время и ресурсы человека. Помимо стандартного функционала голосовые помощники могут выполнять сложные запросы и задачи. Например, они могут диктовать текст, осуществлять его перевод, составлять расписание, создавать напоминания, управлять умным домом и выполнять множество других прикладных задач.

Основная функция голосовых помощников – это обработка человеческой речи с помощью механизмов распознавания ASR и синтеза речи TTS. Они анализируют запросы пользователя и после обработки выдают результат. Некоторые ассистенты используют нейронные сети для точного распознавания и обработки голоса пользователя. Популярные помощники используют нейронные сети, не только для распознавания и синтеза речи, но и для общения с виртуальным помощником, преобразования текста в речь, генерации ответа, текста или даже изображения.

Автоматизация повседневных процессов в жизни человека имеет важное значение, так как это существенно экономит время, повышает эффективность выполнения задач и позволяет упростить управление своим устройством. И как следствие из этого, голосовые ассистенты пользуются популярностью. Таких помощников могут использовать как обычные пользователи, так и пользователи с ограниченными возможностями.

# 1 Анализ предметной области

Голосовые помощники существенно экономят время, повышают продуктивность и позволяют составлять расписание своих задач, поэтому они могут быть полезными занятым людям, бизнесменам, студентам, водителям. Также они будут полезны людям с ограниченными возможностями, детям и школьникам, чтобы задавать вопросы, получать новую информацию, слушать сказки, учить языки и др., а также домохозяйкам и родителям. Таких ассистентов может использовать каждый, так как они подходят не только для специализированных, но и для повседневных задач.

Исследование [1] по изучению взаимосвязи между голосовыми помощниками и эффективностью обучения студентов показало, что использование голосовых ассистентов положительно влияет на обучаемость и продуктивность студентов, но это относилось только к индивидуальной работе обучающихся, то есть не распространялось на групповую работу или взаимодействие с преподавателем. Повышение индивидуальной эффективности обучения студентов является подтверждением того, что использование такого помощника улучшает усвоение материала и повышает мотивацию учащихся.

## 1.1 Анализ аналогов

Существует множество голосовых помощников с разнообразным функционалом, вот наиболее распространённые из них:

Голосовой ассистент Алиса [2], разработанный компанией «Яндекс». Распознаёт естественную речь, имитирует живой диалог, даёт ответы на вопросы пользователя и позволяет решать некоторые прикладные задачи.

К преимуществам данного помощника относятся:

– доступность на различных устройствах;

– интеграция с другими сервисами Яндекса;

– поддержка умного дома;

– умение поддерживать диалог.

К недостаткам можно отнести:

– зависимость от интернета;

– ограниченная поддержка языков;

– ошибки в распознавании речи.

Голосовой помощник Siri [3] от компании Apple. Поддерживает звонки и сообщения, позволяет решать ежедневные задачи и отвечает на вопросы пользователя.

К достоинствам можно отнести:

– контроль устройства голосом;

– поддержка умного дома;

– доступ к некоторым функциям без интернета;

– хорошее распознавание речи;

– интеграция с приложениями Apple;

– поддерживает множество языков;

– высокая степень конфиденциальности.

Недостатки помощника:

– плохо справляется со сложными запросами;

– плохо работает со сторонними сервисами;

– неточность ответов;

– не работает на других платформах.

Компания Google [4] разрабатывает голосового ассистента Google Assistant, который предоставляет быстрый поиск информации и возможность управлять различными функциями мобильного устройства.

Достоинствами являются:

– интеграция с сервисами Google;

– кроссплатформенность;

– управление умным домом;

– высокое понимание контекста;

– поддержка большого количества языков.

К недостаткам можно отнести:

– зависимость от интернета;

– конфиденциальность и сбор данных;

– ограниченная поддержка русского языка.

Компания Microsoft занимается разработкой голосового помощника Cortana [5]. Он позволяет управлять задачами и расписанием, имеет интеллектуальный поиск и рекомендации, а также интеграции с другими сервисами Microsoft.

Достоинствами являются:

– интеграция с сервисами Microsoft;

– управление задачами и напоминаниями.

Недостатками является:

– работает только в Windows 10;

– интеграция только с сервисами Microsoft;

– не работает без интернета;

– больше не поддерживается Microsoft;

– только для бизнес-задач;

– ограниченный функционал;

– плохая поддержка русского языка.

Ниже представлена сводная таблица 1 из достоинств и недостатков каждого ассистента.

Таблица 1 – Сводная таблица достоинств и недостатков аналогов

| Голосовой ассистент | Разработчик | Достоинства | Недостатки |
|---|---|---|---|
| Алиса | Яндекс | Доступность на различных устройствах. Поддержка умного дома. Интеграция с другими сервисами Яндекса. | Зависимость от интернета. Ограниченная поддержка языков. |
| Siri | Apple | Контроль устройства голосом. Поддержка умного дома. Высокая степень конфиденциальности. | Некоторые функции недоступны без интернета. Интеграция только с приложениями Apple. Работает только на Apple-устройствах. |

| Голосовой ассистент | Разработчик | Достоинства | Недостатки |
|---|---|---|---|
| Google Assistant | Google | Кроссплатформенность.<br>Управление умным домом.<br>Поддержка большого количества языков. | Зависимость от интернета.<br>Конфиденциальность и сбор данных.<br>Ограниченная поддержка русского языка. |
| Cortana | Microsoft | Интеграция с сервисами Microsoft.<br>Управление задачами и напоминаниями. | Работает только в Windows 10.<br>Не работает без интернета.<br>Больше не поддерживается Microsoft.<br>Только для бизнес-задач.<br>Ограниченный функционал.<br>Плохая поддержка русского языка. |

На основе анализа достоинств и недостатков существующих ассистентов, было принято решение о необходимости разработки собственного голосового ассистента, который упростит пользование персональным компьютером, поскольку аналоги не имеют взаимодействия с персональным компьютером, либо имеют лишь минимальное взаимодействие.

## 1.2 Выбор инструментов разработки

Приложение предназначено для решения повседневных задач и управления некоторыми программами на персональном компьютере при помощи голосовых команд. Архитектура приложения будет построена по модульному принципу, что будет обеспечивать возможность дальнейшей модернизации проекта и расширения его функциональности.

Для создания голосового помощника подойдут такие языки программирования как: Python, C++ и C#.

C++ – это низкоуровневый язык программирования, который используется для решения разного спектра задач [6].

Достоинства:

– высокая производительность;

– управление памятью;

– обширная стандартная библиотека;

– кроссплатформенность.

Недостатки языка:

– сложность разработки программного обеспечения;

– малое количество библиотек распознавания речи и примеров с их использованием.

Ввиду этого, C++ отлично подходит для решения промышленных задач, требующих скорость и эффективность работы. Однако, он имеет низкий уровень абстракции, а также требует определённых навыков работы с ним и умение управлять памятью.

C# – это объектно-ориентированный язык программирования, который основан на языке C [7].

Достоинства языка:

– интеграция с Windows;

– наличие удобной среды разработки – Visual Studio.

Недостатки:

– малое количество библиотек распознавания;

– кроссплатформенность с ограничениями.

Таким образом, C# имеет умеренную сложность и производительность по сравнению с его аналогами, также отлично подходит для создания Windows-приложений. Однако, этот язык требует определённого опыта работы с ним и имеет ограниченную кроссплатформенность.

Python – это популярный, высокоуровневый и интерпретируемый язык программирования, который является достаточно простым в освоении [8].

Достоинства языка:

– эффективность разработки;

– большое количество библиотек для распознавания речи и примеров их использования;

– множество готовых API;

– кроссплатформенность.

Недостатки:

– невысокая производительность;

– большое потребление памяти.

Таким образом, Python отлично подходит для создания прототипа голосового ассистента, поскольку прост в изучении, имеет множество библиотек для распознавания речи, что упрощает разработку программы. Однако он требователен к ресурсам компьютера.

В таблице 2 представлена сводная информация о достоинствах и недостатках языков программирования.

Таблица 2 – Сводная таблица достоинств и недостатков языков программирования

| Язык программирования | Достоинства | Недостатки |
|---|---|---|
| Python | Эффективность разработки. Наличие множества библиотек распознавания. Кроссплатформенность. | Ресурсоёмкость. Невысокая производительность. |
| C++ | Высокая производительность. Возможность работы с памятью. | Малое количество библиотек распознавания. Требует определённых навыков работы. |
| C# | Хорошо работает с ОС Windows. Наличие хорошей среды разработки. | Ограниченная кроссплатформенность. Требует определённых навыков работы. |

На основании вышеизложенного, Python идеально подходит для данной задачи благодаря своей простоте, большому выбору библиотек и моделей распознавания. Этот язык программирования позволил реализовать и тестировать системы распознавания и синтеза речи при помощи существующих моделей, что было бы значительно сложнее сделать на таких языках, как C++ или C#.

Выбранный комплекс инструментов соответствует требованиям проекта, обеспечивая необходимый функционал для разработки прототипа голосового ассистента.

## 1.3 Выбор среды разработки

Для работы с Python подходят обычные текстовые редакторы: блокнот и Notepad++, но гораздо удобнее использовать специализированные среды разработки. Они предоставляют встроенные инструменты для проверки и отладки программ, управления библиотеками, интеграции с системами контроля версий и других функций, которые упрощают процесс разработки.

Наиболее популярные из них: PyCharm, Visual Studio Code, Jupyter Notebook.

PyCharm — это интегрированная среда разработки от JetBrains, специально созданная для Python. Поддерживает всё на встроенном уровне: автодополнение и анализ кода, инструменты для работы с базами данных и фреймворками. Подходит для различных проектов в сфере веб-разработки и машинного обучения.

VS Code — это редактор кода от Microsoft, который благодаря расширению Python становится полноценной средой разработки. Он поддерживает автодополнение, отладку, виртуальные окружения, управление зависимостями и терминал. Также можно добавить нужные функции через расширения и настроить интерфейс под себя.

Jupyter — это удобная среда для работы с кодом и анализа данных, визуализации и создания алгоритмов. Ячейки кода можно запускать по одной, вставлять графики, текстовые пояснения и формулы. Подходит для задач, где важен постепенный контроль результатов.

В таблице 3 представлена сводная информация о достоинствах и недостатках сред разработки.

Таблица 3 – Сводная таблица достоинств и недостатков сред разработки

| Среда разработки | Достоинства | Недостатки |
|---|---|---|
| Jupyter | Простота использования. Удобный вывод графической информация. | Не удобство разработки больших проектов. Отсутствие автодополнения. Плохая совместимость с github. |
| VS Code | Возможность работы с github. Поддержка различных плагинов. Бесплатное использование. Поддержка кроссплатформенности. | Требует настройки под Python. Не удобен для больших проектов. |
| PyCharm | Автоформатирование кода. Автодополнение. Различные инструменты для работы с базами данных. | Профессиональная версия является платной. Ресурсоёмкая среда разработки. |

Среда разработки Jupyter прекрасно подходит для анализа данных с их графическим отображением. VS Code – универсальная среда, подходящая для множества различных задач. Вместе с тем, PyCharm предназначена для Python-проектов и имеет расширенный функционал для разработки сложных систем. Таким образом, в качестве среды разработки была выбрана PyCharm Community.

## 1.4 Постановка задачи

Задача состоит в разработке программы для управления некоторыми функциями персонального компьютера при помощи голосового ввода и графического пользовательского интерфейса для настройки ассистента. Приложение должно быть основано на командном управлении, то есть ожидать команд от пользователя, для дальнейших действий. Также программа должна иметь возможность распознавать и синтезировать текст.

Программа должна иметь следующий функционал:

– распознавать и синтезировать естественную речь;

– предоставлять информацию о прогнозе погоды по запросу пользователя;

– открывать внешнее приложение на персональном компьютере по запросу пользователя;

– производить поисковой запрос в браузере по запросу пользователя;

– выполнять поиск определения в Wikipedia по запросу пользователя;

– переводить фразу по запросу пользователя;

– выбор случайного числа из заданного диапазона по запросу пользователя.

Программа должна отвечать следующим техническим требованиям:

– ОС Microsoft Windows 10 и выше;

– 64-разрядный процессор, не менее 2-ух ядер;

– оперативная память: не менее 4 Гб;

– дисковое пространство: не менее 2 Гб;

– видеокарта: интегрированная.

Ожидаемый результат:

Программа, позволяющая обрабатывать и синтезировать речь пользователя для выполнения его запроса. После открытия приложения ассистент должен выполнить приветствие и начать ожидать ключевое слово от пользователя.

Пользователь имеет возможность:

– получить информацию о погоде;

– перевести фразу;

– найти термин в Wikipedia;

– сгенерировать случайное число из диапазона, который задаст пользователь;

– выполнить поисковой запрос в браузере;

– открыть конкретное приложение на персональном компьютере;

– настроить ассистента через интерфейс.

Далее на рисунке 1 представлена UML-диаграмма вариантов использования, которая отображает доступный для пользователя функционал.

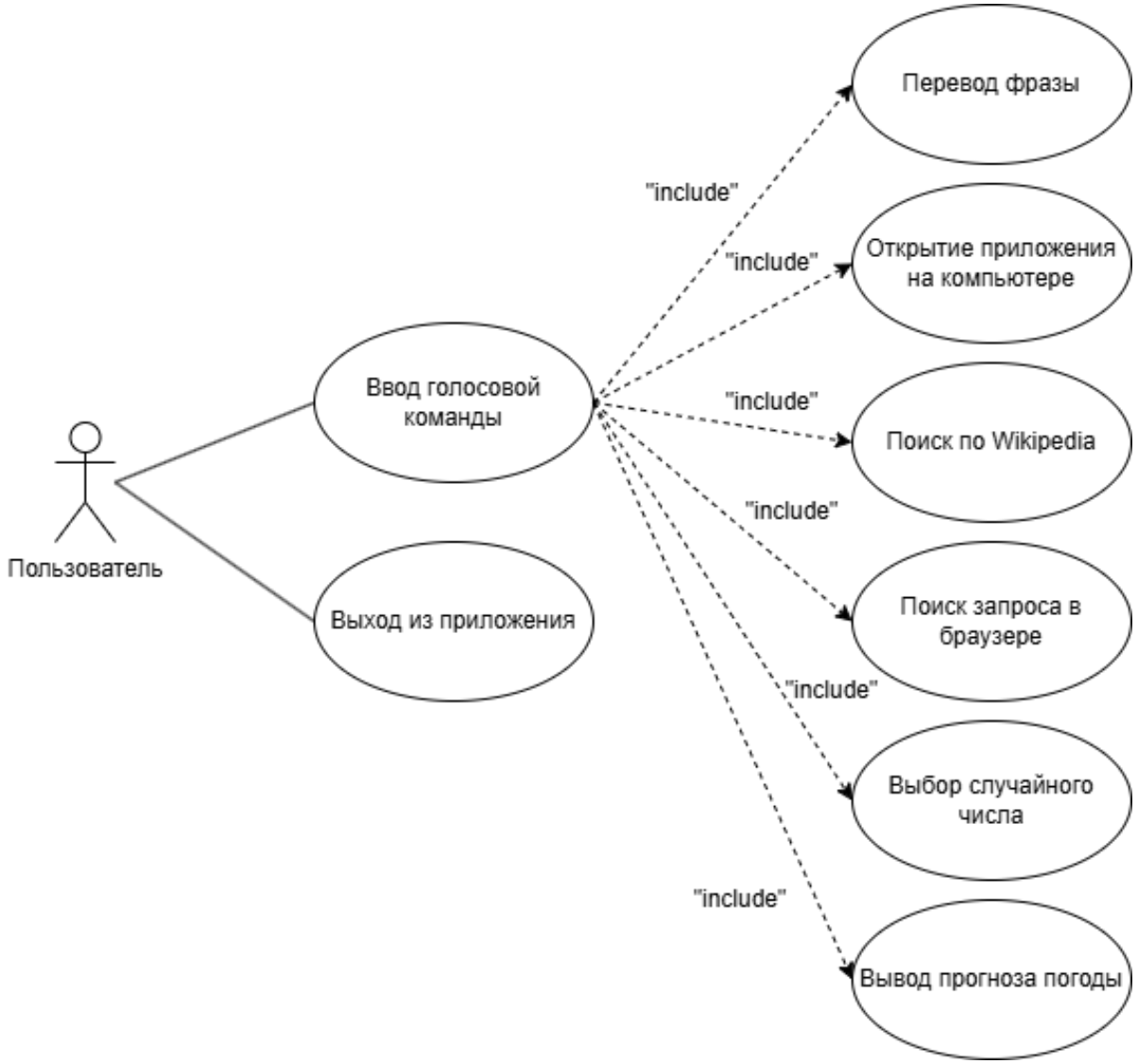



Рисунок 1 – UML-диаграмма вариантов использования

# 2 Проектирование и разработка

В данном разделе описаны: архитектура голосового ассистента, выбор модели распознавания, разработка модулей программы, а именно пользовательского интерфейса, ядра программы, команд голосового ассистента.

## 2.1 Архитектура приложения

Приложение предназначено для решения повседневных задач и управления некоторыми программами на персональном компьютере при помощи голосовых команд. Архитектура приложения будет построена по модульному принципу, что будет обеспечивать возможность дальнейшей модернизации проекта и расширения его функциональности.

Приложение состоит из нескольких компонентов, взаимодействующих друг с другом для организации работы голосового ассистента. Архитектура включает следующие основные блоки:

– пользовательский интерфейс;

– ядро программы;

– модуль обработки речи;

– модуль анализа текста и выполнения команд;

– вспомогательный модуль;

– блок команд.

Структура программы представлена на рисунке 2.

Голосовой ассистент является программой без основного интерфейса, так как он является встроенным и автономно работающим приложением и ввод команды ассистенту осуществляется при помощи речи. Однако голосовой помощник обладает вспомогательным интерфейсом, который предоставляет пользователю некоторую информацию о истории запросов и возможность настройки ассистента. Через графический интерфейс пользователю предоставляется информация о дате сессии, о введённой команде и о результате её выполнения. Также через окно настроек пользователь может задавать следующие параметры: громкость и скорость озвучивания ассистента, город по умолчанию,

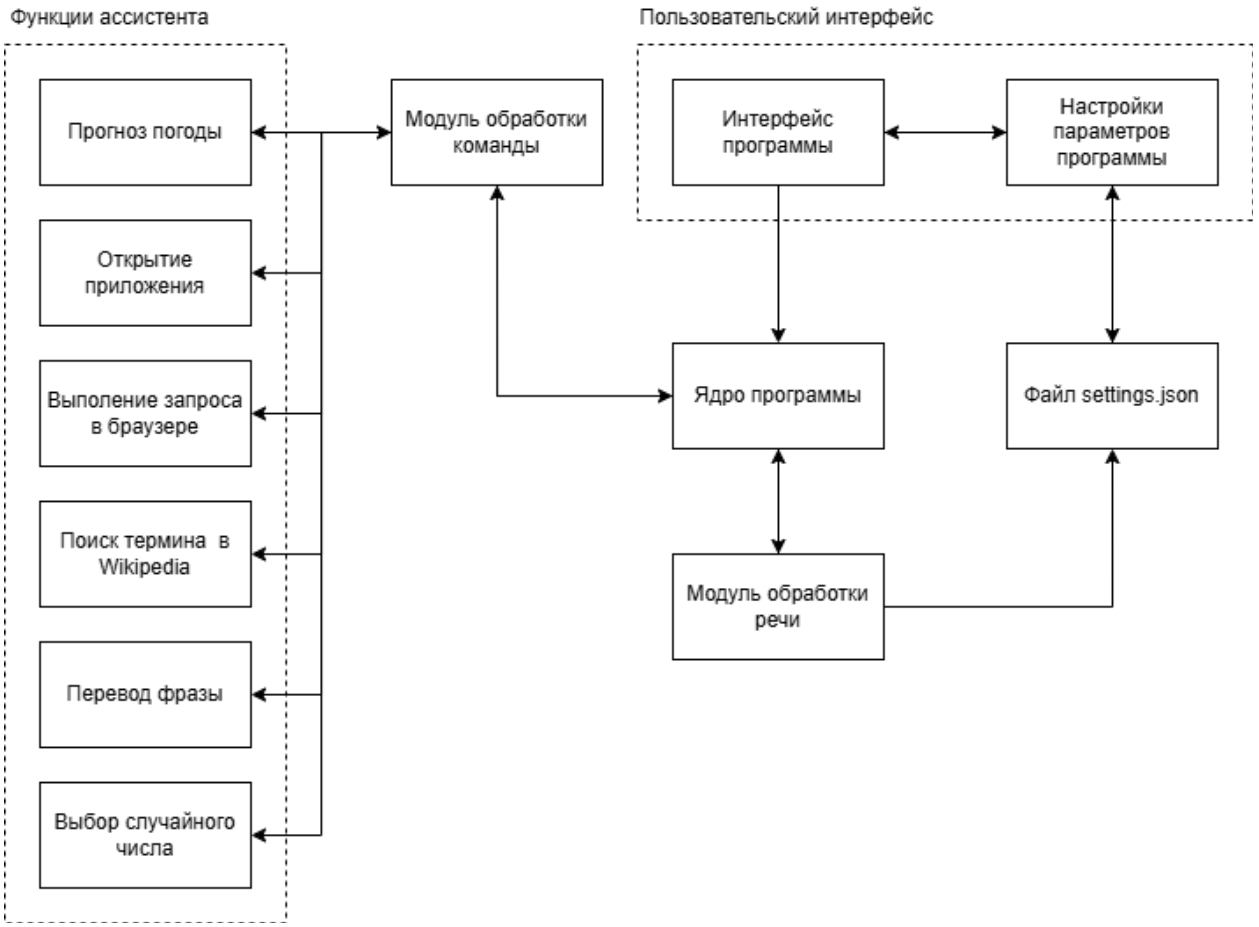



Рисунок 2 – Схема архитектуры программы

который будет использоваться для прогноза погоды без условия и язык модели. Эти параметры и API-ключи хранятся в конфигурационных файлах проекта.

Основу ассистента составляет ядро программы, которое отвечает за основную логику работы голосового помощника. Оно выполняет следующие функции:

– координирует действия между модулями, а именно записывает речь с микрофона, затем переводит её в текст, после чего происходит определение и выполнение команды и далее осуществляется озвучивание ответа;

– отвечает за проверку ключевого слова;

– управляет состоянием ассистента – активация и деактивация.

Модуль ответственный за обработку голоса выполняет следующие функции:

– запись речи, которая производится с помощью библиотек speech_recognition, vosk и pyaudio;

– синтез речи, который осуществляется с помощью библиотеки pyttsx3;

– воспроизведение звуковых сигналов при изменении состояния ассистента.

Модуль анализа текста и выполнения команд осуществляет следующие функции:

– извлечение ключевых слов для идентификации команды;

– определение команды для исполнения;

– обработка контекста с уточнением параметров;

– вызов соответствующей функции.

Вспомогательный модуль отвечает за подсветку определённых слов в журнале работы системы.

Каждая команда выполняется по соответствующему запросу пользователя. Модуль, отвечающий за прогноз погоды, отвечает за получение информации о погоде через API. Модуль, обеспечивающий открытие приложения, ищет на персональном компьютере определённую программу, названную пользователем, и открывает её. Модуль, ответственный за выполнение запроса в браузере, открывает браузер и осуществляет поиск запроса. Модуль, отвечающий за поиск термина в Wikipedia, выполняет запрос на соответствующем сайте через API и озвучивает определение. Модуль, выполняющий перевод фразы, осуществляет перевод фразы на язык носителя и озвучивает её. Модуль, ответственный за выбор случайного числа, выполняет генерацию случайного числа из диапазона, заданного пользователем.

Взаимодействие между компонентами организовано по следующей схеме: пользователь через интерфейс программы или через голосовой ввод по ключевому слову инициирует запись команды, дальше ядро отдаёт звуковой файл обработчику речи, а модуль обработки речи возвращает ядру команду в текстовом виде и оно передаёт команду в обработчик команд, где выполняется определённая функция, в конечном итоге результат этой функции выводится в текстовом поле программы и озвучивается в обработчике речи.

Таким образом, выбранная архитектура и используемые инструменты разработки позволяют связать между собой все компоненты программы, для реализации поставленной задачи.

## 2.2 Выбор модели распознавания

Для распознавания речи потребуется модель, которая будет обрабатывать голосовой ввод. Так как создание своей собственной модели заняло бы большую часть времени, было принято решение о выборе готовой модели распознавания.

В настоящее время существует ряд моделей для распознавания естественной речи ASR с необходимым функционалом для реализации голосового ассистента. Рассмотрим наиболее популярные из них.

Whisper [9] — это модель от OpenAI, которая распознаёт и переводит речь в текст. Она была выпущена в сентябре 2022 года.

К достоинствам модели можно отнести:

– высокая точность – одна из лучших моделей для транскрибации речи в текст;

– контекстное понимание – учитывает контекст речи и делает меньше ошибок в сложных фразах;

– многоязычность – поддерживает множество языков и позволяет определять его автоматически;

– готовая модель не требующая дообучения;

– проста в установке;

– работает без интернета;

– является бесплатной.

Недостатки модели:

– требует GPU для быстрой работы, так как на CPU работает медленнее;

– большой размер модели – от 1,5 Гб для маленькой версии до 10 Гб для большой;

– отсутствует потоковая обработка – может работать только с полными аудиофайлами.

Модель Whisper является хорошим инструментом для распознавания речи, который обладает высокой точностью распознавания, позволяющий минимизировать ошибки, и обширной языковой основой, способной определить язык носителя. Однако, он является ресурсозатратным по отношению к памяти и

GPU, что может создавать трудности для большинства пользователей.

Google Speech Recognition API [10] — это сервис от Google, который позволяет конвертировать речь в письменный текст. Для этого используются алгоритмы машинного обучения и акустические модели. Чтобы использовать API, достаточно отправить данные по определённому URL в формате JSON, которые должны содержать информацию об аудио, которое требуется распознать.

Для работы с Google Speech Recognition требуется использовать библиотеку SpeechRecognition, которая предоставляет возможность работы с различными интерфейсами распознавания речи.

Достоинства сервиса Google Speech Recognition API:

– простота использования – работает через API и легко интегрируется;

– высокая точность распознавания;

– поддержка длинных аудиофайлов;

– возможность создавать настраиваемые модели.

Недостатки данного сервиса:

– требует подключения к интернету;

– ограничения бесплатной версии – плата за большие объёмы обработки аудиофайлов;

– скорость работы зависит от скорости интернета.

Таким образом, Google Speech Recognition API является универсальным сервисом для распознавания аудиоданных, который прост в использовании и обладает достаточно высокой точностью распознавания. Однако, он имеет зависимость от интернета и ограничения в бесплатной версии, если передавать большой объём данных.

Vosk [11] — это библиотека для распознавания речи с открытым исходным кодом от компании Alpha Cephei Inc. Она используется в чат-ботах, виртуальных помощниках и умной бытовой технике. С помощью модели можно создавать субтитры для фильмов, транскрипцию для лекций и интервью.

Достоинства модели Vosk:

– работает без интернета;

– поддержка большого количества языков;

– наличие легковесной модели – маленькая модель от 50 Мб, большая модель от 3,5 Гб;

– потоковая обработка – обрабатывает естественную речь в реальном времени.

Недостатки данного сервиса:

– точность ниже, чем у модели Whisper;

– плохо распознаёт в шумных условиях;

– сложность настройки модели для использования.

Vosk является хорошей моделью для распознавания естественной речи, которая отлично подойдёт для написания легковесных программ и для использования на устройствах с ограниченной памятью или смартфонах. Однако, она требует определённого опыта работы с такой моделью и имеет точность ниже, чем у своих аналогов.

DeepSpeech [12] — это система распознавания речи с открытым исходным кодом, разработанная компанией Mozilla. Компания пыталась сделать технологию распознавания речи более доступной и точной для широкого спектра приложений. Однако, в 2021 году компания сообщила, что сворачивает разработку DeepSpeech и переходит к роли консультанта в этой области.

Достоинства модели DeepSpeech:

– открытый исходный код – модель можно дообучать под свои задачи;

– не требует использование интернета.

Недостатки данного сервиса:

– устаревшая модель – прекращена поддержка данной модели;

– низкая точность распознавания естественной речи;

– требует дообучения для поддержки новых языков.

На основании вышеизложенного, DeepSpeech является простым, но устаревшим инструментом, который можно использовать в силу своей доступности и возможности работы без интернета, однако он требует некоторых доработок.

Wav2Vec 2.0 [13] — это инструмент, разработанный исследователями из Facebook AI, который позволяет распознавать речь и представлять его в виде

текста.

Достоинства модели Wav2Vec 2.0:

– качество распознавания естественной речи – в некоторых случаях превосходит модель Whisper;

– есть возможность дообучения модели – можно адаптировать под разнообразные задачи.

Недостатки данного инструмента:

– сложность использования данной модели – требует знаний фреймворка PyTorch;

– ресурсоёмкость – требует GPU для эффективной работы с моделью.

Wav2Vec 2.0 является хорошей моделью для распознавания речи, которая имеет высокое качество распознавания в сравнении с аналогами, однако требует определённых навыков для работы с этим инструментом.

PocketSphinx [14] — это модель для распознавания естественной речи с открытым исходным кодом, разработанная Университетом Карнеги Меллона.

Достоинства модели PocketSphinx:

– легковесная модель – её можно запустить даже на одноплатном компьютере размером с банковскую карту;

– не требует использование интернета.

Недостатки данной модели:

– очень низкая точность распознавания речи;

– устаревшая технология, которая используется в данной модели;

– распознаёт только простые команды – не подходит для свободной речи.

Таким образом, PocketSphinx имеет преимущество среди легковесных моделей, но отстаёт от своих аналогов по точности распознавания.

В таблице 4 представлена сводная информация о достоинствах и недостатках моделей распознавания естественной речи.

Таблица 4 – Сводная таблица достоинств и недостатков моделей распознавания

| Модель распознавания | Достоинство | Недостаток |
|---|---|---|
| Whisper | Точность распознавания. Бесплатная модель. Многоязычность. Работает без интернета. | Ресурсоёмкость. Отсутствует потоковая обработка. |
| Google Speech Recognition API | Простота использования, точность распознавания. | Работает только с интернетом, ограничения бесплатной версии, скорость зависит от интернета. |
| Vosk | Работает без интернета, легковесная модель, присутствует потоковая обработка. | Сложность настройки модели, точность ниже, чем у модели Whisper. |
| DeepSpeech | Работает без интернета, отрытый исходный код. | Устаревшая модель, низкая точность распознавания. |
| Wav2Vec 2.0 | Высокая точность распознавания. | Сложность использования модели, ресурсоёмкость. |
| PocketSphinx | Легковесная модель, не требует интернета. | Низкая точность распознавания, распознаёт только простые команды. |

Из преимуществ над аналогами Google Speech Recognition имеет возможность распознавать одновременно несколько языков. С помощью API обращается к модели, хранящейся на сервере Google. А модель Vosk имеет высокую скорость распознавания, в отличие от Google Speech Recognition API, работает без подключения к сети интернет и может обрабатывать речь в потоковом режиме. Среди аналогов эти две модели отлично подходят для текущей задачи.

Таким образом, при выборе модели для распознавания отданы предпочтения моделям Google Speech Recognition API и Vosk. Первая модель используется для распознавания команды, так как она лучше справляется со сложными запросами, а вторая модель используется для распознавания ключевого слова, так как она имеет высокую скорость обработки и поддерживает потоковую работу.

## 2.3 Разработка пользовательского интерфейса

Пользовательский интерфейс реализован с помощью библиотеки PyQt5. Она является достаточно популярной и не сложной в освоении. Библиотека PyQt5 была

выбрана для реализации GUI, потому что она имеет множество достоинств:

– поддержка кроссплатформенности;

– множество примеров и документаций;

– множество различных элементов GUI;

– гибкая настройка элементов.

За реализацию интерфейса программы отвечает файл quantum.py, а также он является основным файлом, который запускает голосового ассистента. Ниже представлен код точки входа в программу.

```
if __name__ == "__main__":
    app = TrayApp()
    app.run()
```

Основной класс TryApp инициализирует интерфейс приложения, создаёт второй поток для работы голосового ассистента и добавляет программу в системный трей. Ниже представлена функция реализации голосового ассистента и выделения ему отдельного потока.

```
def start_assistant(self):
     self.assistant = VoiceAssistant()
     self.assistant_thread = AssistantThread(self.assistant)
     self.assistant_thread.start()
```

Помимо запуска голосового помощника, происходит инициализация главного интерфейса и интерфейса настоек ассистента. Класс MainWindow отвечает за реализацию главного диалогового окна, которое содержит журнал работы системы и кнопки настройки и обновления. Далее представлена часть кода, отвечающая за создание текстового поля и его настройку.

```
self.log_output = QTextEdit()
        self.log_output.setReadOnly(True)
        self.log_output.setStyleSheet("""
                    QTextEdit {
                        background-color: #f8f8f8;
                        border: 1px solid #ddd;
                        padding: 5px;
                        font-family: 'Calibri';
                        font-size: 15px;
                    }
```

```
        """)
        group_layout.addWidget(self.log_output)
```

В представленном выше коде создаётся текстовое поле, после оно устанавливается только для чтения, применяет текстовый стиль и добавляет его на слой для последующего отображения.

Класс SettingsWindow отвечает за реализацию окна настроек ассистента. Это окно содержит три кнопки:

– кнопка «По умолчанию», которая заполняет поля данными по умолчанию;

– кнопка «Отмена», отвечающая за сброс введённых данных и закрытие окна настроек;

– кнопка «Сохранить», отвечающая за сохранение введённых данных в конфигурационный файл settings.json.

Также это окно содержит четыре поля настроек:

– скорость речи озвучивания текста ассистентом;

– громкость речи озвучивания;

– язык озвучивания;

– город по умолчанию, задаваемый для команды прогноза погоды.

## 2.4 Разработка ядра программы

Основой приложения является непрерывный цикл, который отслеживает ключевое слово в реальном времени и который напрямую взаимодействует с другими важными частями программы. Составляющее ядра программы:

– модуль отслеживания ключевого слова assistant.py;

– модуль обработки и синтеза речи speech.py;

– модуль обработки команды commands.py.

Модули представленные выше отвечают за основной функционал программы.

После того, как файл с интерфейсом программы запущен, создаётся объект класса assistant, который в свою очередь создаёт два объекта других классов SpeechEngine и CommandHandler, и далее программа начинает ожидать от пользователя ключевое слово.

Ниже представлен код класса VoiceAssistant.

```
class VoiceAssistant:
    def __init__(self):
        self.speech = SpeechEngine()
        self.commands = CommandHandler()
    def run(self):
        while True:
            text = self.speech.listen()
            if text:
                print(text)
                response = self.commands.handle(text)
                self.speech.say(response)
```

Сначала в цикле записывается и преобразуется в текст отрезок речи пользователя, записанной с микрофона, с помощью модели распознавания Vosk, за эти действия отвечает объект класса SpeechEngine. После того, как функция прослушки возвращает записанный текст, в нём проверяется наличие ключевого слова. Если его нет в этом тексте, то программа продолжает слушать, а если есть, то воспроизводится звук активации и начинается запись аудиофайла с командой. После этого аудиофайл отправляется на сервер Google через API, затем сервер отправляет распознанный текст. Далее текст передаётся в обработчик команд и ожидается результат исполнения команды. Далее результат команды преобразуется из текста в речь с помощью функции озвучивания say класса SpeechEngine и цикл вновь ожидает ключевого слова.

Далее показан код метода listen класса SpeechEngine, который отвечает за запись и обработку речи пользователя.

```
def listen(self):
   data = self.stream.read(4096, exception_on_overflow=False)
                    if self.recognizer_vosk.AcceptWaveform(data):
                        result = json.loads(self.recognizer_vosk.Result())
                        text = result.get("text", "").lower()
                        if self.keyword in text:
                             with sr.Microphone() as source:
                                 audio = self.recognizer.listen(source)
                                 command =
self.recognizer.recognize_google(audio, language="ru-RU")
```

```
return command
```

Функция listen записывает аудиофайл с речью пользователя и затем при помощи модели распознавания Vosk преобразует речь в текстовый формат и проверяет текст на наличие ключевого слова. Если его не обнаружилось повторяется цикл записи и преобразования. Когда обнаружится ключевое слово, начнётся запись команды пользователя и её преобразование в текст с помощью модели распознавания Google. Внутри функции с микрофона записывается речь в параметр audio, а затем в переменную text записывается ответ от сервера с распознанным текстом.

Помимо этого метода, в классе есть метод say, который отвечает за озвучивание ответа ассистента. Ниже представлена часть этого метода.

```
engine = pyttsx3.init() #offline
            engine.setProperty('rate', rate)
            engine.setProperty('volume', volume)
            voices = engine.getProperty('voices')
            if language == 'ru':
                for voice in voices:
                    if 'russian' in voice.name.lower():
                        engine.setProperty('voice', voice.id)
                        break
            else:
                engine.setProperty('voice', voices[0].id)
            engine.say(text)
            engine.runAndWait()
```

Сначала задаются параметры озвучивания: частота и громкость озвучивания. Далее идёт разветвление, зависящее от параметра языка озвучки: английского или русского, тем самым задаётся параметр языка озвучивания. И после этого происходит озвучка ответа ассистента, посредством использования методов библиотеки озвучивания.

Из класса CommandHandler вызывается функция handle, которая отвечает за определение и выполнение определённой команды. Ниже показана часть кода, которая отвечает за проверку речи пользователя на команду генерации случайного числа. Остальные условия проверки ключевого слова для вызова команды

реализованы аналогичным образом.

```
elif any(word in text.lower() for word in ["случайное", "рандомное", "сгенерир",
"выбери", "число"]):
            from instructions.gen_num import generation_number
            return generation_number(text)
```

В этом фрагменте кода проверяется условие соответствия команды, то есть если в речи, преобразованной в текст, есть какое-либо из перечисленных ключевых слов списка, то происходит импорт файла, в котором будет происходить реализация команды, в данном случае генерация случайного числа в заданном диапазоне. После импорта следует выполнение этой функции с передачей ей параметра text.

Каждое условие отвечает за вызов определённой функции, отвечающей за исполнение команды голосового ассистента.

Далее на рисунке 3 представлена UML-диаграмма деятельности, показывающая последовательность действий работы ассистента.

Рисунок 3 – UML-диаграмма деятельности

## 2.5 Разработка команд голосового ассистента

Функция generation_number отвечает за генерацию случайного числа. Ниже представлена часть кода этой функции.

```
matches = re.findall(r'(?:от|между)\s*(\d+)\s*(?:до|и)\s*(\d+)',
text.lower())
        if not matches:
            return "Не удалось распознать диапазон."
        from_num, to_num = map(int, matches[0])
```

```
if from_num >= to_num:
            return "Первое число должно быть меньше второго"
        result = random.randint(from_num, to_num)
        return f"Случайное число: {result}"
```

Сначала в тексте команды происходит поиск заданного диапазона чисел через регулярные выражения, если диапазон не найден, то сообщается пользователю, что не удалось распознать диапазон. Далее разбивается интервал на два числа: от и до. Сравниваются эти числа, и если первое число больше второго, то сообщается пользователю, что интервал не верен. После проверки условий генерируется случайное число в заданном интервале и возвращается результат пользователю.

Функция open_app отвечает за открытие приложения. Ниже представлены фрагменты кода этой функции.

```
app_path = cls.find_in_program_files(app_name) #1
if app_path:
        os.startfile(app_path)
        return f"Нашел и открываю {app_name}"
reg_path = cls.find_in_registry(app_name) #2
if reg_path:
        subprocess.Popen(reg_path, shell=True)
        return f"Открываю {app_name}"
    subprocess.Popen(app_name, shell=True) #3
    return f"Запускаю {app_name}"
```

В функцию открытия программы передаётся параметр app_name, в котором хранится название искомой программы. Поиск приложения осуществляется тремя способами. Если первый способ не сработал, то ищем вторым способом, если и этот способ не получил удовлетворительный результат, то используется третий. В случае, если не будет найдено искомое приложение, то пользователь получит озвученное сообщение о том, что приложение не было найдено. Первый поиск осуществляется в папке ProgramFiles и смежной с ней папками, где зачастую находятся установленные пользователем приложения в Windows. Второй способ поиска происходит по реестру, где находятся стандартные приложения Windows. Третий поиск осуществляется через shell, где могут найтись приложения,

которые находятся в path. После того как какой-либо способ найдёт искомое приложение на компьютере, оно откроется.

Функция web_search отвечает за поиск запроса в браузере. Ниже показан фрагмент кода этой функции, который отвечает за поиск запроса в браузере.

```
def search_in_browser(query):
    query = clean_query(query)
    encoded_query = urllib.parse.quote_plus(query)
    search_url = f"https://www.google.com/search?q={encoded_query}"
    webbrowser.open(search_url)
    return "Поиск осуществлён"
```

Сначала в этой функции из запроса пользователя убираются лишние слова, которые определили команду, с помощью функции clean_query. Далее этот текст преобразовывается в запрос для поисковой строки. И затем открывается браузер с данным запросом.

Функция wiki_search отвечает за поиск термина на Wikipedia. Она выдаёт краткое описание термина, полученное через Wikipedia API. Сначала из запроса пользователя убираются лишние слова с помощью регулярного выражения. Далее через API запрашивается страница, содержащая искомый термин. Ниже представлена строка запроса и получение краткого определения.

```
page = self.wiki.page(term)
summary = page.summary.split('. ')
        short_summary = '. '.join(summary[:SUMMARY_LENGTH]) + '.' if
len(summary) > 1 else page.summary
        return f"{page.title}: {short_summary}"
```

Функция translate отвечает за перевод фразы. Она реализуется с помощью translate google API. Сначала производится настройка переводчика, затем происходит определение исходного языка. После осуществляется перевод фразы на заданный язык. Если пользователь отправляет команду перевести фразу на английский, то в параметр языка, на который нужно перевести, заносится соответствующее значение. Аналогично реализовано, если пользователь просит перевести на русский. Ниже представлен фрагмент кода функции translate.

```
translator = Translator(service_urls=['translate.google.com'])
def translate(text, target_language='en'):
    try:
```

```
translation = translator.translate(text, dest=target_language)
        return translation.text, target_language
    except requests.exceptions.ConnectionError:
        return "Нет подключения к интернету для перевода"
    except Exception as e:
        return "Не удалось выполнить перевод"
```

Функция get_weather отвечает за вывод информации о погоде. Она реализуется с помощью API взаимодействующего с сервисом OpenWeatherMap, который выдаёт информацию о погоде. Для реализации требуется API-ключ, который отправляется в этот сервис для получения погоды. Сначала в запросе проверяется наличие названия города, если его нет, то погода выдаётся по городу, установленному по умолчанию, а если есть, то записываем название города в переменную и приводим в его в номинатив, если это требуется. Далее осуществляется запрос с передачей названия города и API-ключа на сервер. Затем полученная информация распаковывается и озвучивается пользователю. Ниже представлен фрагмент функции получения информации о погоде.

```
api_key = WEATHER_API_KEY
        response =
requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid
={api_key}")
        data = response.json()
        return f"Сейчас в городе {city} {round(data['main']['temp'] - 273)}
градусов."
```

# 3 Тестирование и анализ результатов

В данном разделе проводится проверка работоспособности разработанного голосового ассистента. Основное внимание уделяется корректности работы функционала программы и взаимодействию с пользовательским интерфейсом.

Голосовой ассистент не имеет основного интерфейса программы. Он работает скрыто и автономно. Однако, программа имеет свой ярлык на панели задач в скрытых значках (рисунок 4), который при двойном нажатии открывает небольшой интерфейс приложения или при нажатии правой кнопки мыши, выдаст меню, с помощью которого можно открыть интерфейс приложения, закрыть голосовой ассистент полностью или вызвать меню настроек (рисунок 4).

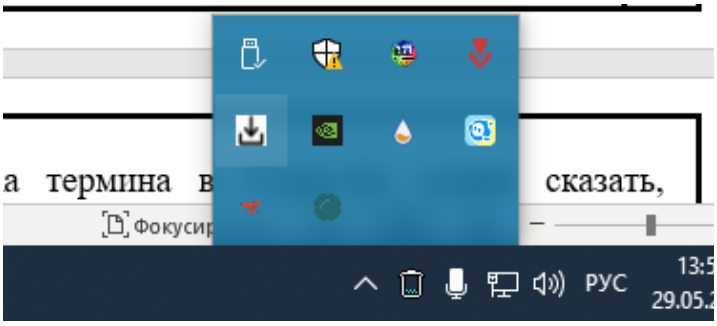



Рисунок 4 – Ярлык программы

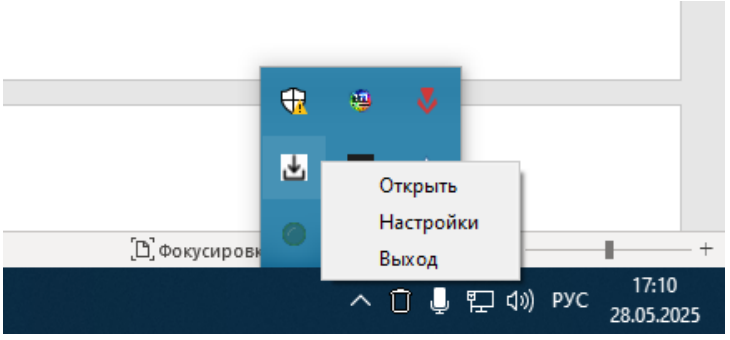



Рисунок 5 – Меню ярлыка программы

После двойного нажатия на ярлык приложения или кнопку «Открыть» открывается главное диалоговое окно приложения, где отображается информация о истории запросов и две кнопки (рисунок 6): кнопка «Обновить», которая отвечает за обновление журнала логов и кнопка «Настройка», которая отвечает за открытие другого диалогового окна с настройками ассистента (рисунок 7).

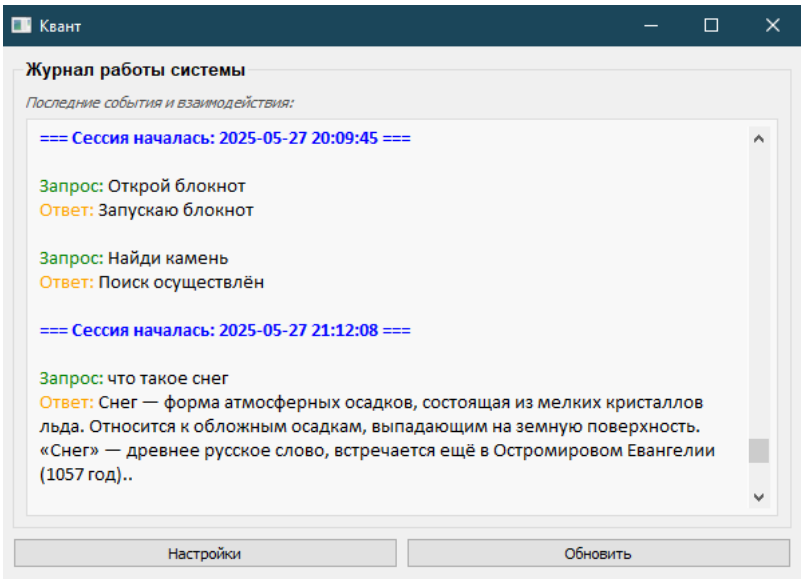



Рисунок 6 – Главное окно приложения

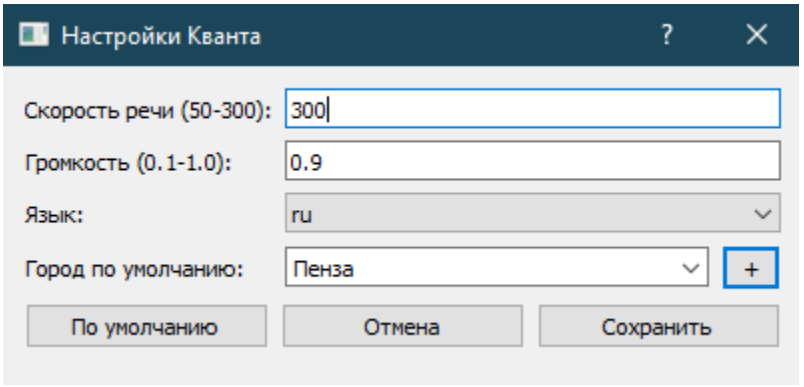



Рисунок 7 – Окно настроек ассистента

Голосовой ввод является главным элементом взаимодействия пользователя с ассистентом. Его тестирование проводилось на всех этапах разработки. Проверялись:

– корректная работоспособность записи и обработки речи пользователя;

– распознавание речи;

– верное определение команды;

– вывод и озвучивание ответа ассистента пользователю.

В рамках тестирования были проверены следующие варианты использования программы.

Отладка для проверки распознавания ключевого слова и команды пользователя проводились в среде разработки PyCharm.

Пользователь при помощи голосового ввода активирует ассистента. Когда голосовой помощник определяет, что пользователь сказал ключевое слово, записывает речь в аудиофайл. Далее речь преобразуется в текст (рисунок 8), а он в свою очередь передается в обработчик команд, где определяется сказанная пользователем команда. Далее она выполняется и результат озвучивается пользователю. После этого ассистент снова начинает отслеживать ключевое слово.

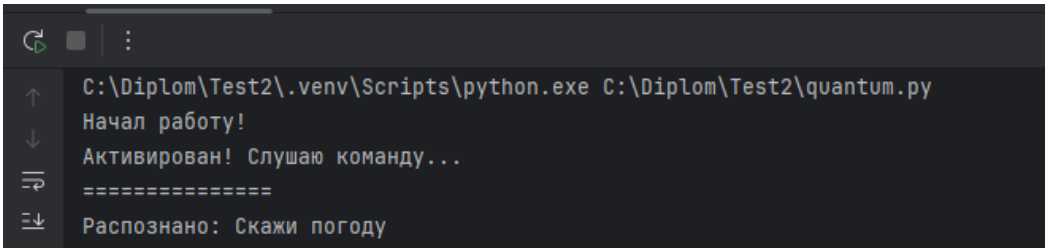



Рисунок 8 – Преобразование речи в текст

Проверка функции вывода информации о погоде заключается в вводе команды с использованием слова «погода». Чтобы проверить работоспособность, нужно сказать одну из доступных команд, например: «квант скажи погоду». Ассистент выведет и озвучит информацию о погоде в городе по умолчанию (рисунок 9). Если уточнить в каком городе нужно узнать погоду, то помощник выведет погоду того города, который был назван пользователем после предлога «в» (рисунок 10).

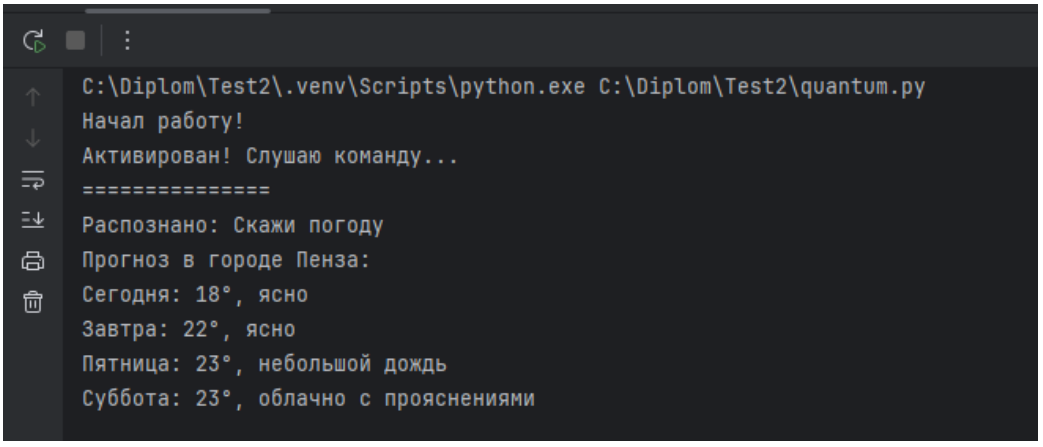



Рисунок 9 – Вывод погоды по умолчанию

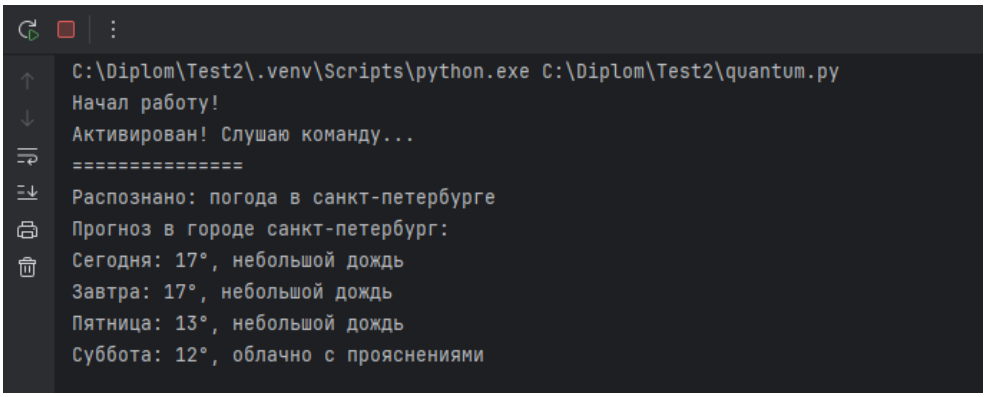



Рисунок 10 – Вывод погоды с уточнением

Для проверки функции открытия внешнего приложения нужно сказать, например: «квант открой блокнот». Ассистент произведёт поиск приложения на компьютере и при успешном обнаружении откроет его (рисунок 11).

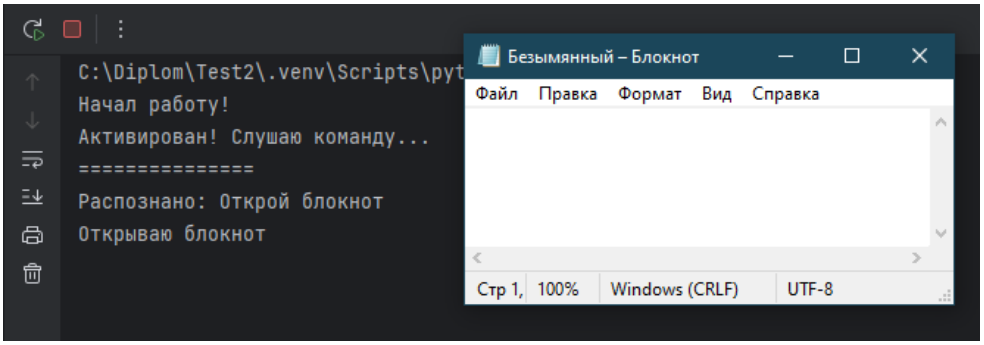



Рисунок 11 – Открытие приложения

Для проверки функции поиска запроса в браузере нужно произнести, например: «квант найди рецепт блинов». Ассистент осуществит поиск запроса и озвучит три наиболее подходящих результата (рисунок 12).

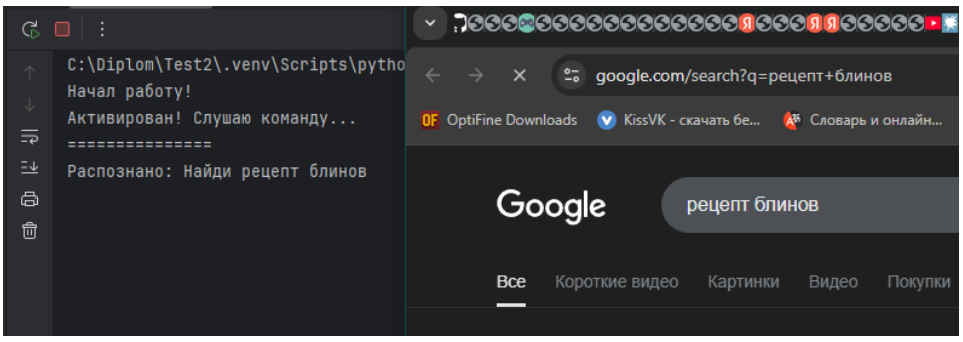



Рисунок 12 – Поиск запроса

Для проверки функции поиска термина в Wikipedia можно сказать, например: «квант что такое снег». Голосовой помощник произведёт поиск термина в Wikipedia и озвучит результат (рисунок 13).

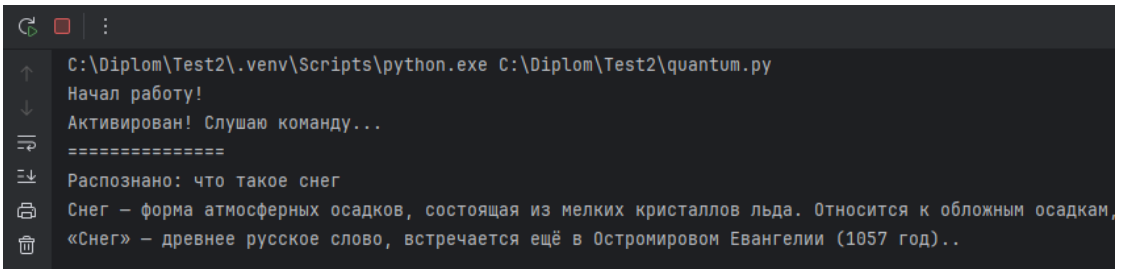



Рисунок 13 – Поиск термина

Для проверки функции переводчика можно произнести такую фразу, например: «квант переведи на английский снеговик». Ассистент осуществит перевод фразы на английский язык и озвучит результат (рисунок 14). Также можно переводить с английского на русский (рисунок 15).

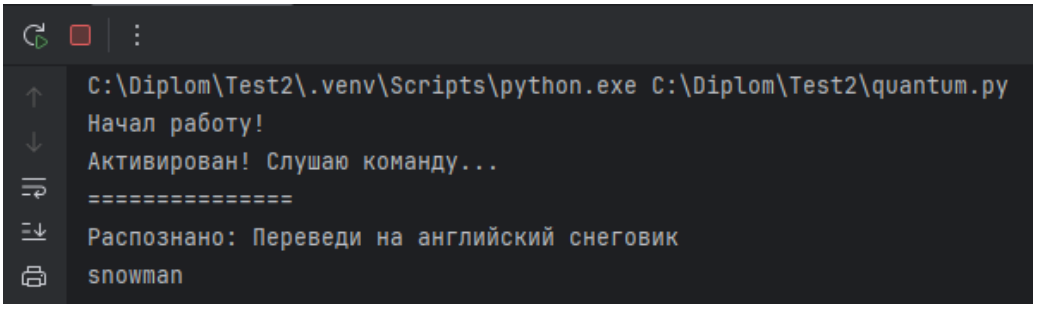



Рисунок 14 – Перевод на английский

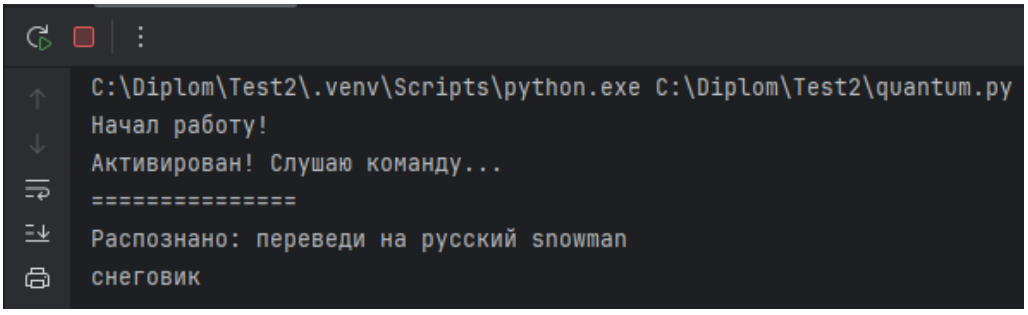



Рисунок 15 – Перевод на русский

Для проверки функции генерации случайного числа в заданном диапазоне можно сказать, например: «квант выбери число от 10 до 20».

Голосовой помощник сгенерирует случайное число в этом интервале и озвучит результат (рисунок 16).

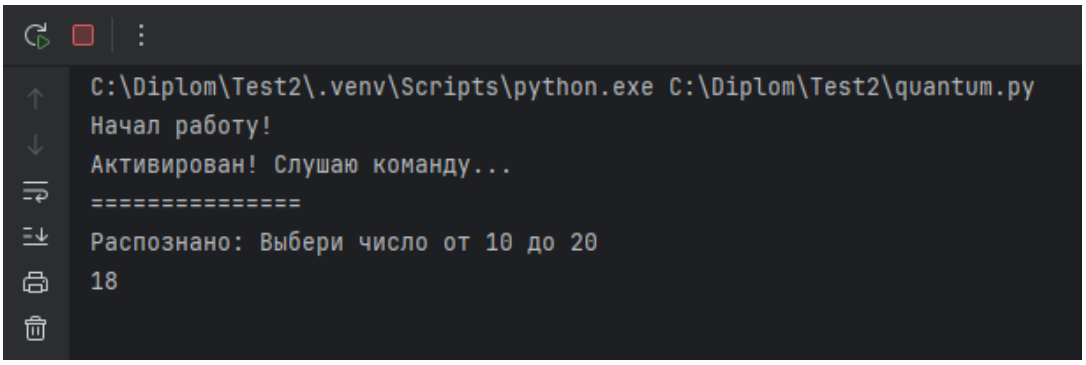



Рисунок 16 – Генерация случайного числа

Таким образом, на основании тестирования ассистента все его функции работают корректно. В результате тестирования голосового помощника средняя оценка точности распознавания команд составила 70 процентов. Ошибки распознавания в основном связаны с нечётким произношением слов или с использованием сложных фонетических конструкций в речи пользователя. Проведённое тестирование подтвердила работоспособность голосового ассистента и его функций. Результаты соответствует заявленным ожиданиям.

# 4 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда пользователей персональных электронно-вычислительных машин, участвующих в разработке и эксплуатации программного обеспечения, а также способы минимизации вредных факторов на здоровье.

## 4.1 Характеристика потенциальных рисков при работе с персональным компьютером

В соответствии с ГОСТ 12.0.003-2015 «Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности», при эксплуатации персонального компьютера выделяются следующие основные риски:

– риски поражения электрическим током из-за: неисправной проводки; отсутствия заземления; попадания жидкости на технику; короткого замыкания при перегрузке сети;

– риски заболеваний опорно-двигательной системы из-за: неправильного положения тела; неправильной высоты стула и стола;

– риски заболеваний зрения из-за: использования персонального компьютера без перерывов для глаз; неправильной яркости экрана и расположения монитора.

## 4.2 Оборудование рабочего места

Основные требования, которые должны соблюдаться при пользовании рабочем местом:

– расстояние от монитора до глаз должно быть не менее 500 – 700 мм. Монитор должен располагаться ниже уровня глаз под углом 15-20°;

– высота стола должна быть 680 – 800 мм (в зависимости от роста пользователя);

– кресло должно иметь регулируемую высоту сиденья и спинку, а также глубину сиденья 400 - 450 мм.

На рисунке 17 показана правильная организация рабочего пространства для персонального компьютера.

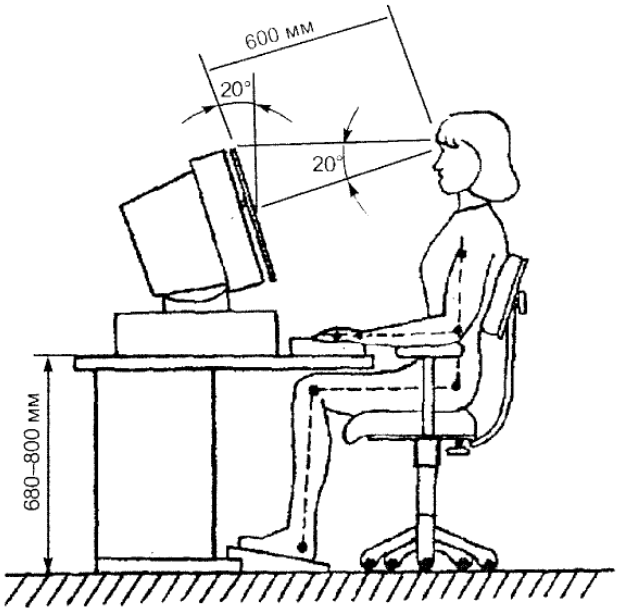



Рисунок 17 – Правильная организация рабочего места

Рабочее место с монитором должно гарантировать пользователю удобную работу с персональным компьютером в положении сидя. При нахождении за персональным компьютером должны выполняться следующие условия: прямая спина с опорой на спинку кресла; расслабленные плечи; кисти и запястья в прямом положении; ноги должны устойчиво стоять на полу.

### 4.3 Освещённость рабочего места

Для обеспечения здоровья органов зрения на рабочем месте должны соблюдаться стандарты освещения. Согласно санитарным правилам, общая освещённость рабочего места должна составлять не менее 400 люкс. Источники освещения должны обеспечивать равномерный рассеянный свет без бликов на мониторе. Цветовая температура источников освещения должна составлять 4000 – 5000К, что соответствует белому цвету. Недопустимо применение мерцающих источников света и расположение в поле зрения пользователя ярких источников света.

## 4.4 Электробезопасность

Электробезопасность при работе с персональным компьютером регламентируется ГОСТ 12.1.019-2017 «Электробезопасность». Исходя из стандарта, всё электрооборудование должно иметь защитное заземление. ГОСТ предусматривает использование только исправного оборудования с целой изоляцией и компонентами.

Кабели должны быть проложены в местах, где они не будут подвергаться механическим воздействиям и повреждениям. Стандарт требует, чтобы все электрические компоненты были защищены от случайного прикосновения, а корпуса электрооборудования были заземлены.

Пользователи оборудования должны быть обучены правилам электробезопасности и оказании первой помощи при поражении электрическим током.

## 4.5 Пожарная безопасность

Основные риски пожарной безопасности связаны с возможным перегревом оборудования, короткими замыканиями и возгоранием компонентов. Для предотвращения опасных ситуаций необходимо поддерживать исправное состояние электрооборудования с средствами защиты. Особое внимание необходимо обратить на правильную организацию рабочего пространства - компьютеры должны располагаться на расстоянии не менее 0.5 метра от отопительных приборов и горючих материалов. В помещениях с компьютерами необходимо наличие средств пожаротушения.

Для повышения противопожарной безопасности необходимо соблюдать следующие правила: проходы и запасные выходы не должны быть перекрыты посторонними предметами; в здании должна быть установлена автоматическая пожарная сигнализация и планы эвакуации.

## 4.6 Профилактические меры для снижения утомления глаз

В качестве профилактических мер для снижения утомления глаз, улучшения кровоснабжения глаз, релаксации глазодвигательных мышц рекомендуется проводить упражнение для глаз во время регламентированных перерывов вместе с другими комплексами физических упражнений каждые 20-30 минут работы за персональной электронно-вычислительной машиной.

При появлении зрительного дискомфорта эти упражнения следует проводить индивидуально в соответствии со своими запросами по периодичности. При возникновении значительного зрительного дискомфорта в процессе работы, несмотря на хорошее качество монитора, освещения и соблюдение эргономических и режимных требований, следует ограничить время работы с дисплеем. Для этого требуется либо увеличить время перерыва для отдыха, либо сменить деятельность на другую, не связанную с работой с электронно-вычислительной машиной.

## Заключение

В ходе выполнения работы был разработан голосовой ассистент для персонального компьютера, который может записывать и распознавать речь, обрабатывать и выполнять различные команды пользователя. Голосовой помощник может выполнять следующие функции: поиск запроса в браузере, поиск термина в Wikipedia, перевод фразы, генерация случайного числа в заданном диапазоне, вывод информации о погоде, открытие приложения.

Для реализации поставленной задачи были изучены методы работы с моделями и библиотеками распознавания и синтеза речи, также со стандартными и другими библиотеками языка python, и помимо этого были получены навыки организации структуры проекта.

Тестирование показало работоспособность голосового ассистента и всех его функций. Точность распознавания команд в среднем составляет семьдесят процентов.

В дальнейшем программу можно усовершенствовать, путём модифицирования уже существующих функций и добавления новых, например: составления расписания, создания заметок, взаимодействия с музыкальным плеером и других.

Таким образом, реализован прототип голосового ассистента, который удовлетворяет требованиям задачи, а значит поставленная цель достигнута. Такой голосовой помощник может быть внедрён в повседневную жизнь, чтобы упростить различные задачи и повысить эффективность работы.

## Список использованных источников


1. Ananta Devkota, Shashank Gupta, Raju Shrestha, Frode Eika Sandnes. Students' Perceptions of Study Efficacy, Effectiveness, and Efficiency: Effects of Voice Assistant Use. Innovative Technologies and Learning: 7th International Conference, ICITL 2024, Tartu, Estonia, August 14–16, 2024, Proceedings, Part II. с. 145-153.
2. Яндекс Алиса. [Электронный ресурс] // alice.yandex.ru – Режим доступа – URL. https://alice.yandex.ru/about. (дата обращения 18.04.2025).
3. Siri. [Электронный ресурс] // support.apple.com – Режим доступа – URL. https://support.apple.com/ru-ru. (дата обращения 18.04.2025).
4. Google Assistant. [Электронный ресурс] // assistant.google.com – Режим доступа – URL. https://assistant.google.com/. (дата обращения 18.04.2025).
5. Cortana. [Электронный ресурс] // copilot.microsoft.com – Режим доступа – URL. https://copilot.microsoft.com/. (дата обращения 24.04.2025).
6. C++. [Электронный ресурс] // learn.microsoft.com – Режим доступа – URL. https://learn.microsoft.com/ru-ru/cpp/overview/visual-cpp-in-visual-studio?view=msvc-170. (дата обращения 28.05.2025).
7. C#. [Электронный ресурс] // learn.microsoft.com – Режим доступа – URL. https://learn.microsoft.com/ru-ru/dotnet/csharp/. (дата обращения 28.05.2025).
8. Python. [Электронный ресурс] //www.python.org – Режим доступа – URL. https://www.python.org/. (дата обращения 28.05.2025).
9. Whisper. [Электронный ресурс] // openai.com – Режим доступа – URL. https://openai.com/index/whisper/. (дата обращения 24.04.2025).
10. Google Speech Recognition API. [Электронный ресурс] // cloud.google.com – Режим доступа – URL. https://cloud.google.com/speech-to-text/docs/reference/rest. (дата обращения 24.04.2025).
11. Vosk. [Электронный ресурс] // github.com – Режим доступа – URL. https://github.com/alphacep/vosk-api. (дата обращения 24.04.2025).
12. DeepSpeech. [Электронный ресурс] // github.com – Режим доступа – URL. https://github.com/mozilla/DeepSpeech. (дата обращения 24.04.2025).

13. Wav2Vec 2.0. [Электронный ресурс] // github.com – Режим доступа – URL. https://github.com/YmfiuByf/wav2vec-2.0. (дата обращения 24.04.2025).

14. PocketSphinx. [Электронный ресурс] // cmusphinx.github.io – Режим доступа – URL. https://cmusphinx.github.io/. (дата обращения 24.04.2025).

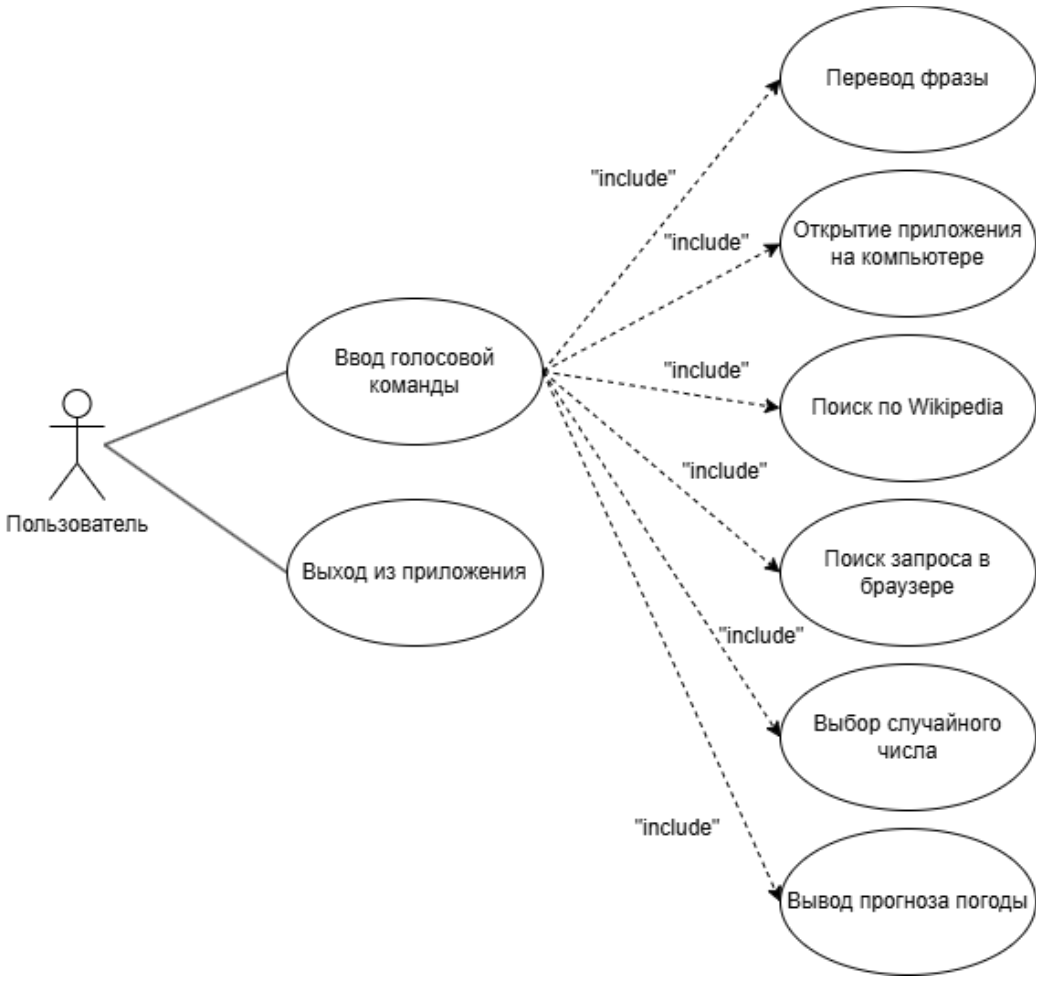
# Приложение А

(обязательное)

Голосовой ассистент для персонального компьютера

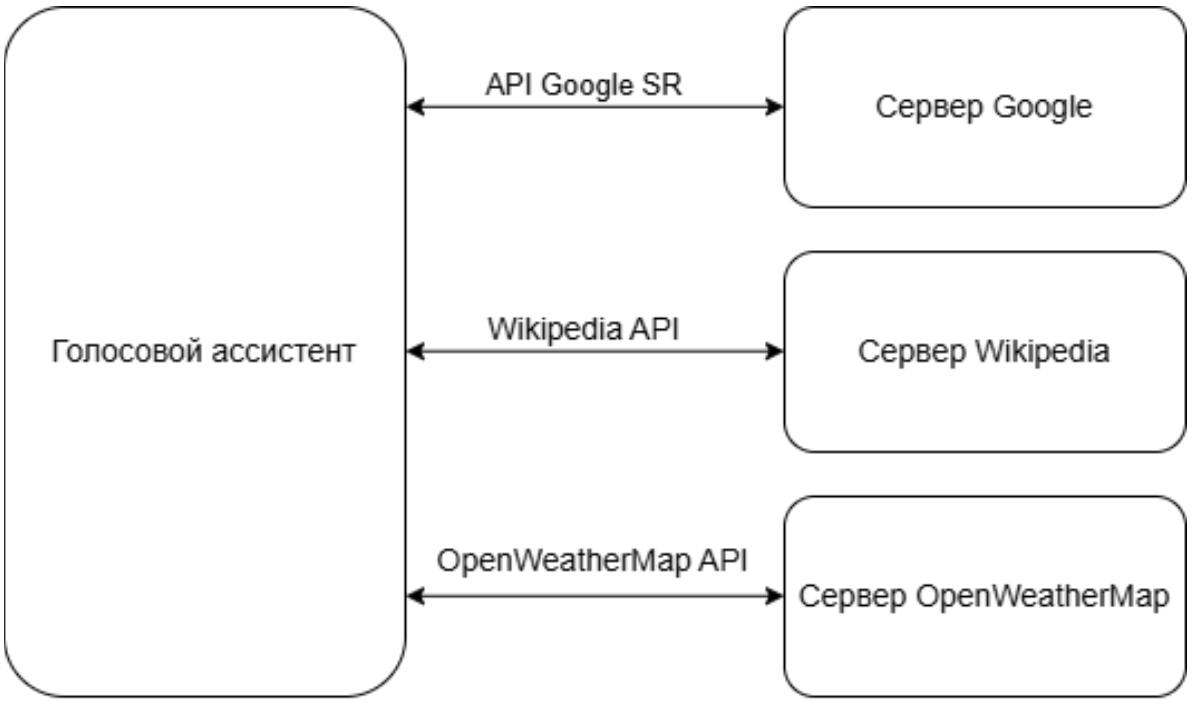
UML – диаграммы

## Приложение А.1 - Диаграмма вариантов использования



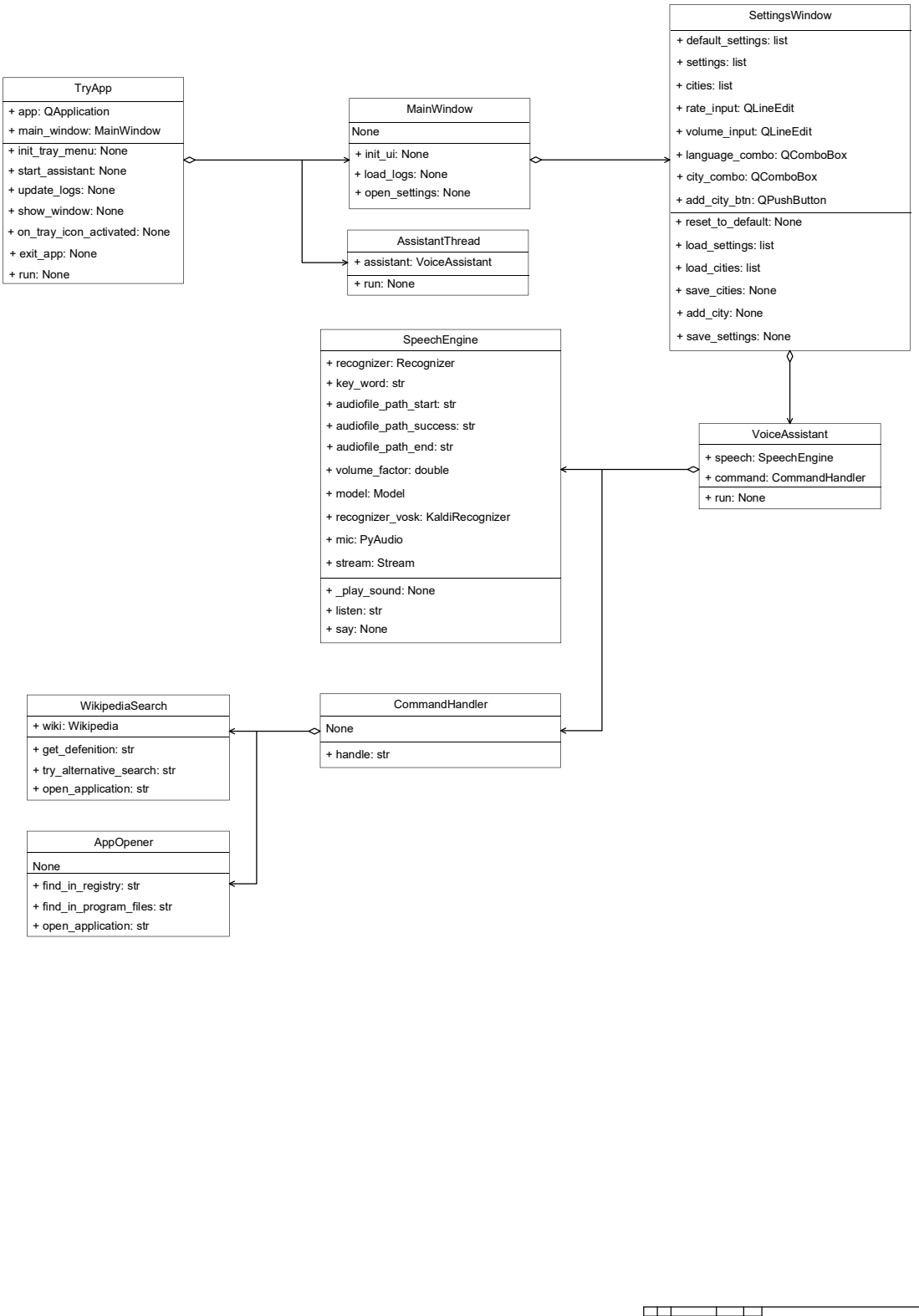

| | | | | | | |
|---|---|---|---|---|---|---|
| Изм. | Лист | № докум. | Подпись | Дата | ПГУ 1.090301.12.001 ПЗ | Лист 47 |

## Приложение А.2 - Диаграмма развёртывания




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | ПГУ 1.090301.12 002.Д19 | |
| | | | | | | | | | Лист / Масса / Масштаб |
| Изм. | Лист | № докум. | Подпись | Дата | | | | Голосовой ассистент для персонального компьютера<br>UML – Диаграмма размертывания | |
| Разраб. | | Кирьянов В.Е. | | | | | | | Лист / Листов |
| Пров. | | Юрова О.В. | | | | | | | |
| Т. контр. | | | | | | | | | |
| Рецепз. | | | | | | | | | |
| Н. контр. | | Быков А.С. | | | | | | | ФВТ, гр. 21ВВВ1 |
| Утв. | | Митрохин М.А. | | | | | | | |

# Приложение А.3 - Диаграмма классов




ПГУ 1.090301.12 002 Д11

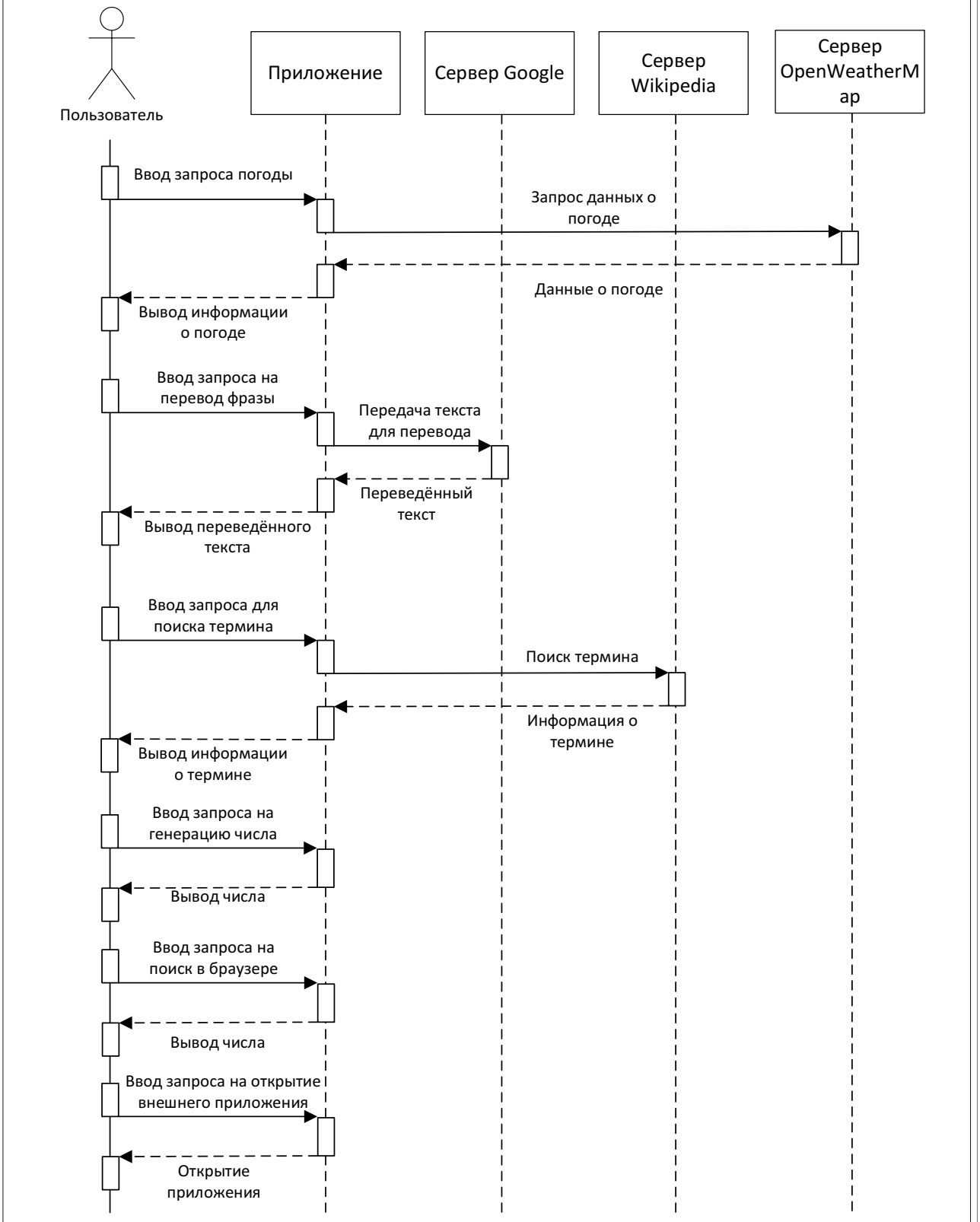
Голосовой ассистент для персонального компьютера
UML – Диаграмма размещения

ФВТ, гр. 21ВВВ1

## Приложение А.4 - Диаграмма деятельности

| | | | | | | |
|---|---|---|---|---|---|---|
| Изм. | Лист | № докум. | Подпись | Дата | ПГУ 1.090301.12 002.Д17 | |
| Разраб. | | Кирьянов В.Е. | | | Голосовой ассистент для персонального компьютера<br>UML – Диаграмма деятельности | Лист / Масса / Масштаб |
| Пров. | | Юрова О.В. | | | | |
| Т. контр. | | | | | | Лист / Листов |
| Реценз. | | | | | | |
| Н. контр. | | Быков А.С. | | | | ФВТ, гр. 21ВВ81 |
| Утв. | | Митрохин М.А. | | | | |

## Приложение А.3 - Диаграмма последовательности




ПГУ 1.090301.12 002 Д13

Голосовой ассистент для персонального компьютера
UML – диаграмма вариантов использования

ФВТ, гр. 21ВВВ1

# Приложение Б

(обязательное)

Голосовой ассистент для персонального компьютера

Листинг программы

Файл «quantum.py»

```
import sys
import json
from PyQt5.QtWidgets import (QApplication, QMainWindow, QLabel,
QVBoxLayout, QHBoxLayout,
                             QWidget, QSystemTrayIcon, QAction,
QMenu, QTextEdit,
                             QPushButton, QDialog, QLineEdit,
QFormLayout, QGroupBox, QComboBox)
from PyQt5.QtGui import QIcon, QFont, QIntValidator,
QDoubleValidator
from PyQt5.QtCore import QThread, pyqtSignal

from config.log_highlighter import LogHighlighter
from kernel.assistant import VoiceAssistant

class SettingsWindow(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Настройки Кванта")
        self.setFixedSize(400, 160)  # Увеличили размер окна
        self.default_settings = {"rate": 150, "volume": 0.9,
"language": "ru", "city_default": "Пенза"}
        self.settings = self.load_settings()
        self.cities = self.load_cities()

        layout = QFormLayout()

        # Поле для скорости (rate)
        self.rate_input = QLineEdit()
        self.rate_input.setValidator(QIntValidator(50, 300))
        self.rate_input.setText(str(self.settings.get("rate", 150)))
        layout.addRow("Скорость речи (50-300):", self.rate_input)

        # Поле для громкости (volume)
        self.volume_input = QLineEdit()
        self.volume_input.setValidator(QDoubleValidator(0.1, 1.0, 2))
        self.volume_input.setText(str(self.settings.get("volume",
0.9)))
        layout.addRow("Громкость (0.1-1.0):", self.volume_input)
        # Выпадающий список для языка
        self.language_combo = QComboBox()
        self.language_combo.addItems(["ru", "en"])

self.language_combo.setCurrentText(self.settings.get("language",
"ru"))
        layout.addRow("Язык:", self.language_combo)

        # Выпадающий список для города с возможностью добавления
        city_layout = QHBoxLayout()

        self.city_combo = QComboBox()
        self.city_combo.setEditable(True)
```

```
self.city_combo.addItems(self.cities)

self.city_combo.setCurrentText(self.settings.get("city_default",
"Пенза"))
        city_layout.addWidget(self.city_combo)

        self.add_city_btn = QPushButton("+")
        self.add_city_btn.setFixedWidth(30)
        self.add_city_btn.setToolTip("Добавить город в список")
        self.add_city_btn.clicked.connect(self.add_city)
        city_layout.addWidget(self.add_city_btn)

        layout.addRow("Город по умолчанию:", city_layout)

        # Кнопки
        buttons_layout = QHBoxLayout()

        # Кнопка "По умолчанию"
        default_btn = QPushButton("По умолчанию")
        default_btn.setIcon(QIcon.fromTheme("document-revert"))
        default_btn.clicked.connect(self.reset_to_default)
        buttons_layout.addWidget(default_btn)

        # Кнопка "Отмена"
        cancel_btn = QPushButton("Отмена")
        cancel_btn.setIcon(QIcon.fromTheme("dialog-cancel"))
        cancel_btn.clicked.connect(self.close)
        buttons_layout.addWidget(cancel_btn)

        # Кнопка "Сохранить"
        save_btn = QPushButton("Сохранить")
        save_btn.setIcon(QIcon.fromTheme("document-save"))
        save_btn.clicked.connect(self.save_settings)
        buttons_layout.addWidget(save_btn)
        layout.addRow(buttons_layout)
        self.setLayout(layout)

    def reset_to_default(self):
        self.rate_input.setText(str(self.default_settings["rate"]))

self.volume_input.setText(str(self.default_settings["volume"]))

self.language_combo.setCurrentText(self.default_settings["language"])

self.city_combo.setCurrentText(self.default_settings["city_default"])
def load_settings(self):
        try:
            with open('config/settings.json', 'r', encoding='utf-8')
as f:
                return json.load(f)
        except (FileNotFoundError, json.JSONDecodeError):
            return {"rate": 150, "volume": 0.9, "language": "ru",
"city_default": "Пенза"}
```

```
    def load_cities(self):
        try:
            with open('config/cities.json', 'r',
encoding='utf-8') as f:
                return json.load(f)
        except (FileNotFoundError, json.JSONDecodeError):
            return ["Москва", "Санкт-Петербург", "Пенза",
 "Новосибирск", "Екатеринбург"]
    def save_cities(self):
        current_text = self.city_combo.currentText()
        if current_text and current_text not in self.cities:
            self.cities.append(current_text)
            with open('config/cities.json', 'w', encoding='utf-8')
as f:
                json.dump(self.cities, f,
 ensure_ascii=False, indent=4)

    def add_city(self):
        current_text = self.city_combo.currentText()
        if current_text and current_text not in self.cities:
            self.city_combo.addItem(current_text)
            self.city_combo.setCurrentText(current_text)
            self.save_cities()

    def save_settings(self):
        self.add_city()
        self.settings = {
            "rate": int(self.rate_input.text()),
            "volume": float(self.volume_input.text()),
            "language": self.language_combo.currentText(),
            "city_default": self.city_combo.currentText()
        }
        with open('config/settings.json', 'w',
encoding='utf-8') as f:
            json.dump(self.settings, f, ensure_ascii=
False, indent=4)
        self.close()
class AssistantThread(QThread):
    update_signal = pyqtSignal(str)

    def __init__(self, assistant):
        super().__init__()
        self.assistant = assistant

    def run(self):
        self.assistant.run()

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Квант")
        self.setGeometry(100, 100, 600, 400)
        self.init_ui()
```

```
    self.load_logs()

    def init_ui(self):
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        main_layout = QVBoxLayout()

        log_group = QGroupBox("Журнал работы системы")
        log_group.setFont(QFont('Arial', 10, QFont.Bold))
        group_layout = QVBoxLayout()

        info_label = QLabel("Последние события и взаимодействия:")
        info_label.setStyleSheet("color: #555; font-style: italic;")
        group_layout.addWidget(info_label)

        self.log_output = QTextEdit()
        self.log_output.setReadOnly(True)
        self.log_output.setStyleSheet("""
                    QTextEdit {
                        background-color: #f8f8f8;
                        border: 1px solid #ddd;
                        padding: 5px;
                        font-family: 'Calibri';
                        font-size: 15px;
                    }
                """)
        group_layout.addWidget(self.log_output)
        #main_layout.addWidget(self.log_output)

        log_group.setLayout(group_layout)
        main_layout.addWidget(log_group)

        button_layout = QHBoxLayout()

        self.settings_btn = QPushButton("Настройки")
        self.settings_btn.setIcon(QIcon.fromTheme("preferences-
system"))
        self.settings_btn.clicked.connect(self.open_settings)
        button_layout.addWidget(self.settings_btn)

        self.refresh_btn = QPushButton("Обновить")
        self.refresh_btn.setIcon(QIcon.fromTheme("view-refresh"))
        self.refresh_btn.clicked.connect(self.load_logs)
        button_layout.addWidget(self.refresh_btn)

        main_layout.addLayout(button_layout)
        central_widget.setLayout(main_layout)
    def load_logs(self):
            try:
                with open('config/log.txt', 'r', encoding='utf-8')
    as f:
                    self.log_output.setPlainText(f.read())
```

```
   self.highlighter = LogHighlighter(self.log_output.document())

                # Прокрутка вниз к новым сообщениям
                cursor = self.log_output.textCursor()
                cursor.movePosition(cursor.End)
                self.log_output.setTextCursor(cursor)
        except FileNotFoundError:
            self.log_output.setPlainText("Логи не найдены. Журнал
будет создан автоматически при работе системы.")
            self.highlighter =
LogHighlighter(self.log_output.document())
    def open_settings(self):
        settings_window = SettingsWindow(self)
        settings_window.exec_()

class TrayApp:
    def __init__(self):
        self.app = QApplication(sys.argv)
        self.app.setQuitOnLastWindowClosed(False)

        self.main_window = MainWindow()
        self.tray_icon = QSystemTrayIcon(QIcon("icon.png"))

        self.init_tray_menu()
        self.start_assistant()

    def init_tray_menu(self):
        self.tray_menu = QMenu()

        self.open_action = QAction("Открыть")
        self.open_action.triggered.connect(self.show_window)
        self.tray_menu.addAction(self.open_action)

        self.settings_action = QAction("Настройки")

self.settings_action.triggered.connect(self.main_window.open_settings)
        self.tray_menu.addAction(self.settings_action)

        self.exit_action = QAction("Выход")
        self.exit_action.triggered.connect(self.exit_app)
        self.tray_menu.addAction(self.exit_action)

        self.tray_icon.setContextMenu(self.tray_menu)
        self.tray_icon.show()
        self.tray_icon.activated.connect(self.on_tray_icon_activated)

    def start_assistant(self):
        self.assistant = VoiceAssistant()
        self.assistant_thread = AssistantThread(self.assistant)
        self.assistant_thread.update_signal.connect(self.update_logs)
        self.assistant_thread.start()
    def update_logs(self, message):
        with open('config/log.txt', 'a', encoding='utf-8') as f:
            f.write(message + "\n")
        if self.main_window.isVisible():
```

```
  self.main_window.load_logs()

    def show_window(self):
        self.main_window.load_logs()
        self.main_window.show()

    def on_tray_icon_activated(self, reason):
        if reason == QSystemTrayIcon.DoubleClick:
            self.show_window()

    def exit_app(self):
        self.assistant.is_running = False
        self.assistant_thread.quit()
        self.tray_icon.hide()
        self.app.quit()

    def run(self):
        sys.exit(self.app.exec_())

if __name__ == "__main__":
    app = TrayApp()
    app.run()
```

Файл «assistant.py»

```
from datetime import datetime
from kernel.commands import CommandHandler
from kernel.speech import SpeechEngine

class VoiceAssistant:
    def __init__(self):
        self.speech = SpeechEngine()
        self.command = CommandHandler()

    def run(self):
        self.speech.say('Здравствуй!')
        print("Начал работу!")
        try:
            with open('config/log.txt', 'a', encoding='utf-8') as f:
                f.write(f"=== Сессия началась:
{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} ===\n\n")
        except Exception as e:
            print(f"Ошибка записи в файл: {e}")
        while True:
            text = self.speech.listen()

            if text:  # Если команда получена после ключевого слова
                print(f"Распознано: {text}")
                response = self.command.handle(text.lower())
                self.speech.say(response)
                try:
                    with open('config/log.txt', 'a', encoding=
'utf-8') as f:
```

```
                        f.write(f"Запрос: {text}\nОтвет:
{response}\n\n")
                except Exception as e:
                  print(f"Ошибка записи в файл: {e}")
```

Файл «speech.py»

```
import numpy as np
import speech_recognition as sr
import pyttsx3
import wave
from vosk import Model, KaldiRecognizer, SetLogLevel
import pyaudio
import json

class SpeechEngine:
    def __init__(self):
        self.recognizer = sr.Recognizer()
        self.keyword = "квант"
        #self.is_active = False
        self.audiofile_path_start = "config/sound_start.wav"
        self.audiofile_path_success = "config/sound_success.wav"
        self.audiofile_path_end = "config/sound_end.wav"
        self.volume_factor = 0.3
        self.log_off = SetLogLevel(-1)

        self.model = Model("models/vosk-model-small-ru-0.22")
        self.recognizer_vosk = KaldiRecognizer(self.model, 16000)
        self.mic = pyaudio.PyAudio()
        self.stream = self.mic.open(format=pyaudio.paInt16,
channels=1, rate=16000, input=True, frames_per_buffer=8192)

    def _play_sound(self, file_path):
        with wave.open(file_path, 'rb') as wf:
            # Создаем объект PyAudio
            p = pyaudio.PyAudio()
            # Открываем поток для воспроизведения
            stream =
p.open(format=p.get_format_from_width(wf.getsampwidth()),
                           channels=wf.getnchannels(),
                           rate=wf.getframerate(),
                           output=True)
            # Читаем и воспроизводим данные с пониженной громкостью
            chunk = 1024
            data = wf.readframes(chunk)
            while data:
                # Преобразуем байты в массив numpy и уменьшаем
громкость
                audio_data = np.frombuffer(data, dtype=np.int16)
                audio_data = (audio_data *
self.volume_factor).astype(np.int16)
                # Воспроизводим измененные данные
                stream.write(audio_data.tobytes())
                data = wf.readframes(chunk)
```

```
            # Закрываем поток и PyAudio
            stream.close()
            p.terminate()
        def listen(self):
            while True:
                try:
                    data = self.stream.read(4096,
exception_on_overflow=False)
                    if self.recognizer_vosk.AcceptWaveform(data):
                        result =
json.loads(self.recognizer_vosk.Result())
                        text = result.get("text", "").lower()
                        #print(text)
                        if self.keyword in text:
                            #self.is_active = True
                            print("Активирован! Слушаю команду...")
                            with sr.Microphone() as source:

self._play_sound(self.audiofile_path_start)
                                print("===============")
                                command_audio =
self.recognizer.listen(source, phrase_time_limit=8)
                                command =
self.recognizer.recognize_google(command_audio, language="ru-RU")

self._play_sound(self.audiofile_path_success)
                                return command
                except OSError as e:
                    print(f"Аудио ошибка: {e}, продолжаю работу...")
                    continue
                except sr.UnknownValueError:
                    self._play_sound(self.audiofile_path_end)
                    continue
                except Exception as e:
                    print(f"Ошибка распознавания: {e}")
                    return ""

    def say(self, text):
        try:
            with open("config/settings.json", "r", encoding="utf-8")
as f:
                settings = json.load(f)
            language = settings["language"]
            rate = settings["rate"]
            volume = settings["volume"]

            engine = pyttsx3.init() #offline

            engine.setProperty('rate', rate)
            engine.setProperty('volume', volume)

            voices = engine.getProperty('voices')
            if language == 'ru':
                for voice in voices:
```

```
                    if 'russian' in voice.name.lower():
                        engine.setProperty('voice', voice.id)
                        break
        def listen(self):
            while True:
                try:
                    data = self.stream.read(4096,
exception_on_overflow=False)
                    if self.recognizer_vosk.AcceptWaveform(data):
                        result =
json.loads(self.recognizer_vosk.Result())
                        text = result.get("text", "").lower()
                        #print(text)
                        if self.keyword in text:
                            #self.is_active = True
                            print("Активирован! Слушаю команду...")
                            with sr.Microphone() as source:

self._play_sound(self.audiofile_path_start)
                                print("===============")
                                command_audio =
self.recognizer.listen(source, phrase_time_limit=8)
                                command =
self.recognizer.recognize_google(command_audio, language="ru-RU")

self._play_sound(self.audiofile_path_success)
                                return command
                except OSError as e:
                    print(f"Аудио ошибка: {e}, продолжаю работу...")
                    continue
                except sr.UnknownValueError:
                    self._play_sound(self.audiofile_path_end)
                    continue
                except Exception as e:
                    print(f"Ошибка распознавания: {e}")
                    return ""

    def say(self, text):
        try:
            with open("config/settings.json", "r", encoding="utf-8")
as f:
                settings = json.load(f)
            language = settings["language"]
            rate = settings["rate"]
            volume = settings["volume"]

            engine = pyttsx3.init() #offline

            engine.setProperty('rate', rate)
            engine.setProperty('volume', volume)

            voices = engine.getProperty('voices')
            if language == 'ru':
                for voice in voices:
```

```
                    if 'russian' in voice.name.lower():
                        engine.setProperty('voice', voice.id)
                        break
            else:  # английский по умолчанию
                engine.setProperty('voice', voices[0].id)

            engine.say(text)
            engine.runAndWait()

        except Exception as e:
            print(f"Ошибка синтеза речи: {e}")
```

Файл «commands.py»

```
class CommandHandler:
    def handle(self, text):
        if any(word in text for word in ["погода", "погоду",
"погоды"]):
            from instructions.weather import get_weather
            return get_weather(text)
        elif any(word in text for word in ["открой", "запусти"]):
            app_name = text.replace("открой", "").replace("запусти",
"").strip()
            from instructions.open_app import open_app
            return open_app(app_name)
        elif any(word in text for word in ["найди", "найти",
"загугли", "поищи", "поиск"]):
            from instructions.web_src import search_in_browser
            return search_in_browser(text)
        elif any(word in text for word in ["что такое", "кто такой",
"определение", "википедия"]):
            # Извлекаем термин после ключевых слов
            term = text.split("что такое")[-1].split("кто такой")[-
1].strip()
            from instructions.wiki_src import wiki_search
            return wiki_search(term)
        elif any(word in text for word in ["случайное", "рандомное",
"сгенерир", "выбери", "число"]): # Сгенерируй
            from instructions.gen_num import generation_number
            return generation_number(text)
        elif "переведи" in text: # перевод
            from instructions.translator import translate
            # Извлекаем язык и фразу (пример: "переведи на
английский как дела")
            parts = text.split()
            if "английский" in text:
                phrase = ' '.join(parts[parts.index("английский") +
1:])
                lang = 'en'
                return translate(phrase, lang)
            elif "русский" in text:
                phrase = ' '.join(parts[parts.index("русский") + 1:])
                lang = 'ru'
                return translate(phrase, lang)
            else:
```

```
                return "Укажите язык перевода: 'на английский'
или 'на русский'"
        elif "время" in text:
            from instructions.time import get_time
            return get_time()

        else:
               return "Неизвестная команда. Попробуйте ещё раз."
```

Файл «weather.py»

```
import json
from dotenv import load_dotenv
import os
from datetime import datetime, timedelta
import requests
import pymorphy3

def city_to_nominative(city):
        morph = pymorphy3.MorphAnalyzer()
        lemma = morph.parse(city)[0].normal_form
        return lemma

def get_weather(text):
    with open("config/settings.json", "r", encoding="utf-8") as f:
        settings = json.load(f)
    city = settings["city_default"]
    try:
        pretext = " в "
        if pretext in text:
            words = text.split()
            #words = text.split(pretext, maxsplit=1)
            #if len(words) > 1 and words[1].strip():
            for i in range(len(words) - 1):
                if words[i].lower() == "в":
                    has_preposition = True
                    city = city_to_nominative(words[i + 1])
                    break
        load_dotenv("config/api.env")
        api_key = os.getenv("WEATHER_API_KEY")

        forecast = requests.get(
            f"http://api.openweathermap.org/data/2.5/forecast?q=
{city}&appid={api_key}&units=metric&lang=ru"
        ).json()
        weekdays_ru = {
            'Monday': 'Понедельник',
            'Tuesday': 'Вторник',
            'Wednesday': 'Среда',
            'Thursday': 'Четверг',
            'Friday': 'Пятница',
            'Saturday': 'Суббота',
            'Sunday': 'Воскресенье'
        }
```

```
        forecast_text = ""
        for i in range(0, 4):
            day = datetime.now() + timedelta(days=i)
            day_str = day.strftime("%Y-%m-%d")
            daily = [f for f in forecast['list'] if
f['dt_txt'].startswith(day_str)]
            if not daily:
                continue
            avg_temp = sum(f['main']['temp'] for f in daily) /
len(daily)
            if i == 0:
                day_name = "Сегодня"
            elif i == 1:
                day_name = "Завтра"
            else:
                eng_day = day.strftime("%A")
                day_name = weekdays_ru.get(eng_day, eng_day)
            forecast_text += (
                f"{day_name}: {avg_temp:.0f}°,
{daily[0]['weather'][0]['description']}\n"
            )
        print(f"Прогноз в городе {city}:\n" + forecast_text)
        return f"Прогноз в городе {city}:\n" + forecast_text
    except Exception as e:
        print(f"Ошибка: {e}")
        return "Неверный запрос или название города."
```

Файл «wiki_src.py»

```
import wikipediaapi
import requests
from bs4 import BeautifulSoup
from urllib.parse import quote
#from core.utils import clean_text
import logging
import re
from typing import Optional
import os
from dotenv import load_dotenv

# Настройки
WIKI_LANG = 'ru'  # Язык Википедии
SUMMARY_LENGTH = 3  # Количество предложений в сокращенном ответе

def clean_wiki_query(raw_query: str) -> Optional[str]:
    # Удаляем ключевые фразы с помощью регулярного выражения
    query = re.sub(
r'(?:что|кто)\s+(?:такое|такой)|определение|поищи
|найди|википедия|расскажи\s+(?:про|о)|в\s+википедии',
        '',
        raw_query,
        flags=re.IGNORECASE
    ).strip()
```

```
    if not query:
        return None
    # Удаляем вопросительные знаки и точки в конце
    query = re.sub(r'[?.!]+$', '', query).strip()
    # Удаляем обрамляющие кавычки (но сохраняем внутренние)
    query = re.sub(r'^["\'](.*)["\']$', r'\1', query)
    # Заменяем последовательности пробелов/переносов на один пробел
    query = ' '.join(query.split())
    query = re.sub(r'[^\w\s\-—\'\"]', '', query, flags=re.UNICODE)
    # Обрезаем слишком длинные запросы (первые 100 символов)
    if len(query) > 100:
        query = query[:100].rsplit(' ', 1)[0] + '...'
    return query if query else None

class WikipediaSearch:
    def __init__(self):
        self.wiki = wikipediaapi.Wikipedia(
            language=WIKI_LANG,
            extract_format=wikipediaapi.ExtractFormat.WIKI,
            user_agent = os.getenv("WIKI_USER_AGENT")
        )
    def get_definition(self, term, short = True):
        try:
            page = self.wiki.page(term)

            if not page.exists():
                return self._try_alternative_search(term)

            # Форматируем результат
            if short:
                return self._get_short_definition(page)
            return self._get_full_definition(page)

        except Exception as e:
            logging.error(f"Ошибка поиска в Википедии: {e}")
            return "Не удалось получить информацию. Попробуйте
уточнить запрос."

    def _get_short_definition(self, page) -> str:
        """Возвращает краткое определение (первые предложения)"""
        summary = page.summary.split('. ')
        short_summary = '. '.join(summary[:SUMMARY_LENGTH]) + '.' if
len(summary) > 1 else page.summary
        print (f"{short_summary}")
        return f"{short_summary}"

    def _get_full_definition(self, page) -> str:
        """Возвращает полное определение с разделами"""
        sections = []
        for section in page.sections:

sections.append(f"\n\n{section.title}:\n{section.text[:500]}...")
        return f"{page.title}:\n{page.summary}
{''.join(sections[:3])}"
```

```
    def _try_alternative_search(self, term: str) -> str:
        """Альтернативный поиск через веб-скрапинг"""
        try:
            url =
f"https://{WIKI_LANG}.wikipedia.org/wiki/{quote(term)}"
            response = requests.get(url, timeout=5)

            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')
                paragraphs = soup.select('div.mw-parser-output > p')

                for p in paragraphs:
                    if p.text.strip():
                        return f"Найдено: {p.text.split('.')[0]}..."

            # Если ничего не найдено, предлагаем варианты
            search_results = self.wiki.search(term)
            if search_results:
                return (f"Точного определения не найдено. Возможно,
вы имели в виду: "
                        f"{', '.join(search_results[:3])}")

            return "Ничего не найдено. Попробуйте другой запрос."
        except:
            return "Не корректный запрос или страница на русской
Википедии отсутствует."
# Интерфейс для командного обработчика
def wiki_search(term):
    load_dotenv("config/api.env")
    return WikipediaSearch().get_definition(term)
```

Файл «web_src.py»

```
import webbrowser
import urllib.parse
import re

def clean_query(query):
    # Удаляем команды поиска
    query = re.sub(

r'(найди|найти|загугли|поищи|ищи|поиск|гугл|искать|search|find)\s*',
        '',
        query,
        flags=re.IGNORECASE
    ).strip()
    # Удаляем все символы, кроме букв, цифр, пробелов, дефисов и
точек
    query = re.sub(r'[^\w\s\-\.]', '', query, flags=re.UNICODE)

    # Удаляем лишние пробелы
    query = re.sub(r'\s+', ' ', query).strip()
    return query
```

```
   def search_in_browser(query):
       query = clean_query(query)
       # Кодируем запрос для URL
       encoded_query = urllib.parse.quote_plus(query)
       # Формируем URL для поиска в Google
       search_url = f"https://www.google.com/search?q={encoded_query}"
       webbrowser.open(search_url)
    return "Поиск осуществлён"
```

Файл «translator.py»

```
import requests
from googletrans import Translator
#from core.utils import clean_text
import logging

def translate(text, target_language='en'):
    translator = Translator(service_urls=['translate.google.com'])
    try:
        # Определение исходного языка (автоопределение)
        translation = translator.translate(text, dest=target_language)

        # Логирование для отладки
        logging.info(f"Перевод: {text} -> {translation.text}
({translation.src}->{translation.dest})")
        print(translation.text)
        return translation.text

    except requests.exceptions.ConnectionError:
        return "Нет подключения к интернету для перевода"
    except Exception as e:
        logging.error(f"Ошибка перевода: {e}")
        return "Не удалось выполнить перевод"
```

Файл «open_app.py»

```
import os
import subprocess
import webbrowser
import winreg
from typing import Optional

class AppOpener:
    @staticmethod
    def find_in_registry(app_name: str) -> Optional[str]:
        """
        Ищет путь к приложению в реестре Windows
        Возвращает полный путь к EXE-файлу или None
        """
        try:
            with winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,

r"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths") as key:
                try:
                    subkey = winreg.OpenKey(key, f"{app_name}.exe")
                    path, _ = winreg.QueryValueEx(subkey, "")
```

```
                    return path
                except FileNotFoundError:
                    return None
        except Exception as e:
            print(f"Ошибка: {e}")
            return None

    @staticmethod
    def find_in_program_files(app_name: str) -> Optional[str]:
        """
        Ищет приложение в стандартных папках Program Files
        """
        search_dirs = [
            os.environ["ProgramFiles"],
            os.environ["ProgramFiles(x86)"],
            os.path.join(os.environ["APPDATA"], "Local", "Programs"),
            os.path.join(os.environ["LOCALAPPDATA"], "Programs"),
        ]

        target_exe = f"{app_name.lower()}.exe"

        for dir in search_dirs:
            if not os.path.exists(dir):
                continue

            for root, _, files in os.walk(dir):
                for file in files:
                    if file.lower() == target_exe:
                        return os.path.join(root, file)
        return None

    @classmethod
    def open_application(cls, app_name_default: str) -> str:
        app_mapping_two = {
            "paint": "mspaint",
            "pain": "mspaint",
            "блокнот": "notepad",
            "калькулятор": "calc",
            "проводник": "explorer",
            "диспетчер задач": "taskmgr",
        } # "командная строка": "cmd",
        app_mapping_one = {
            "word": "winword",
            "powerpoint": "powerpnt",
            "power point": "powerpnt",
        } # excel
        app_name = app_mapping_one.get(app_name_default,
app_name_default)

        # 0. Браузер по умолчанию
        if app_name == "браузер":
            try:
                webbrowser.open("https://yandex.ru")
                return f"Открываю {app_name_default}"
```

```
            except Exception as e:
                print(f"Ошибка: {e}")

        # 1. Поиск в реестре
        reg_path = cls.find_in_registry(app_name)
        if reg_path:
            try:
                subprocess.Popen(reg_path, shell=True)
                print(f"Открываю {app_name_default}")
                return f"Открываю {app_name_default}"
            except Exception as e:
                print(f"Ошибка: {e}")

        # 2. Попытка запуска через shell
        if app_name in app_mapping_two:
            app_to_open = app_mapping_two[app_name]
            try:
                subprocess.Popen(app_to_open, shell=True)
                print(f"Открываю {app_name_default}")
                return f"Запускаю {app_name_default}"
            except Exception as e:
                print(f"Ошибка: {e}")

        # 3. Поиск в Program Files
        app_path = cls.find_in_program_files(app_name)
        if app_path:
            try:
                os.startfile(app_path)
                print(f"Нашел и открываю {app_name_default}")
                return f"Нашел и открываю {app_name_default}"
            except Exception as e:
                print(f"Ошибка: {e}")
        return f"Не удалось найти приложение {app_name_default}"
# Интерфейс для командного обработчика
def open_app(app_name: str) -> str:
    return AppOpener.open_application(app_name)
```

Файл «gen_num.py»

```
import random
import re
#from core.utils import clean_text
import logging


def generation_number(text):
    try:
        # Очищаем текст и приводим к нижнему регистру
        #cleaned_text = clean_text(text)

        # Ищем паттерны типа "от X до Y"
        matches =
re.findall(r'(?:от|между)\s*(\d+)\s*(?:до|и)\s*(\d+)', text.lower())
```

```
        if not matches:
            return "Не удалось распознать диапазон."

        # Извлекаем числа из первого совпадения
        from_num, to_num = map(int, matches[0])

        # Проверяем валидность диапазона
        if from_num >= to_num:
            return "Первое число должно быть меньше второго"

        # Генерируем случайное число
        result = random.randint(from_num, to_num)

        # Логируем для отладки
        logging.info(f"Сгенерировано число: {result} (диапазон
{from_num}-{to_num})")
        print(result)
        return f"Случайное число: {result}"

    except Exception as e:
        logging.error(f"Ошибка генерации числа: {e}")
              return "Произошла ошибка при генерации числа"
```

Файл «log_highlighter.py»

```
from PyQt5.QtCore import QRegExp
from PyQt5.QtGui import QTextCharFormat, QSyntaxHighlighter, QColor


class LogHighlighter(QSyntaxHighlighter):
    def __init__(self, parent=None):
        super(LogHighlighter, self).__init__(parent)

        self.highlighting_rules = []

        # Формат для "Запрос:"
        query_format = QTextCharFormat()
        query_format.setForeground(QColor(0, 128, 0))  # Зелёный
        self.highlighting_rules.append((QRegExp("Запрос:"),
query_format))

        # Формат для "Ответ:"
        response_format = QTextCharFormat()
        response_format.setForeground(QColor(255, 165, 0))  #
Оранжевый
        self.highlighting_rules.append((QRegExp("Ответ:"),
response_format))

        # Формат для строк с сессиями
        session_format = QTextCharFormat()
        session_format.setForeground(QColor(0, 0, 255))  # Синий
        session_format.setFontWeight(75)
```

```
        self.highlighting_rules.append((QRegExp("=== Сессия
началась:.*==="), session_format))

    def highlightBlock(self, text):
        for pattern, format in self.highlighting_rules:
            expression = QRegExp(pattern)
            index = expression.indexIn(text)
            while index >= 0:
                length = expression.matchedLength()
                self.setFormat(index, length, format)
                index = expression.indexIn(text, index + length)
```

Файл «settings.json»

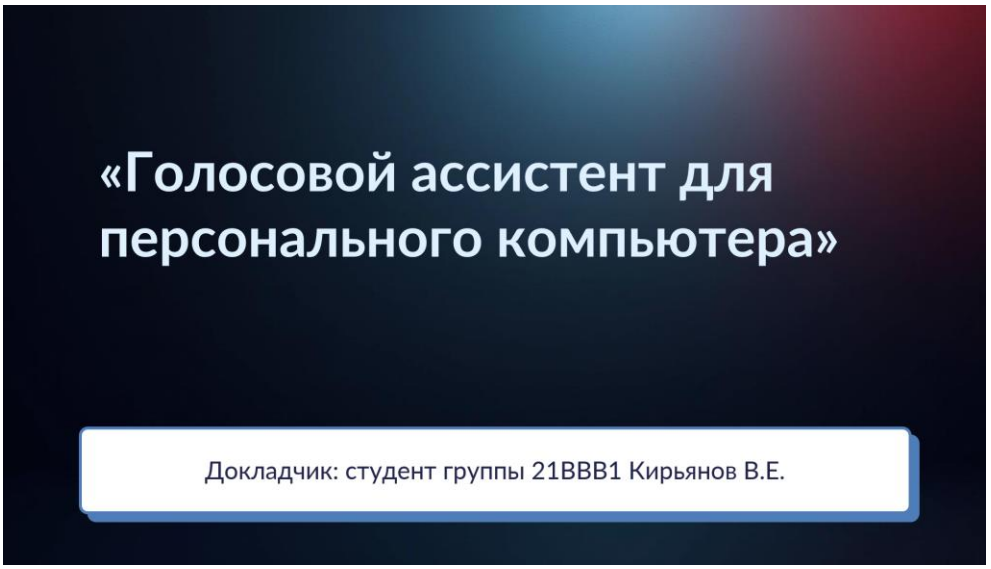
```
{
    "rate": 300,
    "volume": 0.9,
    "language": "ru",
    "city_default": "Пенза"
}
```

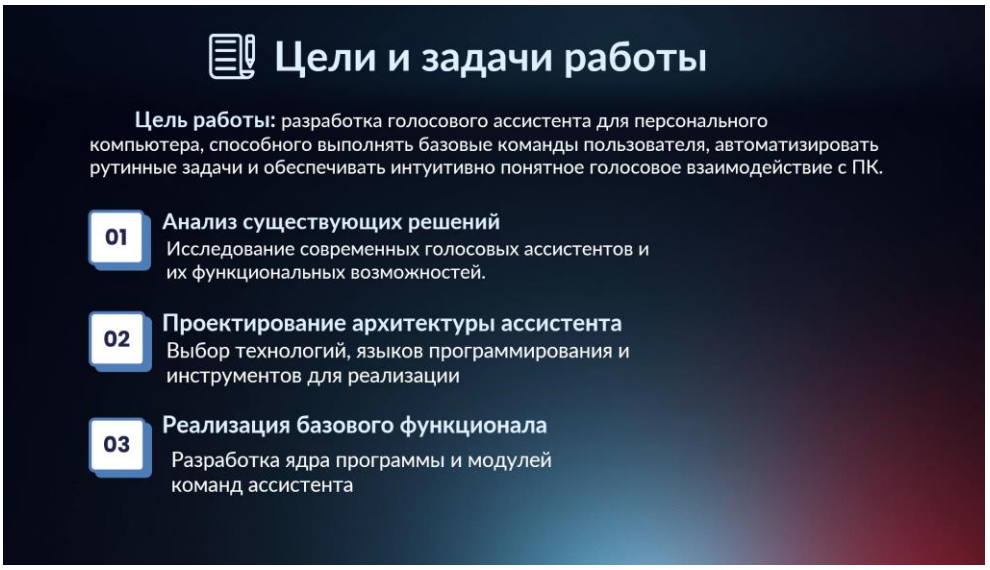
# Приложение В

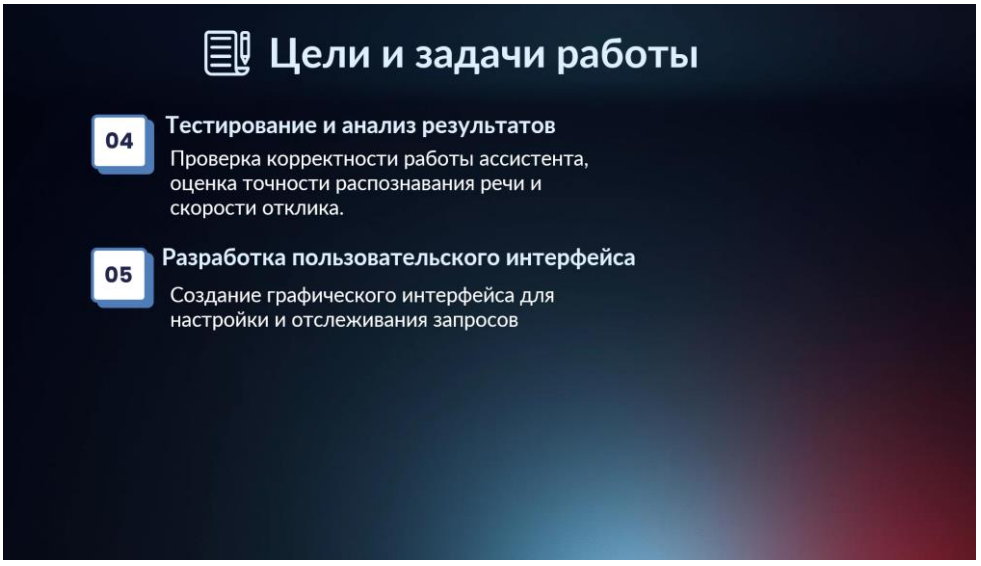
(Обязательное)

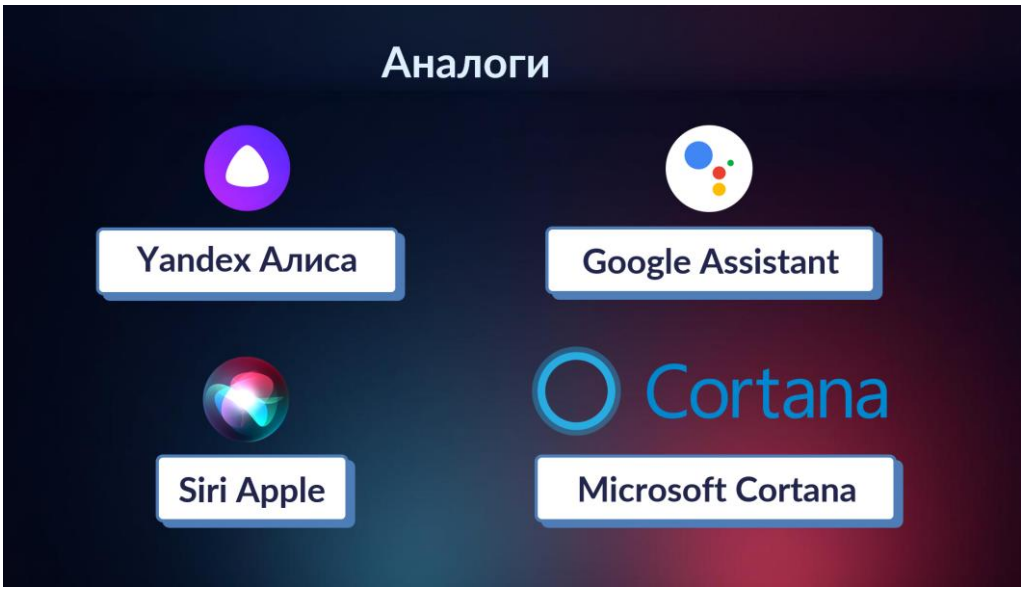
Голосовой ассистент для персонального компьютера

Презентация


«Голосовой ассистент для персонального компьютера»
Докладчик: студент группы 21ВВВ1 Кирьянов В.Е.

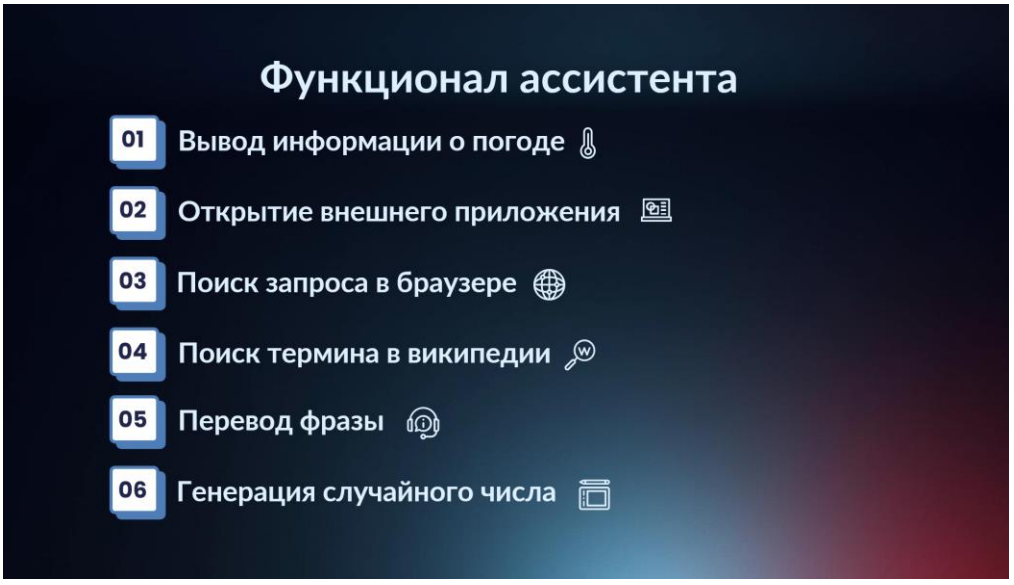

Цели и задачи работы
Цель работы: разработка голосового ассистента для персонального компьютера, способного выполнять базовые команды пользователя, автоматизировать рутинные задачи и обеспечивать интуитивно понятное голосовое взаимодействие с ПК.
01
Анализ существующих решений
Исследование современных голосовых ассистентов и их функциональных возможностей.
02
Проектирование архитектуры ассистента
Выбор технологий, языков программирования и инструментов для реализации
03
Реализация базового функционала
Разработка ядра программы и модулей команд ассистента


Цели и задачи работы
04
Тестирование и анализ результатов
Проверка корректности работы ассистента, оценка точности распознавания речи и скорости отклика.
05
Разработка пользовательского интерфейса
Создание графического интерфейса для настройки и отслеживания запросов

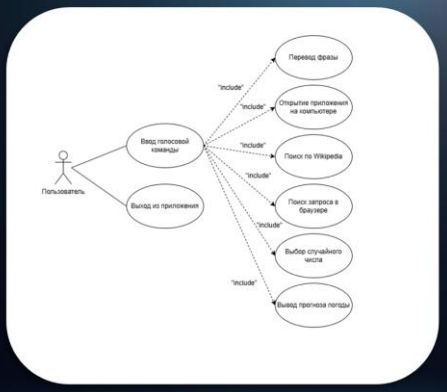



## Выбор модели распознавания

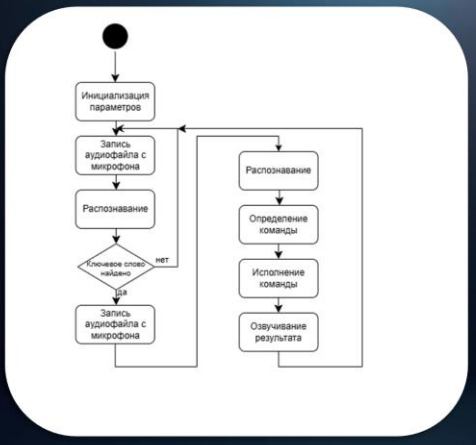
| Модель распознавания | Достоинства | Недостатки |
|---|---|---|
| Whisper | Высокая точность распознавания. Бесплатная модель. Многоязычность. Работает без интернета. | Ресурсоёмкость. Отсутствует потоковая обработка. |
| Google Speech Recognition API | Простота использования. Хорошая точность распознавания. Поддержка многоязычности. | Работает только с интернетом. Ограничения бесплатной версии. Скорость зависит от интернета. |
| Vosk | Работает без интернета. Легковесная модель. Присутствует потоковая обработка. | Сложность настройки модели. Точность ниже, чем у модели Whisper. |
| DeepSpeech | Работает без интернета, отрытый исходный код. | Устаревшая модель. Низкая точность распознавания. |
| Wav2Vec 2.0 | Высокая точность распознавания. | Сложность использования модели. Ресурсоёмкость. |



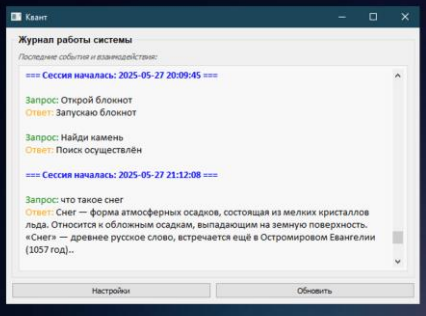
## Диаграмма вариантов использования



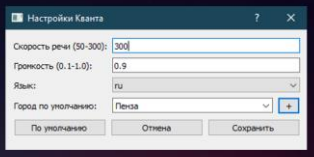

## Диаграмма деятельности



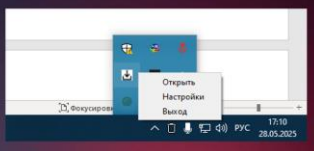

## Интерфейс ассистента




История запросов пользователя




Настройки ассистента




Ярлык в системном трее

# Заключение

Таким образом, на основании тестирования ассистента все его функции работают корректно. Также, в результате тестирования голосового помощника средняя оценка точности распознавания команд составила ≈70 %. Ошибки распознавания в связаны с не идеальностью моделей распознавания, нечётким произношением слов или с использованием сложных фонетических конструкций в речи пользователя. Проведённое тестирование подтвердила работоспособность голосового ассистента и его функций. Результаты соответствует заявленным ожиданиям.

В дальнейшем программу можно усовершенствовать, путём модифицирования уже существующих функций и добавления новых, например: составления расписания, создания заметок, взаимодействия с музыкальным плеером и других.

Таким образом, реализован прототип голосового ассистента, который удовлетворяет требованиям задачи, следовательно поставленная цель достигнута. Такой голосовой помощник может быть внедрён в повседневную жизнь, чтобы упростить различные задачи и повысить эффективность работы за компьютером.

# Спасибо за внимание!