Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Институт информационных технологий и анализа данных

наименование института (факультета)

Допускаю к защите

Руководитель А. В. Жуков

И.О. Фамилия

Разработка голосового анкетирования для платформы "Яндекс-Диалоги".

Наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту по дисциплине

«WEB-программирование»

1.020.00.00 ПЗ

обозначение документа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент | АСУб-21-1 |  |  |  | В.В. Пашкова |
| шифр группы |  | подпись |  | И. О. Фамилия |
|  |  |  |  |  |  |
| Нормоконтроль |  |  |  |  | А.В. Жуков |
|  |  | подпись |  | И. О. Фамилия |

Курсовой проект защищена с оценкой **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Иркутск 2024 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

ЗАДАНИЕ

НА КУРСОВОЙ ПРОЕКТ

|  |  |  |  |
| --- | --- | --- | --- |
| По курсу | WEB-программирование | | |
| Студенту | Пашковой В. В. | | |
|  | | (фамилия, инициалы) | |
| Тема проекта: | | Разработка голосового анкетирования для платформы | |
| "Яндекс-Диалоги". | | | |
| Исходные данные: | | | Создать информационную модель по Яндекс.Диалогам. |
| Создать базу данных, которая содержит информацию о существующих | | | |
| голосовых помощниках. Создать разговорный интерфейс для анкеты | | | |
| «профиль здоровья». Создать подсистему управления диалогом на основе | | | |
| общей модели анкеты. | | | |
| Рекомендуемая литература: | | | |
| 1. Документация по диалогам | | | |
|  | | | |
| 2. Облачные системы (Яндекс) | | | |
|  | | | |
| 3. Спросить Кирилла | | | |
|  | | | |
| 4. | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | | |  |  |
| Графическая часть на | – | | листах. | | |  |  |
| Дата выдачи задания | «» | 2024 г. | | | |  |  |
| Задание получил | | | |  |  |  |  | В.В. Пашкова |
|  | | | |  |  | подпись |  | И.О.Фамилия |

|  |  |  |  |
| --- | --- | --- | --- |
| Дата представления проекта руководителю | «» 2024 г. | | |
| Руководитель курсового проекта |  |  | А.В. Жуков |
|  | подпись | И.О.Фамилия |

Содержание

[Введение 5](#_Toc167054940)

[1 Постановка задачи 6](#_Toc167054941)

[2 Обоснование необходимости разработки системы голосового анкетирования 7](#_Toc167054942)

[2.1 Проблематика традиционных методов сбора данных 7](#_Toc167054943)

[2.2 Необходимость внедрения более удобных и доступных методов 8](#_Toc167054944)

[2.3 Перспективы и преимущества использования голосовых технологий на платформе Яндекс.Диалоги 9](#_Toc167054945)

[2.4 Обоснование разработки прототипа системы голосового анкетирования 10](#_Toc167054946)

[3 Знакомство с платформой Яндекс.Диалоги 12](#_Toc167054947)

[4 Структура информационной модели голосового помощника 13](#_Toc167054948)

[4.1 Описание информационной модели 13](#_Toc167054949)

[4.2 База данных на основе информационной модели 15](#_Toc167054950)

[5 База данных, содержащая информацию о голосовых помощниках, разработанных на платформах Яндекс.Диалоги и Сбер.Салют. 16](#_Toc167054951)

[6 Описание архитектуры и функциональных возможностей разрабатываемого голосового помощника 18](#_Toc167054952)

[6.1 Архитектура. 18](#_Toc167054953)

[6.2 Функциональные возможности. 19](#_Toc167054954)

[6.3 Структура словарей 21](#_Toc167054955)

[6.4 Реализация функции перехода между состояниями 22](#_Toc167054956)

[6.5 Листинг программы 23](#_Toc167054957)

[7 Описание архитектуры и функциональных возможностей разрабатываемого генератора 37](#_Toc167054958)

[8 Описание архитектуры и функциональных возможностей разрабатываемого конвертера 38](#_Toc167054959)

[9 Тестирование 39](#_Toc167054960)

[10 Руководство пользователя 40](#_Toc167054961)

[10.1 Алгоритм работы с программой 40](#_Toc167054962)

[10.1.1. Пользователи системы 40](#_Toc167054963)

[10.1.2. Порядок использования инструментального средства 40](#_Toc167054964)

[Заключение 41](#_Toc167054965)

[Список использованных источников 42](#_Toc167054966)

# Введение

В современном мире цифровые технологии проникают во все сферы жизни, упрощая и автоматизируя многие процессы. Одним из перспективных направлений является развитие голосовых интерфейсов и голосовых помощников. Голосовое взаимодействие позволяет сделать общение с цифровыми системами более естественным и удобным для пользователей.

Одной из областей, где голосовые технологии могут найти широкое применение, является проведение опросов и анкетирования. Традиционные способы сбора данных, такие как бумажные анкеты или онлайн-опросы, имеют ряд недостатков: они требуют значительных временных и трудовых затрат, а также могут быть неудобны для некоторых категорий респондентов.

Внедрение голосового анкетирования позволит преодолеть эти ограничения. Респонденты смогут проходить опросы, используя только голосовые команды, что значительно упростит процесс и сделает его более доступным для людей с ограниченными возможностями или тех, кто предпочитает голосовое взаимодействие.

В рамках данного курсового проекта будут разработаны: прототип системы голосового анкетирования на базе платформы "Яндекс-Диалоги", информационная модель по Яндекс.Диалогам, база данных с информацией о существующих ГП, генератор текста спецификации для Яндекс.Диалогов на основе модели и конвертер (не факт). Это позволит оценить перспективы и преимущества использования голосовых технологий в сфере опросов, а также выявить потенциальные проблемы и ограничения.

Проект будет состоять из следующих разделов: описание информационной модели ГП, описание БД и информационной модели, описание архитектуры и функциональных возможностей разрабатываемого ГП и конвертера, процесс их реализации, тестирование и анализ полученных результатов.

(написать в ведении что все разрабатывается на платформе создания ВА).

# Постановка задачи

1. Изучить платформу Яндекс.Диалоги (краткая справка).
2. Создать информационную модель голосового помощника Яндекс.Диалогов с использованием РБНФ.
3. Создать базу данных для хранения информации о голосовых помощниках, разработанных на платформах Яндекс.Диалоги и Сбер.Салют. (Вставить картинку и описать всё)
4. Разработать архитектуру голосового помощника. (она будет состоять из функций (блок получения информации, блок преобразования…), будет генерировать диалог, отправлять результаты на сервер. Добавить пример анкеты, экранные формы диалогов…).
5. Описать алгоритм работы с программой и голосовым помощником. рассказать что делать, указать параметры. Возможно сделать схему в star.uml с вариантами использования (которая с человечком).
6. Получить на вход структуру анкеты в форме json и модель общего разговорного интерфейса. Сгенерировать текст спецификации для Яндекс.Диалогов на основе модели.
7. Сгенерировать параметры для компонента управления диалога на основе структуры анкеты и модели разговорного интерфейса. Параметры включают в себя: алгоритм активации вопросов анкеты, связь вопросов с намерениями (разговорный интерфейс).

Добавить архитектуру всей системы (Пользователь-яндекс-)

# Обоснование необходимости разработки системы голосового анкетирования

* 1. Проблематика традиционных методов сбора данных

Для дальнейшего углубления в исследование и обеспечения полноты работы, важно чётко определить проблематику, которая будет адресована в рамках исследования.

Традиционные способы сбора данных, такие как бумажные анкеты и онлайн-опросы, несмотря на свое широкое применение, имеют ряд существенных ограничений и недостатков:

1. Временные и трудовые затраты
   * Проведение бумажных анкетирований требует значительных временных и человеческих ресурсов на распространение, сбор, обработку и ввод данных в электронный вид.
   * Онлайн-опросы также предполагают временные затраты на разработку, тестирование и публикацию анкет, а также на последующий анализ полученных данных.
   * Для организаций, регулярно проводящих опросы, эти временные и трудовые издержки становятся существенным бременем, отвлекающим от основной деятельности.
2. Неудобство для некоторых категорий респондентов
   * Бумажные анкеты могут быть неудобны для людей с ограниченными физическими возможностями, пожилых респондентов или тех, кто предпочитает использовать цифровые устройства.
   * Онлайн-опросы также могут быть недоступны для категорий населения, не имеющих устойчивого доступа к сети Интернет или не владеющих навыками работы с компьютером.
   * Эти ограничения приводят к неполному охвату целевой аудитории и искажению результатов опросов.
3. Сложность интеграции с другими системами
   * Традиционные методы сбора данных, как правило, плохо интегрируются с другими информационными системами организации.
   * Это затрудняет дальнейшую обработку, анализ и использование полученных данных в рамках единой информационной среды.

Таким образом, традиционные методы сбора данных характеризуются высокими временными и трудовыми затратами, а также ограниченностью в плане удобства и доступности для различных категорий респондентов. Эти недостатки создают необходимость поиска альтернативных, более эффективных способов проведения опросов и анкетирования.

* 1. Необходимость внедрения более удобных и доступных методов

Ограничения традиционных методов, таких как бумажные анкеты и онлайн-опросы, обуславливают необходимость внедрения более современных и удобных для респондентов способов сбора данных. В этом контексте особый интерес представляет развитие голосовых интерфейсов и внедрение систем голосового анкетирования.

1. Упрощение процесса прохождения опросов
   * Голосовое взаимодействие позволяет сделать процесс ответов на вопросы более естественным и интуитивно понятным для респондентов.
   * Респонденты смогут проходить опросы, используя только голосовые команды, что значительно упростит и ускорит процесс.
   * Это особенно актуально для людей, предпочитающих голосовое взаимодействие с цифровыми системами.
2. Повышение доступности для людей с ограниченными возможностями
   * Голосовое анкетирование открывает новые возможности для людей с ограниченными физическими возможностями, которым могут быть неудобны традиционные методы.
   * Респонденты с нарушениями зрения, опорно-двигательного аппарата или другими ограничениями смогут участвовать в опросах, используя только голосовые команды.
   * Это значительно расширит охват целевой аудитории и повысит инклюзивность проводимых исследований.
3. Интеграция с другими цифровыми системами
   * Голосовые интерфейсы легко интегрируются с различными информационными системами.
   * Это позволит автоматизировать процессы сбора, обработки и анализа данных, полученных в ходе опросов.
   * Интеграция с другими приложениями и сервисами также открывает возможности для более эффективного использования собранной информации.

Таким образом, внедрение голосового анкетирования может значительно упростить процесс прохождения опросов для респондентов, сделав его более удобным и доступным. Это особенно важно для категорий населения с ограниченными возможностями, а также для тех, кто предпочитает использовать голосовые интерфейсы. Кроме того, интеграция голосовых систем с другими информационными системами позволит повысить эффективность сбора и обработки данных.

* 1. Перспективы и преимущества использования голосовых технологий на платформе Яндекс.Диалоги

Платформа Яндекс.Диалоги обладает рядом уникальных возможностей, которые делают ее привлекательной для разработки систем голосового анкетирования.

1. Развитая экосистема голосовых технологий
   * Яндекс.Диалоги являются частью обширной экосистемы голосовых технологий Яндекса, включающей в себя голосового помощника "Алиса", речевое распознавание, синтез речи и другие передовые решения. Это обеспечивает высокое качество и надежность голосового взаимодействия, а также широкие возможности для интеграции с другими сервисами Яндекса.
2. Простота разработки и развертывания
   * Платформа Яндекс.Диалоги предоставляет удобный инструментарий для быстрой разработки, тестирования и развертывания голосовых приложений.
   * Наличие готовых компонентов, шаблонов и API значительно упрощает и ускоряет процесс создания прототипов и рабочих версий системы голосового анкетирования.
3. Широкая аудитория и распространение
   * Голосовой помощник "Алиса" и платформа Яндекс.Диалоги имеют обширную аудиторию пользователей, что повышает потенциальный охват респондентов.
   * Интеграция разработанной системы голосового анкетирования с другими сервисами Яндекса (например, Яндекс.Браузер, Яндекс.Станция) позволит расширить доступность и удобство использования для конечных пользователей.
4. Возможности аналитики и интеграции
   * Платформа Яндекс.Диалоги предоставляет развитые средства аналитики, позволяющие отслеживать и анализировать взаимодействие пользователей с голосовыми приложениями.
   * Кроме того, Яндекс.Диалоги обладают широкими возможностями интеграции с другими информационными системами, что упрощает обмен данными и автоматизацию процессов.
5. Надежность и безопасность
   * Платформа Яндекс.Диалоги отличается высокой надежностью и отказоустойчивостью, что критически важно для систем, работающих с конфиденциальными данными.
   * Кроме того, Яндекс предлагает комплексные решения для обеспечения безопасности и защиты персональных данных пользователей.

Таким образом, использование платформы Яндекс.Диалоги для разработки системы голосового анкетирования позволит максимально использовать преимущества развитой экосистемы голосовых технологий Яндекса, упростить процесс создания и развертывания решения, а также обеспечить высокую надежность, безопасность и широкий охват аудитории. Это делает Яндекс.Диалоги оптимальным выбором для реализации данного проекта.

* 1. Обоснование разработки прототипа системы голосового анкетирования

Разработка прототипа системы голосового анкетирования является важным шагом для внедрения современных технологий в процессы сбора данных. Создание такого прототипа обеспечит основу для дальнейшего развития и совершенствования решений, использующих голосовые интерфейсы для проведения опросов и анкетирования.

Первым ключевым элементом данной разработки является создание информационной модели голосового помощника на платформе Яндекс.Диалоги с использованием РБНФ. Формализация структуры и функциональности голосового помощника заложит фундамент для последующей реализации и интеграции различных компонентов системы.

Параллельно с этим, будет разработана база данных, содержащая информацию о существующих голосовых помощниках, созданных на платформах Яндекс.Диалоги и Сбер.Салют. Наличие такой базы данных обеспечит информационную поддержку при проектировании и реализации системы голосового анкетирования, позволяя учитывать особенности различных платформ.

Следующим шагом станет проектирование архитектуры самого голосового помощника. Эта архитектура будет включать в себя функциональные блоки для получения информации, ее преобразования, генерации диалога и взаимодействия с сервером. Такая структура обеспечит основу для реализации полнофункциональной системы голосового анкетирования, способной эффективно взаимодействовать с пользователями.

Для обеспечения понимания процессов, происходящих в системе, будет подробно описан алгоритм работы с программой и голосовым помощником. Это описание будет включать варианты использования, параметры и схемы взаимодействия, что упростит дальнейшее развитие и совершенствование системы.

Важным элементом разработки станет получение на входе структуры анкеты в формате JSON и модели общего разговорного интерфейса. Это позволит обеспечить гибкость и адаптивность системы голосового анкетирования, позволяя ей работать с различными типами анкет и разговорных интерфейсов.

Завершающими этапами станут генерация текста спецификации для Яндекс.Диалогов и параметров для компонента управления диалога на основе полученных входных моделей. Автоматизация этих процессов повысит эффективность разработки и упростит интеграцию системы голосового анкетирования с платформой Яндекс.Диалоги.

Таким образом, разработка прототипа системы голосового анкетирования, включающая создание информационной модели, базы данных, архитектуры, алгоритмов и генераторов, обеспечит основу для дальнейшего развития и внедрения эффективных решений в сфере опросов и анкетирования с использованием голосовых технологий.

# Знакомство с платформой Яндекс.Диалоги

Яндекс.Диалоги — это облачная платформа для создания и развертывания голосовых помощников и чат-ботов. Она предоставляет разработчикам инструменты для построения диалоговых систем с использованием технологий распознавания и синтеза речи, а также обработки естественного языка.

* + Основные возможности платформы Яндекс.Диалоги:
  + Создание диалоговых сценариев с помощью визуального конструктора или языка разметки.
  + Интеграция с различными каналами общения, такими как голосовые вызовы, мессенджеры, веб-чаты и др.
  + Использование предобученных моделей распознавания речи и обработки естественного языка.
  + Возможность кастомизации моделей под специфические задачи.
  + Масштабируемость и высокая доступность благодаря облачной инфраструктуре.
  + Аналитика и мониторинг работы голосовых помощников.

В рамках данного проекта платформа Яндекс.Диалоги была выбрана для развертывания разрабатываемого голосового помощника для анкетирования. Ее возможности позволяют быстро создавать и настраивать диалоговые системы, а также обеспечивают необходимую производительность и отказоустойчивость.

Интеграция с инструментальным средством для генерации кода спецификации упрощает процесс разработки, позволяя автоматизировать создание диалоговых сценариев на основе заданной анкеты.

# Структура информационной модели голосового помощника

* 1. Описание информационной модели

Информационная модель голосового помощника в формате РБНФ описывает структуру диалоговой системы, в которой пользовательский запрос обрабатывается и сопоставляется с соответствующим интентом, выраженной в конкретной реплике.

Каждый **интент** соответствует одной форме (фрейму), который содержит набор слотов, представляющих собой поля формы с названием, типом данных и признаком обязательности:

*<форма> ::= <интент>, {слот}*

**Слоты** — это переменные в форме, которые необходимо заполнить для выполнения интента. Каждый слот имеет название, тип данных (например, число, текст, дата) и признак обязательности

*<слот> ::= <название>, <тип данных>, <обязателен или нет>*

**Шаблоны фраз** используются для распознавания входных данных пользователя.

**Сущности и нетерминалы** используются для определения шаблонов фраз, которые могут быть сопоставлены с запросами пользователя:

*<шаблон фразы> ::= <сущность-шаблон> | <нетерминал-шаблон>*

**Сущности** представляют конкретные значения (например, "Москва", "завтра"), которые можно извлечь из запроса пользователя.

*<сущность-шаблон> ::= <сущность>, [ директивы ].*

*<нетерминал-шаблон> ::= <нетерминал >, [ директивы ].*

*<сущность> ::= <имя сущности>, { значение }, <параметры>.*

*<значение> ::= <имя значения>, { синоним }.*

*<нетерминал> ::= <имя нетерминала>, {синонимы}.*

*<имя нетерминала> ::= {символ алфавита}*

**Нетерминал (нетерминальный символ)** — объект, обозначающий какую-либо сущность языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения. Нетерминалы обозначаются символом $ и могут быть скрыты внутри родительского нетерминала.

**Сущности** представляют собой конкретные значения, которые могут быть сопоставлены с нетерминалами в запросе пользователя.

**Директивы** помогают указать, как следует интерпретировать нетерминалы и сущности, например, с учетом формы слова или точного совпадения.

*<директивы> ::= <ключевое слово>.*

*<ключевое слово>::=( <lemma>  | <exact> | <negative> ) | <оператор “[]”>*

**Директива** *-* руководящее указание

*%lemma – нетерминалы сравниваются без учета формы слова.*

*%exact – нетерминалы сравниваются по точному совпадению.*

*%negative – директива для указания отрицательных примеров для элемента. Отрицательные правила должны быть более конкретными, чем положительные.*

**Оператор []**– позволяет игнорировать порядок слов в грамматике.

**Квантификаторы** используются для указания количества вхождений нетерминалов и сущностей в запросе пользователя. Квантификаторы могут быть "?" (одно или ноль вхождений), "\*" (ноль или больше вхождений) и "+" (хотя бы одно вхождение):

*<квантификаторы> ::= <спец. символы>.*

*<спец. символы> ::=<?> | <\*> | <+>*

*? — одно или ноль вхождений;*

*\* — ноль или больше вхождений;*

*+ — хотя бы одно вхождение.*

Ответ системы может состоять из текста, звука, карточки, кнопок и инструкции. Текст представляет собой текст, который следует показать и озвучить пользователю. Звук представляет собой ответ в формате TTS (text-to-speech), который не отображается в виде текста, но озвучивается. Кнопки представляют собой кнопки, которые следует показать пользователю. Инструкция представляет собой возможность запуска аудиоплеера, начала процесса авторизации или запроса геолокации. Карточка представляет собой сообщение с поддержкой изображений. Изображение может быть типа BigImage (одно изображение), ItemList (список из нескольких изображений) или ImageGallery (галерея из нескольких изображений):

*<ответ> ::= ( [ текст], [ звук], [ карточка ], [ кнопки ], [ инструкция ] ), <end\_session>.*

*end\_session – Признак конца разговора.*

*<звук> ::= <текст>, <озвучка текста>*

*<кнопки> ::= <заголовок>, <скрыть>*

*<скрыть> = <да> | <нет>*

*<инструкция> ::= <audio\_player> | <start\_account\_linking> | <request\_geolocation>*

*Directives – Возможен запуск аудиоплеера, начала процесса авторизации, запроса геолокации*

*<карточка> ::= <изображение>, <тип изображения>, <заголовок>*

*<тип изображения> ::= <BigImage> | <ItemList> | <ImageGallery>*

Эта модель помогает разработчикам голосовых помощников структурировать обработку запросов, определять необходимые данные для выполнения интентов и формировать адекватные и мультимодальные ответы на запросы пользователей.

* 1. База данных на основе информационной модели

# База данных, содержащая информацию о голосовых помощниках, разработанных на платформах Яндекс.Диалоги и Сбер.Салют.

Эта база данных предназначена для хранения и управления информацией о голосовых помощниках, разработанных на платформах Яндекс.Диалоги и Сбер.Салют.

Схема базы данных, где отображены все связи представлена на рисунке 1.

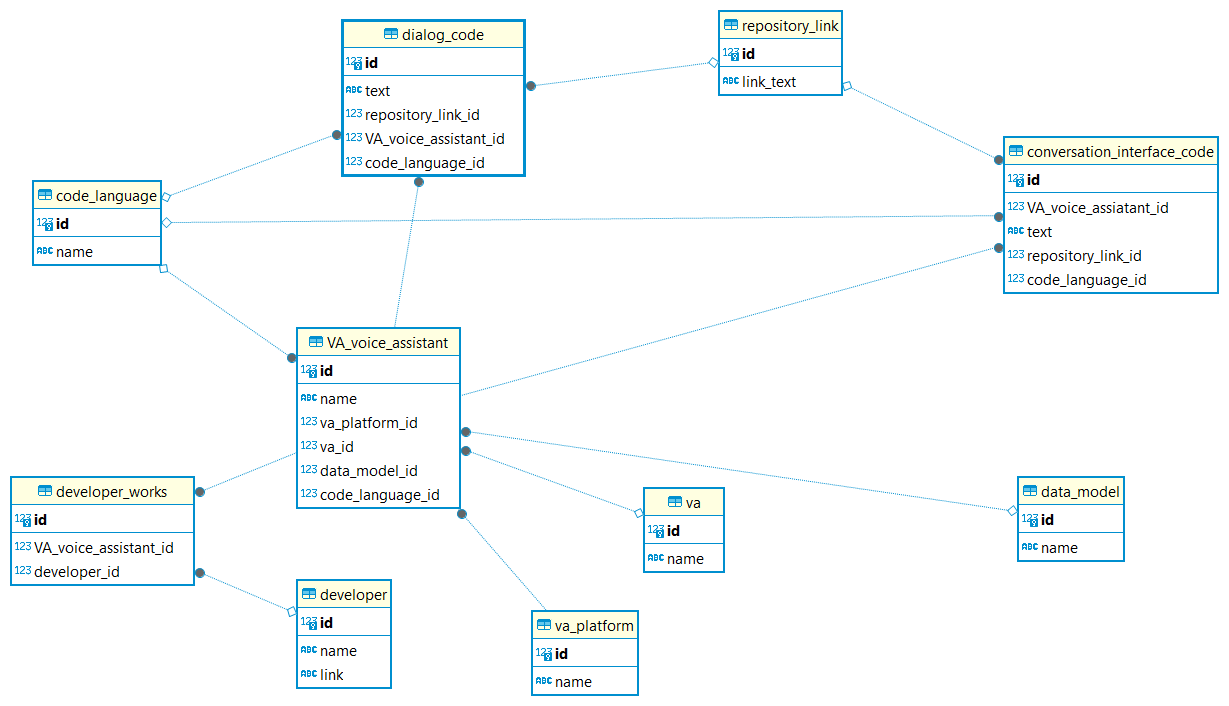


Рисунок 1 – База данных «ВА шаблоны»

Каждая таблица в базе данных специализируется на различных аспектах, связанных с голосовыми помощниками, включая информацию о разработчиках, языках программирования, платформах и моделях данных. Вот краткое описание основных таблиц:

1. code\_language: хранит информацию о языках программирования, используемых для разработки голосовых помощников. Каждый язык имеет уникальный идентификатор и наименование.
   * id: уникальный идентификатор языка программирования. Используется для связывания с другими таблицами.
   * name: название языка программирования, например, Python, JavaScript и т.д.
2. data\_model: содержит информацию о различных моделях данных, которые могут использоваться голосовыми помощниками. Это могут быть спецификации структуры данных, которые помощники используют для обработки запросов.
   * id: уникальный идентификатор модели данных.
   * name: название модели данных, например, JustAI DSL или YandexDialog.
3. developer: записывает данные о разработчиках, включая их имена и ссылки на профили на github.
   * id: уникальный идентификатор разработчика.
   * name: имя или никнейм разработчика.
   * link: ссылка на профиль разработчика на github.
4. repository\_link: содержит ссылки на репозитории кода, которые могут быть связаны с конкретными голосовыми помощниками или их компонентами.
   * id: уникальный идентификатор ссылки на репозиторий.
   * link\_text: ссылка на репозиторий кода.
5. va\_platform: хранит информацию о платформах, таких как Яндекс.Диалоги и Сбер.Салют, на которых разрабатываются голосовые помощники.
   * id: Уникальный идентификатор платформы.
   * name: Название платформы, например, Яндекс.Диалоги или Сбер.Салют.
6. conversation\_interface\_code, dialog\_code: Эти таблицы хранят код, который используется для разговорного интерфейсов и диалогов голосовых помощников. Они также связывают код с его репозиториями и языками программирования.
   * Обе таблицы содержат поля для хранения кода (текст) и связей с репозиториями и языками программирования.
   * conversation\_interface\_code фокусируется на коде разговорного интерфейса, в то время как dialog\_code хранит коды, связанные с кодом диалогов.
7. developer\_works: отслеживает, какие разработчики работали над какими голосовыми помощниками, обеспечивая связь между разработчиками и помощниками.
8. VA\_voice\_assistant: центральная таблица, которая интегрирует информацию из других таблиц, связывая голосовых помощников с их платформами, моделями данных и языками программирования.
   * id и другие поля связывают голосового помощника с его платформой, моделью данных и языком программирования через внешние ключи.
9. va: Таблица для хранения базовой информации о голосовых помощниках. Каждый помощник имеет уникальный идентификатор.
   * id: уникальный идентификатор голосового помощника.
   * name: название голосового помощника.

Эта структура базы данных позволяет управлять и анализировать информацию о разработке и функционировании голосовых помощников, обеспечивая доступ к детализированной информации о каждом аспекте разработки и их взаимосвязях.

# Описание архитектуры и функциональных возможностей разрабатываемого голосового помощника

Данный код предназначен для проведения анкетирования на платформе Яндекс Диалоги. Данное решение предоставляет удобный и интуитивно понятный способ сбора информации от пользователей через интерактивное общение.

Чат-бот обладает гибкой системой управления состояниями, которая позволяет настраивать последовательность вопросов и переходы между ними в зависимости от ответов пользователя. Благодаря этому можно создавать сложные и разветвленные сценарии анкетирования, адаптированные под конкретные потребности.

Бот поддерживает различные типы вопросов, включая вопросы с вариантами ответов (закрытые вопросы), вопросы с вводом текста и числовых значений (открытые вопросы), а также условные вопросы, которые задаются в зависимости от предыдущих ответов пользователя. Это позволяет собирать разнообразную информацию и получать точные и релевантные данные.

Код обеспечивает обработку ответов пользователя, распознавая интенты и извлекая необходимые значения из текста. Бот умеет обрабатывать специальные команды, такие как пропуск вопроса или досрочное завершение анкетирования, предоставляя пользователям гибкость и удобство в прохождении опроса.

Чат-бот не только собирает данные, но и предоставляет удобные возможности для их сохранения и дальнейшей обработки. Результаты анкетирования автоматически загружаются на Яндекс.Диск в формате JSON, что позволяет легко получить доступ к собранной информации. Кроме того, данные отправляются на указанный URL с помощью POST-запроса, что дает возможность интеграции с внутренними системами и сервисами.

Код чистый, структурированный и легко поддерживаемый. Каждая функция имеет четкое назначение и документацию, что упрощает понимание и модификацию кода при необходимости.

Данное решение станет незаменимым инструментом для проведения опросов, сбора обратной связи, анкетирования клиентов и многих других задач, где требуется получение структурированной информации от пользователей.

* 1. Архитектура.
  + Код построен по принципу конечного автомата (Finite State Machine), где каждый вопрос представляет собой состояние (state), и переходы между состояниями определяются на основе ответов пользователя и условий перехода.
  + Данные о структуре опроса (вопросы, следующие состояния, условия перехода) хранятся в словарях questions и next\_states, которые могут быть загружены из файла JSON или определены непосредственно в коде.
  + Функция handler является точкой входа и обрабатывает входящие события от Яндекс Диалогов. Она определяет текущее состояние, вызывает соответствующие функции для обработки ответа пользователя и возвращает ответ.
  + Функции process\_\* обрабатывают различные типы вопросов и обновляют состояние в зависимости от ответа пользователя.
  + Функции upload\_json\_to\_yandex\_disk и upload\_post отвечают за сохранение результатов анкетирования на Яндекс.Диск и отправку данных на указанный URL.
  1. Функциональные возможности.

**Сначала происходит импорт необходимых библиотек:**

* + json: для работы с данными в формате JSON.
  + requests: для отправки HTTP-запросов.
  + upload: модуль, содержащий функции для загрузки данных на Яндекс.Диск и отправки POST-запросов.
  + random: для генерации случайных значений.

После этого определяются константы STATE\_REQUEST\_KEY и STATE\_RESPONSE\_KEY для работы с состояниями в Яндекс Диалогах.

После этого идут функции с определёнными ролями, подробнее рассмотрим каждую функцию и ее роль в коде:

**Функции для работы с Яндекс Диалогами:**

* + make\_response(text, buttons=None, state=None): создает словарь ответа для отправки пользователю, включая текст, кнопки (если указаны) и состояние (если указано).
  + get\_intents(event): извлекает интенты из входящего события Яндекс Диалогов.
  + is\_new\_session(event): проверяет, является ли сессия новой, на основе информации из входящего события.
  + is\_not\_new(state): проверяет, является ли состояние не новым, т.е. содержит ли оно ключ 'screen'.
  + get\_state(event): извлекает состояние из входящего события или инициализирует новое состояние, если оно отсутствует.

**Функции для работы с вопросами и состояниями:**

* + load\_questions\_from\_url(url) и load\_next\_states\_from\_url(url): загружают вопросы и информацию о следующих состояниях из указанного URL и собирают их в нужном формате (эти функции находятся в разработке и пока не используются).
  + load\_questions() и load\_next\_states(): загружают вопросы и информацию о следующих состояниях из предопределенных словарей.
  + get\_current\_state(state): извлекает текущее состояние (идентификатор текущего вопроса) из словаря состояний.
  + get\_next\_state(cur\_state): получает следующее состояние для текущего вопроса из словаря next\_states.
  + compare(actual, expected, operator): сравнивает фактическое значение с ожидаемым значением, используя указанный оператор сравнения.
  + get\_conditional\_next\_state(cur\_state, state): определяет следующее состояние на основе условий перехода для текущего вопроса и текущего состояния.
  + get\_question\_type(cur\_state), get\_expected\_intent(cur\_state), get\_slots\_to\_fill(cur\_state), get\_possible\_values(cur\_state): получают соответствующую информацию о текущем вопросе из словаря next\_states.
  + if\_stop(intents), if\_skip(intents), `if\_was\_next(cur\_state, intents)`: проверяют наличие определенных интентов в ответе пользователя.
  + get\_value(cur\_state, intents): извлекает значения слотов из ответа пользователя для текущего вопроса.
  + get\_question\_text(cur\_state, state): получает текст вопроса для текущего состояния, учитывая флаги 'replay' и 'no\_skip'.
  + initialize\_state(): инициализирует новое состояние анкетирования.

**Функции для обработки ответов пользователя:**

* + process\_stop\_intent(state, cur\_state): обрабатывает намерение остановки, устанавливая состояние 'not\_end' и переходя к состоянию 'End'.
  + process\_skip\_intent(state, cur\_state): обрабатывает намерение пропуска, переходя к следующему вопросу или устанавливая флаг 'no\_skip', если текущий вопрос нельзя пропустить.
  + process\_check\_question(state, cur\_state, intents): обрабатывает ответ на вопрос типа 'check', сохраняя значения слотов и переходя к следующему вопросу, если ответ соответствует ожидаемому интенту, или устанавливая флаг 'replay', если ответ некорректный.
  + process\_option\_question(state, cur\_state, intents): обрабатывает ответ на вопрос типа 'option', сохраняя значения слотов и переходя к следующему вопросу, если ответ соответствует ожидаемому интенту и возможным значениям, или устанавливая флаг 'replay', если ответ некорректный.
  + process\_conditional\_question(state, cur\_state, intents): обрабатывает ответ на условный вопрос, определяя следующее состояние на основе условий перехода.
  + validate\_answer(state, intents, cur\_state): проверяет и обрабатывает ответ пользователя в зависимости от типа текущего вопроса, вызывая соответствующую функцию process\_\*.

**Основные функции:**

* + first\_question(state): определяет первый вопрос анкетирования и устанавливает начальное состояние.
  + ask\_question(state): задает текущий вопрос пользователю в зависимости от текущего состояния, или завершает анкетирование, если достигнуто состояние 'End'.
  + handler(event, context): основная функция-обработчик, которая получает входящее событие и контекст от Яндекс Диалогов, определяет текущее состояние, вызывает соответствующие функции для обработки ответа пользователя и возвращает ответ.

**Функции выгрузки данных:**

* + upload\_json\_to\_yandex\_disk(state): загружает результаты анкетирования в формате JSON на Яндекс.Диск, используя API Яндекс.Диска.
  + upload\_post(state): отправляет результаты анкетирования на указанный URL с помощью POST-запроса.
  1. Структура словарей

Давайте более подробно рассмотрим структуру словарей, так как от неё зависит правильная работа остального кода.

Структура словарей:

1. Словарь questions:
   * Ключи: идентификаторы вопросов в формате 'ask\_<question\_id>'. (этот формат используется в варианте чтения данных из БД, в варианте предустановленного словаря используются свободные названия, главное, чтобы была правильная структура)
   * Значения: словари, содержащие текст вопроса в виде списка строк, например: {'text': ['Вопрос 1', 'Вопрос 2']}.
2. Словарь next\_states:
   * Ключи: идентификаторы вопросов в формате 'ask\_<question\_id>'.
   * Значения: словари, описывающие следующее состояние для каждого вопроса, включая:
     + 'next\_state': идентификатор следующего вопроса или словарь с условиями перехода.
     + 'question\_type': тип вопроса ('check', 'option' или 'conditional').
     + 'expected\_intent': ожидаемый интент для данного вопроса.
     + 'slot\_to\_fill' или 'slots\_to\_fill': слот или список слотов, которые необходимо заполнить.
     + 'possible\_values': список возможных значений для вопросов типа 'option'.
3. Словарь state:
   * Ключи: различные параметры состояния анкетирования.
   * Значения: соответствующие значения параметров.
   * Важные ключи:
     + 'screen': идентификатор текущего вопроса.
     + 'no\_skip': флаг, указывающий, что текущий вопрос нельзя пропустить.
     + 'not\_end': идентификатор вопроса, на котором было прервано анкетирование (используется при возобновлении).
     + 'replay': флаг, указывающий, что нужно повторить текущий вопрос.
   1. Реализация функции перехода между состояниями

Теперь более подробно рассмотрим способы переходов между состояниями:

1. Начало анкетирования:
   * При начале новой сессии вызывается функция first\_question(state), которая определяет первый вопрос и устанавливает состояние 'screen' в значение 'welcome\_message' или первый вопрос, если приветственное сообщение отсутствует.
2. Переход к следующему вопросу:
   * После обработки ответа пользователя функции process\_\* определяют следующее состояние на основе текущего вопроса и ответа пользователя.
   * Если следующее состояние является идентификатором вопроса, состояние 'screen' обновляется на этот идентификатор.
   * Если следующее состояние является словарем с условиями перехода, выполняется функция get\_conditional\_next\_state(cur\_state, state), которая определяет следующий вопрос на основе условий и обновляет состояние 'screen' соответствующим образом.
3. Обработка специальных намерений:
   * Если пользователь выражает намерение остановки ('stop'), вызывается функция process\_stop\_intent(state, cur\_state), которая устанавливает состояние 'not\_end' в текущий вопрос и переходит к состоянию 'End'.
   * Если пользователь выражает намерение пропуска ('skip'), вызывается функция process\_skip\_intent(state, cur\_state), которая либо переходит к следующему вопросу, либо устанавливает флаг 'no\_skip', если текущий вопрос нельзя пропустить.
4. Обработка ответов пользователя:
   * Функции process\_\* проверяют, соответствует ли ответ пользователя ожидаемому интенту для текущего вопроса.
   * Если ответ соответствует ожидаемому интенту, функции извлекают значения слотов из ответа и сохраняют их в состоянии state для текущего вопроса.
   * Если ответ не соответствует ожидаемому интенту, устанавливается флаг 'replay', указывающий на необходимость повторить текущий вопрос.
5. Завершение анкетирования:
   * Когда достигнуто состояние 'End', вызываются функции upload\_json\_to\_yandex\_disk(state) и upload\_post(state) для сохранения результатов анкетирования на Яндекс.Диск и отправки данных на указанный URL.
   1. Листинг программы (в приложении)

Основной код: что делать с оформлением

import json

import requests

from upload import upload\_json\_to\_yandex\_disk, upload\_post

import logging

import random

# Настройка базового конфига для логирования

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

STATE\_REQUEST\_KEY = 'session'

STATE\_RESPONSE\_KEY = 'session\_state'

# Функции для работы с Яндекс Диалогами

def make\_response(text, buttons=None, state=None):

    response = {'text': text}

    if buttons:

        response['buttons'] = buttons

    webhook\_response = {'response': response, 'version': '1.0'}

    if state is not None:

        webhook\_response[STATE\_RESPONSE\_KEY] = state

    return webhook\_response

def get\_intents(event):

    return event['request']['nlu'].get('intents', {})

def is\_new\_session(event):

    return event['session']['new']

def is\_not\_new(state):

    if 'screen' in state:

        return True

    else:

        return False

def get\_state(event):

    return event.get('state', {}).get(STATE\_REQUEST\_KEY, initialize\_state())

# Функции для работы с вопросами и состояниями

def load\_questions\_from\_url(url):

    response = requests.get(url)

    data = response.json()

    questions = {}

    for item in data:

        question\_key = f"ask\_{item['question\_id']}"

        questions[question\_key] = {

            'text': [item['question\_name']]

        }

    print(questions)

    return questions

def load\_next\_states\_from\_url(url):

    response = requests.get(url)

    data = response.json()

    next\_states = {}

    sorted\_data = sorted(data, key=lambda x: x['priority'])

    # Создаем словарь next\_states со всеми вопросами

    for item in sorted\_data:

        question\_key = f"ask\_{item['question\_id']}"

        next\_states[question\_key] = {

            'next\_state': None,

            'question\_type': None,

            'expected\_intent': None,

            'slot\_to\_fill': None,

            'possible\_values': [option['option\_text'] for option in item['question\_options']]

        }

    # Обрабатываем условия и устанавливаем следующие состояния

    for i, item in enumerate(sorted\_data):

        question\_key = f"ask\_{item['question\_id']}"

        if item['transition\_type'] == 1:

            next\_states[question\_key]['question\_type'] = 'option'

            if i < len(sorted\_data) - 1:

                next\_question\_key = f"ask\_{sorted\_data[i+1]['question\_id']}"

                next\_states[question\_key]['next\_state'] = next\_question\_key

        elif item['transition\_type'] == 2:

            next\_states[question\_key]['question\_type'] = 'conditional'

            next\_states[question\_key]['conditions'] = []

            next\_states[question\_key]['next\_state'] = {}

            for condition in item['question\_conditions']:

                condition\_question\_key = f"ask\_{condition['operand\_question\_id']}"

                if condition\_question\_key in next\_states:

                    condition\_value = next\_states[condition\_question\_key]['possible\_values'][condition['q\_options'][0]['option\_id'] - 1]

                    next\_states[question\_key]['conditions'].append([

                        {'question': condition\_question\_key, 'value': condition\_value, 'operator': '=='}

                    ])

                    next\_question\_key = f"ask\_{sorted\_data[i+1]['question\_id']}"

                    next\_states[question\_key]['next\_state']['condition1'] = next\_question\_key

            if i < len(sorted\_data) - 1:

                next\_question\_key = f"ask\_{sorted\_data[i+1]['question\_id']}"

                next\_states[question\_key]['next\_state']['default'] = next\_question\_key

            else:

                next\_states[question\_key]['next\_state']['default'] = 'End'

    print(next\_states)

    return next\_states

def load\_questions():

    return {

        'welcome\_message': {'text': 'Здравствуйте, давайте познакомимся. Как к Вам лучше обращаться?'},

        'ask\_sex': {'text': ['Какой пол мне указать в анкете? М / Ж', 'Укажите, пожалуйста, ваш пол', 'Вы мужчина или женщина?']},

        'ask\_age': {'text': ['Сколько Вам лет?', 'Укажите, пожалуйста, ваш возраст', 'Можете назвать свой возраст?']},

        'ask\_height\_weight': {'text': ['Пожалуйста, назовите свой рост в сантиметрах и вес [Например, 170 см, 60 кг]',]},

        'ask\_stud': {'text': ['Вы студент? Какая у вас температура?',]},

        'ask\_dom': {'text': ['Подскажите пожалуйста где вы проживаете в настоящее время: в общежитии, снимаете квартиру или с родителями?',]},

        'ask\_smoke': {'text': ['Курите ли Вы в настоящее время?',]},

        'ask\_tip': {'text': ['Вы курите сигареты или используете альтернативные системы нагревания табака?',]},

        'ask\_past': {'text': ['А раньше курили?',]},

        'ask\_cigarettes\_count': {'text': ['Сколько сигарет или стиков в день Вы выкуриваете?',]},

        'ask\_smoking\_experience': {'text': ['Вы курите больше года/ровно год или меньше года?',]},

        'ask\_smoking\_years': {'text': ['Сколько лет вы курите?',]},

        'ask\_conditional\_question': {'text': ['Да ладно, у меня запустился этот вопрос?',]},

        'ask\_question1': {'text': ['Это вопрос для мужчин',]},

        'ask\_question2': {'text': ['Это вопрос для девочек',]},

    }

def load\_next\_states():

    return {

        'welcome\_message': {

            'next\_state': 'ask\_sex',

            'question\_type': 'check',

            'expected\_intent': 'fio',

            'slot\_to\_fill': 'slot\_fio'

        },

        'ask\_sex': {

            'next\_state': 'ask\_age',

            'question\_type': 'option',

            'expected\_intent': 'sex',

            'slot\_to\_fill': 'var\_sex',

            'possible\_values': ['male', 'female']

        },

        'ask\_age': {

            'next\_state': 'ask\_height\_weight',

            'question\_type': 'check',

            'expected\_intent': 'num',

            'slot\_to\_fill': 'slot\_num'

        },

        'ask\_height\_weight': {

            'next\_state': 'ask\_stud',

            'question\_type': 'check',

            'expected\_intent': 'height\_weight',

            'slots\_to\_fill': ['slot\_height', 'slot\_weight']

        },

        'ask\_stud': {

            'next\_state': 'ask\_dom',

            'question\_type': 'option',

            'expected\_intent': 'stud',

            'slots\_to\_fill': ['var\_stud', 'slot\_num'],

            'possible\_values': ['stud', 'no\_stud']

        },

        'ask\_dom': {

            'next\_state': {

                'ob': 'ask\_smoke',

                'kvar': 'ask\_past',

                'rod': 'ask\_smoking\_experience'

            },

            'question\_type': 'option',

            'expected\_intent': 'choose\_dom',

            'slots\_to\_fill': ['var\_dom', 'slot\_time'],

            'possible\_values': ['ob', 'kvar', 'rod']

        },

        'ask\_smoke': {

            'next\_state': 'ask\_tip',

            'question\_type': 'option',

            'expected\_intent': 'smoke',

            'slot\_to\_fill': 'var\_smoke',

            'possible\_values': ['yes\_smoke', 'no\_smoke']

        },

        'ask\_tip': {

            'next\_state': 'ask\_cigarettes\_count',

            'question\_type': 'option',

            'expected\_intent': 'kyr',

            'slot\_to\_fill': 'var\_kyr',

            'possible\_values': ['cigarettes', 'vape', 'icos']

        },

        'ask\_past': {

            'next\_state': 'ask\_cigarettes\_count',

            'question\_type': 'option',

            'expected\_intent': 'past',

            'slot\_to\_fill': 'var\_past',

            'possible\_values': ['yes\_past', 'no\_past']

        },

        'ask\_cigarettes\_count': {

            'next\_state': 'ask\_smoking\_experience',

            'question\_type': 'check',

            'expected\_intent': 'num',

            'slot\_to\_fill': 'slot\_num'

        },

        'ask\_smoking\_experience': {

            'next\_state': 'ask\_smoking\_years',

            'question\_type': 'option',

            'expected\_intent': 'exp',

            'slot\_to\_fill': 'var\_exp',

            'possible\_values': ['more\_year', 'less\_year']

        },

        'ask\_smoking\_years': {

            'next\_state': 'ask\_conditional\_question',

            'question\_type': 'check',

            'expected\_intent': 'num',

            'slot\_to\_fill': 'slot\_num'

        },

        'ask\_conditional\_question': {

            'next\_state': {

                'condition1': 'ask\_question1',

                'condition2': 'ask\_question2',

                'default': 'End'

            },

            'question\_type': 'conditional',

            'conditions': [

                [

                    {'question': 'ask\_sex', 'value': 'male', 'operator': '=='},

                    {'question': 'ask\_age', 'value': '18', 'operator': '>='}

                ],

                [

                    {'question': 'ask\_sex', 'value': 'female', 'operator': '=='},

                    {'question': 'ask\_age', 'value': '18', 'operator': '<'}

                ]

            ]

        },

        'ask\_question1': {

            'next\_state': 'End',

            'question\_type': 'check',

            'expected\_intent': 'num',

            'slot\_to\_fill': 'slot\_num'

        },

        'ask\_question2': {

            'next\_state': 'End',

            'question\_type': 'check',

            'expected\_intent': 'num',

            'slot\_to\_fill': 'slot\_num'

        }

    }

# Этот вариант загрузки вопросов ещё в доработке

# url = "http://51.250.4.123:5005/getInterviewStructure?id=1"

# questions = load\_questions\_from\_url(url)

# next\_states = load\_next\_states\_from\_url(url)

# Вариант используемый сейчас

questions = load\_questions()

next\_states = load\_next\_states()

def get\_current\_state(state):

    return state.get('screen')

def get\_next\_state(cur\_state):

    return next\_states[cur\_state]['next\_state']

def compare(actual, expected, operator):

    if operator in ['>=', '<']:  # Если оператор предполагает числовое сравнение

        try:

            actual = int(actual)

            expected = int(expected)

        except ValueError:

            print(f"Error converting {actual} or {expected} to integers.")

            return False

    if operator == '==':

        return actual == expected

    elif operator == '>=':

        return actual >= expected

    elif operator == '<':

        return actual < expected

    else:

        print(f"Unknown operator {operator}.")

        return False

def get\_conditional\_next\_state(cur\_state, state):

    conditions = next\_states[cur\_state]['conditions']

    next\_states\_dict = next\_states[cur\_state]['next\_state']

    for i, condition\_group in enumerate(conditions, start=1):

        print(f"Checking condition group {i}: {condition\_group}")

        all\_conditions\_passed = True

        for condition in condition\_group:

            question = condition['question']

            key\_to\_use = next\_states[question].get('slot\_to\_fill', '') or next\_states[question].get('slots\_to\_fill')

            if isinstance(key\_to\_use, tuple):

                key\_to\_use = key\_to\_use[0]  # Если есть несколько ключей, используем первый для простоты

            actual = state.get(question, {}).get(key\_to\_use, '')

            expected = condition['value']

            operator = condition['operator']

            print(f"Comparing {actual} {operator} {expected}")

            if not compare(actual, expected, operator):

                all\_conditions\_passed = False

                print(f"Failed condition: {actual} {operator} {expected}")

                break

        if all\_conditions\_passed:

            print(f"Condition {i} passed, moving to {next\_states\_dict[f'condition{i}']}")

            return next\_states\_dict[f'condition{i}']

    print("No conditions passed, moving to default")

    return next\_states\_dict['default']

def get\_question\_type(cur\_state):

    return next\_states[cur\_state]['question\_type']

def get\_expected\_intent(cur\_state):

    return next\_states[cur\_state]['expected\_intent']

def get\_slots\_to\_fill(cur\_state):

    return next\_states[cur\_state].get('slots\_to\_fill', next\_states[cur\_state].get('slot\_to\_fill'))

def get\_possible\_values(cur\_state):

    return next\_states[cur\_state].get('possible\_values')

def if\_stop(intents):

    return 'stop' in intents

def if\_skip(intents):

    return 'skip' in intents

def if\_was\_next(cur\_state, intents):

    expected\_intent = get\_expected\_intent(cur\_state)

    return expected\_intent in intents

def get\_value(cur\_state, intents):

    expected\_intent = get\_expected\_intent(cur\_state)

    slots\_to\_fill = get\_slots\_to\_fill(cur\_state)

    values = {}

    if isinstance(slots\_to\_fill, str):

        values[slots\_to\_fill] = intents[expected\_intent]['slots'].get(slots\_to\_fill, {}).get('value')

    else:

        for slot in slots\_to\_fill:

            values[slot] = intents[expected\_intent]['slots'].get(slot, {}).get('value')

    possible\_values = get\_possible\_values(cur\_state)

    return values, possible\_values

def get\_question\_text(cur\_state, state):

    if state.get('replay', False):

        text = 'Извините, не могли бы вы более точно ответить на вопрос:\n' + random.choice(questions[cur\_state]['text'])

        state['replay'] = False

    elif state.get('no\_skip', False):

        text = 'Извините, но этот вопрос нельзя пропустить, так как он влияет на следующий. Пожалуйста, ответьте на вопрос:\n' + random.choice(questions[cur\_state]['text'])

        state['no\_skip'] = False

    else:

        text = random.choice(questions[cur\_state]['text'])

    return text

def initialize\_state():

    return {'screen': None, 'not\_end': None, 'no\_skip': False, 'replay': False}

# Функции для обработки ответов пользователя

def process\_stop\_intent(state, cur\_state):

    state['not\_end'] = cur\_state

    state['screen'] = 'End'

    return ask\_question(state)

def process\_skip\_intent(state, cur\_state):

    if next\_states.get(cur\_state):

        next\_state = get\_next\_state(cur\_state)

        if isinstance(next\_state, dict):

            state['no\_skip'] = True

            state['screen'] = cur\_state

            return ask\_question(state)

        else:

            state[cur\_state] = None

            state['screen'] = next\_state

            return ask\_question(state)

    else:

        state['no\_skip'] = True

        state['screen'] = cur\_state

        return ask\_question(state)

def process\_check\_question(state, cur\_state, intents):

    next\_state = get\_next\_state(cur\_state)

    if if\_was\_next(cur\_state, intents):

        values, \_ = get\_value(cur\_state, intents)

        if cur\_state not in state:

            state[cur\_state] = {}

        for slot, value in values.items():

            state[cur\_state][slot] = value

        state['screen'] = next\_state

        return ask\_question(state)

    else:

        state['replay'] = True

        state['screen'] = cur\_state

        return ask\_question(state)

def process\_option\_question(state, cur\_state, intents):

    if if\_was\_next(cur\_state, intents):

        values, possible\_values = get\_value(cur\_state, intents)

        slots\_to\_fill = get\_slots\_to\_fill(cur\_state)

        if cur\_state not in state:

            state[cur\_state] = {}

        if isinstance(slots\_to\_fill, str):

            value = list(values.values())[0]

            if possible\_values is not None:

                if value in possible\_values:

                    state[cur\_state][slots\_to\_fill] = value

                    if isinstance(get\_next\_state(cur\_state), dict):

                        next\_state = get\_next\_state(cur\_state)[value]

                    else:

                        next\_state = get\_next\_state(cur\_state)

                    state['screen'] = next\_state

                    return ask\_question(state)

            else:

                state[cur\_state][slots\_to\_fill] = value

                next\_state = get\_next\_state(cur\_state)

                state['screen'] = next\_state

                return ask\_question(state)

        else:

            for slot in slots\_to\_fill:

                if slot in values:

                    state[cur\_state][slot] = values[slot]

            next\_state = get\_next\_state(cur\_state)

            if isinstance(next\_state, dict):

                selected\_value = state[cur\_state][slots\_to\_fill[0]]

                state['screen'] = next\_state[selected\_value]

            else:

                state['screen'] = next\_state

            return ask\_question(state)

    else:

        state['replay'] = True

        state['screen'] = cur\_state

        return ask\_question(state)

def process\_conditional\_question(state, cur\_state, intents):

    next\_state = get\_conditional\_next\_state(cur\_state, state)

    state['screen'] = next\_state

    return ask\_question(state)

def validate\_answer(state, intents, cur\_state):

    question\_type = get\_question\_type(cur\_state)

    if if\_stop(intents):

        return process\_stop\_intent(state, cur\_state)

    elif if\_skip(intents):

        return process\_skip\_intent(state, cur\_state)

    elif question\_type == 'check':

        return process\_check\_question(state, cur\_state, intents)

    elif question\_type == 'option':

        return process\_option\_question(state, cur\_state, intents)

    elif question\_type == 'conditional':

        return process\_conditional\_question(state, cur\_state, intents)

    else:

        return make\_response('Некорректный ответ. Пожалуйста, попробуйте еще раз.', state=state)

# Основные функции

def first\_question(state):

    if 'welcome\_message' in questions:

        return make\_response(questions['welcome\_message']['text'], state={'screen': 'welcome\_message'} if not state.get('screen') else state)

    else:

        return make\_response('Здравствуйте! Давайте начнем опрос.', state={'screen': 'ask\_15'} if not state.get('screen') else state)

def ask\_question(state):

    cur\_state = get\_current\_state(state)

    if cur\_state == 'End':

        upload\_json\_to\_yandex\_disk(state)

        upload\_post(state)

        return make\_response('Спасибо, анкетирование окончено, файлы сохранены', state=state)

    else:

        if cur\_state in questions:

            text = get\_question\_text(cur\_state, state)

            return make\_response(text, state=state)

        else:

            return make\_response('Вопрос не найден.', state=state)

def handler(event, context):

    state = get\_state(event)

    intents = get\_intents(event)

    if is\_new\_session(event):

        return first\_question(state)

    elif is\_not\_new(state):

        cur\_state = get\_current\_state(state)

        return validate\_answer(state, intents, cur\_state)

    elif next\_states[state.get('screen')] == 'End':

        return make\_response('Спасибо за прохождение анкетирования!', state=state)

Функции для выгрузки полученных данных (находятся в файле upload.py):

import json

import requests

def upload\_json\_to\_yandex\_disk(state):

    json\_data = {'results': state}

    file\_name = 'data.json'

    yandex\_disk\_token = 'y0\_AgAAAABNWZOrAAtrUQAAAAD9wo8DAACEP7UX46JHGKIS8dDwccA7sDbE7A'

    upload\_url = 'https://cloud-api.yandex.net/v1/disk/resources/upload'

    headers = {'Authorization': f'OAuth {yandex\_disk\_token}'}

    params = {'path': file\_name, 'overwrite': 'true'}

    files = {'file': json.dumps(json\_data, ensure\_ascii=False)}

    response = requests.get(upload\_url, headers=headers, params=params)

    upload\_data = response.json()  # Получаем данные для загрузки файла

    upload\_url = upload\_data['href']  # Получаем URL для загрузки файла

    response = requests.put(upload\_url, files=files)

    if response.status\_code == 201:

        print("JSON успешно загружен на Яндекс Диск.")

    else:

        print("Ошибка при загрузке JSON на Яндекс Диск:", response.text)

def upload\_post(state):

    url = 'https://functions.yandexcloud.net/d4ei2d94t39iaompfhs8'

    headers = {'Content-Type': 'application/json'}

    try:

        data\_json = json.dumps(state, ensure\_ascii=False).encode('utf-8')

        print("Отправляемые данные:", data\_json)

        response = requests.post(url, data=data\_json, headers=headers)

        response.raise\_for\_status()  # Проверяем, был ли успешным запрос

        print("Ответ сервера:", response.text)

        return {

            "statusCode": response.status\_code,

            "body": response.text

        }

    except requests.exceptions.RequestException as e:

        print("Ошибка при отправке POST запроса:", str(e))

        return {

            "statusCode": 500,

            "body": f"Error: {str(e)}"

        }

Функция, которая приводит ответ в нужный вид для базы данных:

def format\_answers(state):

answers = []

for question\_key, answer\_data in state.items():

if question\_key in next\_states:

question\_of\_interview\_id = next\_states[question\_key]['question\_of\_interview\_id']

question\_id = next\_states[question\_key]['question\_id']

if answer\_data is not None:

if isinstance(answer\_data, dict):

option\_id = None

answer\_text = None

for key, value\_data in answer\_data.items():

if 'possible\_values' in next\_states[question\_key] and value\_data['value'] in [val['value'] for val in next\_states[question\_key]['possible\_values']]:

option\_id = [val['option\_id'] for val in next\_states[question\_key]['possible\_values'] if val['value'] == value\_data['value']]

else:

if value\_data['value'] is not None:

if answer\_text is None:

answer\_text = [value\_data['value']]

else:

answer\_text.append(value\_data['value'])

if option\_id:

answers.append({

"question\_of\_interview\_id": question\_of\_interview\_id,

"question\_id": question\_id,

"option\_id": option\_id,

"answer\_text": answer\_text,

"special\_answer\_type": None

})

else:

answers.append({

"question\_of\_interview\_id": question\_of\_interview\_id,

"question\_id": question\_id,

"option\_id": None,

"answer\_text": answer\_text,

"special\_answer\_type": None

})

else:

answers.append({

"question\_of\_interview\_id": question\_of\_interview\_id,

"question\_id": question\_id,

"option\_id": None,

"answer\_text": None,

"special\_answer\_type": None

})

else:

answers.append({

"question\_of\_interview\_id": question\_of\_interview\_id,

"question\_id": question\_id,

"option\_id": None,

"answer\_text": None,

"special\_answer\_type": 3

})

return answers

# Описание архитектуры и функциональных возможностей разрабатываемого генератора

# Описание архитектуры и функциональных возможностей разрабатываемого конвертера

# Тестирование

# Руководство пользователя

* 1. Пользователи системы

В рамках разрабатываемой системы можно выделить два типа пользователей:

* + Респондент – пользователь, который проходит анкетирование с помощью голосового помощника. Данный пользователь не является основным в контексте разработки системы.
  + Разработчик голосового помощника – основной пользователь, который использует инструментальное средство для создания спецификации голосового помощника для анкетирования на платформе Яндекс.Диалогов с использованием языка программирования Python.
  1. Руководство пользователя респондента

Для использования разработанного инструментального средства разработчику голосового помощника необходимо выполнить следующую последовательность действий:

1. Создать анкету для опроса респондентов на платформе создания ВА.
2. Зарегистрироваться в системе Яндекс.Диалогов.
3. Выгрузить нужное интервью, в результате чего будет сгенерирован код для голосового помощника.
4. Создать облачную функцию на платформе Яндекса, переписать сгенерированный код и настроить точки доступа.
5. Настроить спецификации разговорного интерфейса для голосового помощника.
6. Выгрузить код спецификации.
7. Ввести сгенерированную информацию в соответствующие поля системы Яндекса.

Таким образом, разработанное инструментальное средство позволяет автоматизировать процесс создания голосового помощника для анкетирования, сокращая время и усилия, необходимые для его разработки.

* 1. Руководство пользователя разработчика голосового помощника

# Заключение

В заключении данного курсового проекта были рассмотрены возможности использования голосовых интерфейсов в сфере опросов и анкетирования. Был разработан прототип системы голосового анкетирования на базе платформы "Яндекс-Диалоги", информационная модель, база данных и конвертер.

В ходе работы были изучены существующие решения в области голосового анкетирования, определены их основные преимущества и недостатки. Была разработана информационная модель ГП, база данных, генератор спецификации и конвертер, которые обеспечивают функциональность голосового анкетирования.

Тестирование прототипа показало его работоспособность и подтвердило преимущества использования голосовых технологий в данной сфере, такие как упрощение процесса анкетирования.

Однако, в ходе работы также были выявлены потенциальные проблемы и ограничения. Написать про проблемы.

В целом, разработанный прототип является перспективным решением для сферы опросов и анкетирования, но требует дальнейшей доработки и усовершенствования для обеспечения максимальной удобства и эффективности использования. Для этого предлагается провести дополнительные исследования и определить наилучшие практики в области голосовых интерфейсов.

# Список использованных источников