

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
филиал «Минский радиотехнический колледж»

Допущен к защите  
Преподаватель  
\_\_\_\_\_/А.А. Широкая/

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ «ГЕНЕРАТОР ДИАЛОГОВ С ИИ С  
ИСПОЛЬЗОВАНИЕМ .NET MAUI И AI»**

Пояснительная записка  
курсового проекта по предмету  
«Конструирование программ и языки программирования»  
МРК КП 2-400101 34 108ПЗ

Выполнил учащийся группы 2К9394  
\_\_\_\_\_/К.Г. Пасеко/

## СОДЕРЖАНИЕ

Введение.....	3
1 Постановка задачи .....	5
1.1 Описание предметной области.....	5
1.2 Обзор существующих аналогов .....	6
1.3 Информационная база задачи .....	10
1.4 Функциональное назначение.....	11
2 Проектирование задачи .....	17
2.1 Алгоритм решения задачи .....	17
2.2 Логическое моделирование .....	19
2.3 Описание инструментов разработки.....	23
3 Программная реализация .....	27
3.1 Физическая структура.....	27
3.2 Описание разработанных модулей .....	28
4 Тестирование.....	34
5 Применение.....	40
5.1 Назначение и условия применения .....	40
5.2 Руководство пользователя.....	40
Заключение .....	42
Список использованных источников.....	44
Приложение А (обязательное) Текст программы .....	45
Приложение Б (обязательное) Результат работы мобильного приложения .....	55

					МРК КП 2-400101 34 108ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.	Пасеко				Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» Пояснительная записка	Лит.	Лист	Листов
Провер.	Широкая						2	61
Реценз.						МРК		
Н. Контр.								
Утверд.								

## ВВЕДЕНИЕ

В современном мире, где технологии стремительно развиваются, а взаимодействие между людьми и машинами становится всё более сложным, возникает необходимость в создании инструментов, способствующих эффективному общению. Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» предлагает пользователям уникальную возможность взаимодействовать с искусственным интеллектом (ИИ), создавая реалистичные и контекстуально обоснованные диалоги. Это особенно актуально в условиях, когда грамотное использование ИИ может значительно повысить продуктивность и улучшить качество общения.

В условиях повседневной жизни, насыщенной информацией, пользователю требуется эффективный инструмент, который поможет не только генерировать идеи, но и развивать навыки общения. Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет предназначено для создания разнообразных сценариев общения, что позволит пользователю тренировать свои навыки, получить новые идеи и находить решения для различных ситуаций.

Основной целью курсового проектирования является создание интуитивно понятной и функциональной платформы, которая использует .NET MAUI для кроссплатформенной разработки и интеграции возможностей искусственного интеллекта. Это обеспечит доступность приложения на различных устройствах, что позволит пользователям взаимодействовать с ИИ в удобной для них среде.

Для достижения поставленной цели необходимо решить ряд задач:

- провести анализ потребностей пользователей, изучить актуальные тенденции в области ИИ и диалоговых систем, определить ключевые функциональные требования к приложению;
- создать основные функции генерации диалогов, включая настройку параметров взаимодействия, выбор тематики и стиля общения;
- разработать диаграммы, отражающие структуру приложения и взаимодействие его компонентов, что поможет в визуализации архитектуры системы;
- создать привлекательный и удобный интерфейс, обеспечивающий положительный пользовательский опыт и легкость в использовании;
- провести тестирование приложения для выявления и устранения ошибок, а также оптимизировать его производительность;
- подробно задокументировать процесс разработки, описывая ключевые принятые решения и полученные результаты, что позволит в дальнейшем улучшить приложение.

Документация к мобильному приложению «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет призвана обеспечить пользователей полным спектром информации о разработанном программном обеспечении.

В первой главе «Постановка задачи» будут определены требования к приложению, описаны задачи, решаемые с его помощью, и представлена информация о целевой аудитории. Также будет проведён анализ существующих аналогов, что обосновывает необходимость разработки нового программного обеспечения.

Во второй главе «Проектирование задачи» будет представлено описание диаграмм деятельности, вариантов использования и алгоритмов, что поможет обеспечить понимание архитектуры системы.

В третьей главе «Программная реализация» будет детализироваться структура приложения, модули и ключевые функции, а также приведены примеры кода для реализации основных возможностей.

В четвёртой главе «Тестирование и отладка приложения» будет проведён анализ качества программы и описание процесса отладки, что позволит выявить и исправить ошибки.

В пятой главе «Применение» будет предоставлено руководство пользователя, включающее инструкции по установке, описание функций и возможные проблемы с их решениями.

Конечным результатом данной работы станет не только создание функционального и современного приложения, но и вклад в развитие инструментов, способствующих более тесному и продуктивному взаимодействию человека с искусственным интеллектом. Разработка мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» отражает стремление к технологическому прогрессу, в котором важную роль играет удобство, доступность и практическая ценность для пользователя. В эпоху цифровизации подобные решения становятся неотъемлемой частью повседневной жизни, помогая людям адаптироваться к новым реалиям и максимально эффективно использовать возможности, которые предоставляет искусственный интеллект.

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Описание предметной области

Мобильное приложение – это программное обеспечение, разработанное специально для работы на мобильных устройствах, таких как смартфоны и планшеты. Эти приложения предназначены для выполнения различных задач и функций, обеспечивая пользователям доступ к информации, услугам и развлекательному контенту в удобном формате [1].

Мобильные приложения могут быть доступны для скачивания и установки из онлайн-магазинов приложений, таких как Google Play для устройств на базе «Android» и «App Store» для устройств на базе iOS. Они могут быть бесплатными или платными, с различными моделями монетизации, такими как реклама, покупка в приложении, подписки.

В зависимости от целей и функциональности, мобильные приложения могут быть разработаны как нативные, веб-ориентированные или гибридные:

- нативные приложения создаются специально для определенной операционной системы и обычно обеспечивают лучшую производительность и пользовательский опыт;
- веб-ориентированные приложения, с другой стороны, основаны на веб-технологиях работают через браузер мобильного устройства;
- гибридные приложения представляют собой комбинацию нативных и веб-ориентированных подходов [2].

Мобильные приложения имеют важное значение в современном информационном обществе, предлагая пользователям широкий спектр инструментов и сервисов для удовлетворения разнообразных потребностей. С помощью этих приложений люди могут не только развлекаться играми и общаться в социальных сетях, но и эффективно управлять своими финансами, обучаться, заботиться о здоровье и выполнять множество других задач.

ИИ – это область информатики, сосредоточенная на создании систем и программ, способных выполнять задачи, требующие интеллектуальных усилий, аналогичных человеческим. ИИ нацелен на то, чтобы машины могли обрабатывать информацию, обучаться на основе данных, принимать решения и решать проблемы.

Работа с ИИ становится все более важной в современном мире, так как она значительно изменяет подходы к решению различных задач и улучшению качества жизни. ИИ способен обрабатывать огромные объемы данных, выявляя скрытые паттерны и предоставляя ценные инсайты, что особенно полезно в таких областях, как медицина, финансы и образование.

С его помощью можно автоматизировать рутинные процессы, снижая трудозатраты и повышая эффективность. Это позволяет специалистам сосредоточиться на более сложных задачах, требующих креативного подхода и

человеческого интеллекта. ИИ также помогает в принятии более обоснованных решений, предоставляя аналитические инструменты и прогнозы на основе исторических данных.

Кроме того, технологии ИИ открывают новые горизонты для инноваций, создавая возможности для разработки новых продуктов и услуг. Важно отметить, что работа с ИИ требует понимания этических аспектов, таких как конфиденциальность данных и прозрачность алгоритмов. Исходя из этого, взаимодействие с ИИ – это не только путь к повышению эффективности, но и необходимость учитывать влияние технологий на общество и личную жизнь.

В целом, работа с искусственным интеллектом играет ключевую роль в современном мире, предлагая новые подходы к решению задач и улучшению качества жизни. ИИ позволяет обрабатывать большие объемы данных, автоматизировать рутинные процессы и принимать более обоснованные решения. Это способствует повышению эффективности и освобождает время для более креативных и сложных задач. Однако важно помнить о этических аспектах и влиянии технологий на общество. Исходя из этого, интеграция ИИ в различные сферы жизни открывает новые горизонты для инноваций и требует внимательного подхода к вопросам конфиденциальности и ответственности.

## **1.2 Обзор существующих аналогов**

На рынке существует множество приложений, использующих искусственный интеллект для взаимодействия с пользователями, включая голосовых помощников и чат-ботов. Аналоги, такие как Siri, Cortana и Replika, предлагают различные функции и возможности, удовлетворяющие разнообразные потребности пользователей. Эти приложения фокусируются на удобстве использования, интуитивности интерфейса и эмоциональной поддержке, что делает их популярными среди широкой аудитории. Изучение их опыта и особенностей позволит выявить лучшие практики и недостатки, которые могут быть учтены при разработке нового мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI».

На данный момент существует множество аналогов, но наиболее известные из них «Siri», «Cortana», «Replika».

«Siri» – это интеллектуальный голосовой помощник, разработанный компанией Apple, который был впервые представлен в 2011 году. Он использует сложные алгоритмы искусственного интеллекта и обработки естественного языка для взаимодействия с пользователями и выполнения их команд. «Siri» доступен на множестве устройств «Apple», включая «iPhone», «iPad», «Mac», «Apple Watch» и «HomePod», что делает его удобным инструментом для пользователей экосистемы «Apple».

Вид окна «Siri» представлен на рисунке 1.1.

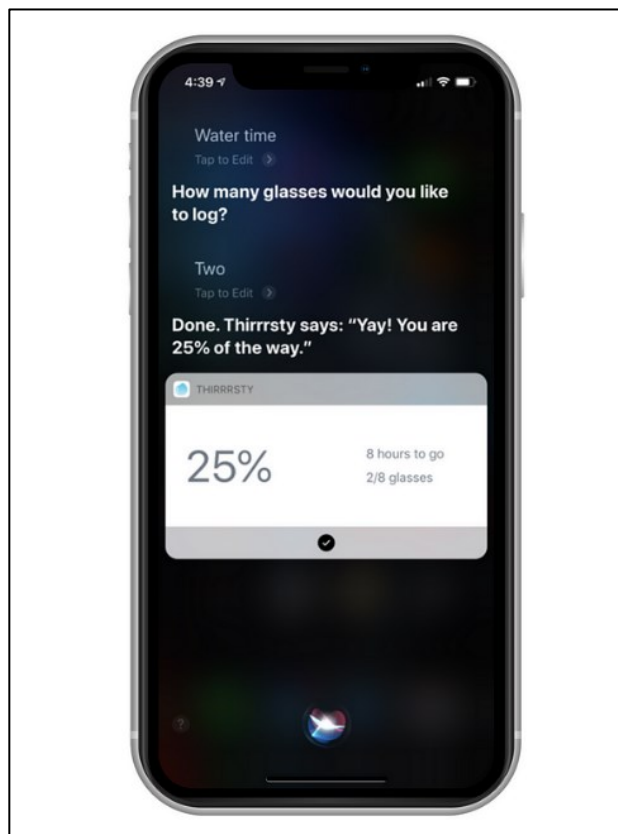


Рисунок 1.1 – Окно «Siri»

#### Преимущества:

- «Siri» позволяет пользователям выполнять задачи быстро и эффективно, не отрываясь от текущих дел;
- использование естественного языка делает взаимодействие с устройством более естественным и понятным, что особенно полезно для людей, не знакомых с технологией;
- «Siri» отлично работает с другими продуктами «Apple», обеспечивая плавное и согласованное взаимодействие между устройствами.

#### Недостатки:

- несмотря на постоянное развитие, «Siri» иногда может не понимать сложные или многозначные вопросы, что приводит к неверным ответам или неуместным рекомендациям;
- для выполнения многих функций требуется подключение к интернету, что может быть ограничением в условиях низкой связи или отсутствия Wi-Fi;
- некоторые пользователи могут беспокоиться о сборе данных и конфиденциальности при использовании голосового помощника, так как «Siri» обрабатывает и анализирует данные пользователей для улучшения своих функций.

«Cortana» – это голосовой помощник, разработанный компанией «Microsoft», который был впервые представлен в 2014 году. Названный в честь искусственного интеллекта из видеоигры «Halo», «Cortana» предназначен для выполнения различных

задач и улучшения взаимодействия пользователей с устройствами на базе «Windows».

Вид окна «Cortana» представлен на рисунке 1.2.

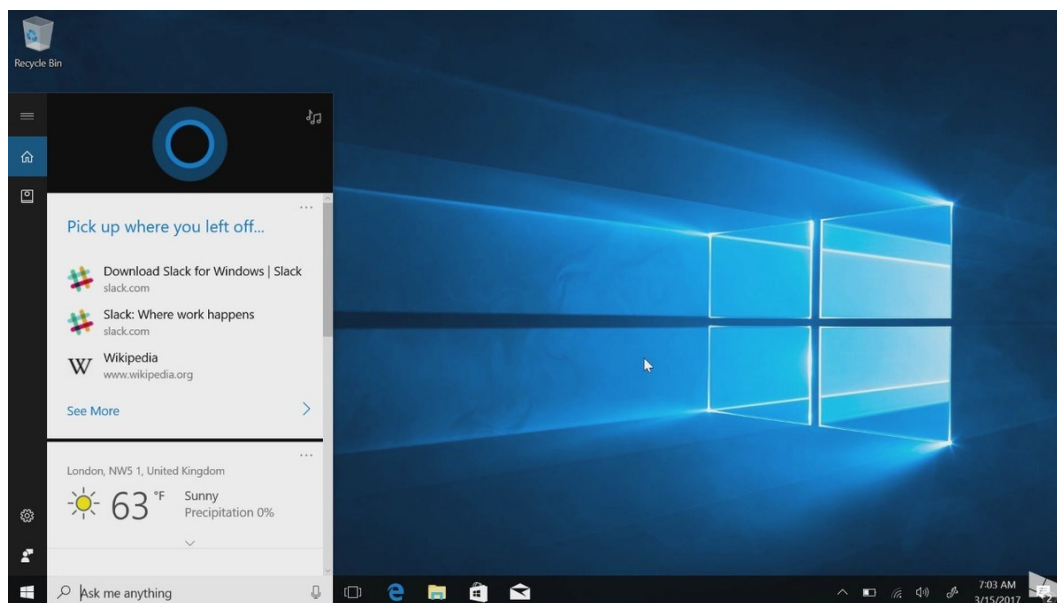


Рисунок 1.2 – Окно «Cortana»

Преимущества:

- «Cortana» упрощает выполнение задач, позволяя пользователям управлять устройствами голосом, что особенно полезно в условиях многозадачности;
- использование естественного языка делает взаимодействие с «Cortana» более простым и понятным.
- «Cortana» помогает пользователям оставаться организованными, отслеживая напоминания и важные события.

Недостатки:

- несмотря на широкий спектр функций, «Cortana» может уступать конкурентам в понимании сложных запросов и выполнении определенных задач;
- в последние годы «Microsoft» сократила поддержку «Cortana», что привело к снижению ее функциональности и интеграции с некоторыми приложениями;
- как и у других голосовых помощников, у пользователей могут возникнуть опасения по поводу сбора данных и конфиденциальности;
- «Cortana» больше не доступна на мобильных устройствах, что ограничивает ее использование для пользователей смартфонов.

«Replika» – это виртуальный чат-бот и голосовой помощник, созданный с использованием технологий искусственного интеллекта. Запущенный в 2017 году, «Replika» предназначен для предоставления эмоциональной поддержки, общения, а также для помощи пользователям в самопознании и развитии.

Вид окна «Replika» представлен на рисунке 1.3.





Рисунок 1.3 – Окно «Replika»

#### Преимущества:

- «Replika» предоставляет пользователям возможность выразить свои чувства и получить поддержку без осуждения;
- общение с ботом позволяет пользователям делиться своими мыслями и переживаниями в безопасной и конфиденциальной обстановке;
- пользователи могут общаться с «Replika» в любое время, что делает ее удобным ресурсом для поддержки и самопознания.

#### Недостатки:

- несмотря на использование технологий искусственного интеллекта, «Replika» может не всегда точно воспринимать сложные эмоции или ситуации, что может привести к недопониманию;
- некоторые пользователи могут начать полагаться на виртуальную поддержку, что может отвлекать их от общения с реальными людьми;
- пользователи могут испытывать беспокойство по поводу обработки и хранения их данных и разговоров.

Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» должно иметь интуитивно понятный интерфейс, который облегчает пользователям взаимодействие с ИИ. Дизайн приложения должен быть простым и

логичным, чтобы даже новички могли легко ориентироваться в его функциональности.

Удобство использования должно достигаться за счет минималистичного оформления, четкой навигации и понятных элементов управления. Например, кнопки и меню должны быть расположены так, чтобы пользователи могли быстро находить нужные функции без лишних кликов.

Важным аспектом является использование естественного языка в интерфейсе, что позволяет пользователям формулировать запросы так, как они бы общались с другом. Это сделает взаимодействие более естественным и снижает барьер входа для тех, кто не знаком с технологиями.

Также стоит предусмотреть адаптивный дизайн, который хорошо будет выглядеть и функционировать на различных устройствах, будь то смартфон или планшет. Это обеспечит комфортное использование приложения в любых условиях.

Исходя из этого, интуитивно понятный интерфейс и удобство использования являются ключевыми факторами, которые помогут пользователям эффективно взаимодействовать с приложением и получать от него максимальную пользу.

### **1.3 Информационная база задачи**

Информационная база – это определенным способом организованная совокупность данных, хранимых в памяти вычислительной системы в виде файлов, с помощью которых удовлетворяются информационные потребности управленческих процессов и решаемых задач.

Входные данные – это набор данных, которые программа получает на входе, чтобы выполнить определенную обработку или операцию. Информация может быть представлена в различных форматах, например, в виде чисел, текста, изображений, аудио или видео файлов, или других типов данных, в зависимости от типа программы и функциональности.

Выходные данные – это результат работы программы, который возвращается после выполнения определенных вычислений, операций или обработки входных данных. Это могут быть данные, которые выводятся на экран, сохраняются в файл или передаются другой программе [3].

Входными данными для разрабатываемого мобильного приложения являются:

- электронная почта, это уникальный идентификатор пользователя, используемый для аутентификации и связи;
- пароль, это секретный код, который пользователь вводит для доступа к своему аккаунту;
- запрос пользователя, это текстовый запрос или вопрос, который пользователь хочет задать ИИ;

- имя пользователя, это имя, под которым пользователь будет обращаться к ИИ, что добавляет персонализацию в диалог;

- API-key, ключ доступа к сторонним API, необходимый для отправки запросов к ИИ и получения ответов.

Выходными данными являются:

- ответ ИИ, это результат обработки запроса пользователя, который возвращается в виде текстового ответа;

- история диалогов, это сохранённые обмены сообщениями между пользователем и ИИ, которые могут быть отображены в интерфейсе приложения;

- статистика взаимодействия, это информация о количестве запросов, времени ответа и других метриках, которые могут быть полезны для анализа использования приложения;

- в случае неудачи при обработке запроса приложение должно возвращать соответствующее сообщение об ошибке.

В заключение, мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет предлагать пользователям удобный и персонализированный способ взаимодействия с искусственным интеллектом. Используя надежные методы аутентификации и предоставляя разнообразные выходные данные, такие как ответы ИИ и история диалогов, приложение обеспечивает качественный пользовательский опыт. Статистика взаимодействия и обработка ошибок дополнительно укрепят функциональность системы, позволяя пользователям эффективно анализировать свое взаимодействие с ИИ и получать необходимую поддержку. Такой подход сделает приложение не только удобным, но и эффективным инструментом для общения и получения информации.

## **1.4 Функциональное назначение**

Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет разработано с целью помочь пользователям эффективно взаимодействовать с искусственным интеллектом, улучшая качество общения и предоставляя поддержку в различных аспектах повседневной жизни.

Мобильное приложение должно обладать различными функциями, каждая из которых направлена на улучшение повседневного взаимодействия пользователей с ИИ.

Главная страница мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет представлять собой интуитивно понятный и дружелюбный интерфейс для начала общения с искусственным интеллектом. В верхней части экрана будет размещено название приложения и иконка меню, позволяющая пользователю получить доступ к дополнительным функциям. Центральное место будет занимать изображение симпатичного робота, создающего

ощущение персонального помощника и вызывающего доверие. В нижней части экрана будет расположено текстовое поле для ввода сообщений с кнопкой отправки в виде стрелки, позволяющей незамедлительно начать разговор. Такой дизайн сделает процесс общения простым, комфортным и доступным для всех категорий пользователей. Макет главной страницы представлен на рисунке 1.4.

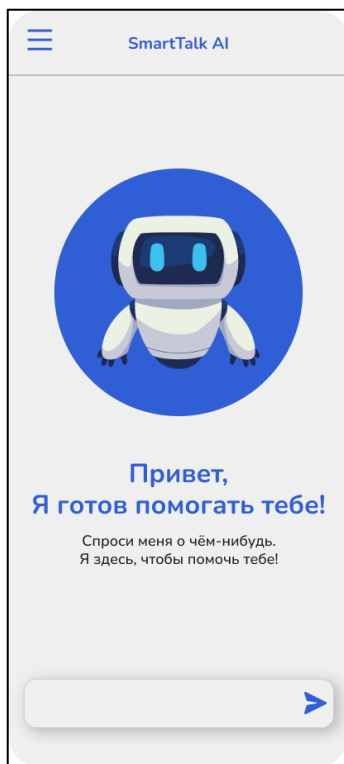


Рисунок 1.4 – Макет главной страницы

После того как пользователь отправит сообщение, интерфейс автоматически переключится на вкладку чата, где отобразится диалог с искусственным интеллектом. Это позволит сразу увидеть ответ ИИ и продолжить взаимодействие в привычном формате. Вкладка чата служит основным пространством для общения, поэтому после каждого действия, связанного с отправкой запроса, пользователь направляется именно туда, чтобы не терять нить разговора и удобно следить за его развитием. Макет чата представлен на рисунке 1.5.

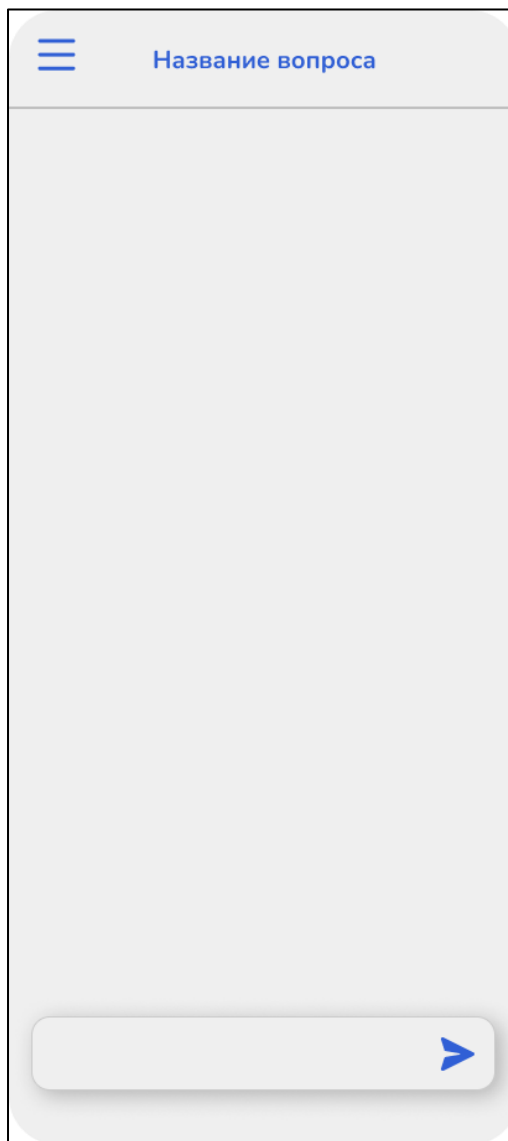


Рисунок 1.5 – Макет чата

В разделе «Боковая панель» пользователю будет предоставляться доступ к истории его взаимодействий с ИИ, представленной в виде списка предыдущих диалогов. Это пространство будет организовано таким образом, чтобы обеспечить удобную навигацию и быстрый доступ к нужным беседам. Интерфейс позволит без лишних усилий возвращаться к ранее начатым темам, просматривать контекст прошедших разговоров и при необходимости продолжать их с того места, где общение было прервано. Такой подход способствует формированию более связного и осмысленного взаимодействия, в котором каждый сеанс общения сохраняет свою ценность и может быть использован повторно. Макет боковой панели представлен на рисунке 1.6.

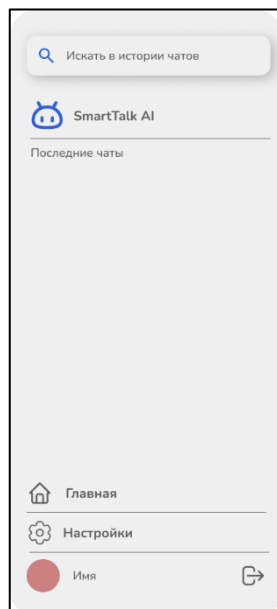


Рисунок 1.6 – Макет боковой панели

Приложение будет реализовывать полноценный механизм управления доступом, включающий процессы авторизации и регистрации. Пользователь при первом входе сможет создать персональный аккаунт, введя необходимые данные в удобной форме, представленной на соответствующем интерфейсе. После регистрации система сохранит учетную запись, что позволит в дальнейшем входить в приложение с помощью логина и пароля. Макет окна регистрации представлен на рисунке 1.7.

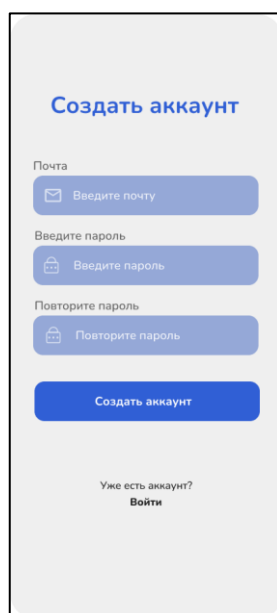


Рисунок 1.7 – Макет вкладки «Регистрации»

В процессе авторизации будут предусмотрены механизмы защиты данных, которые обеспечивают конфиденциальность и безопасность доступа. Если

пользователь по каким-либо причинам потеряет доступ к своему аккаунту, будет предусмотрена возможность восстановления пароля: достаточно следовать пошаговым подсказкам на экране, чтобы восстановить доступ без лишних затруднений. Макет окна авторизации представлен на рисунке 1.8.

Войти в аккаунт

Почта  
Введите почту

Пароль  
Повторите пароль

Забыли пароль?

Войти

Нет аккаунта?  
Создать аккаунт

Рисунок 1.8 – Макет вкладки «Авторизации»

Если пользователь уже будет иметь зарегистрированную учетную запись, но не сможет вспомнить пароль, система предоставляет возможность восстановления доступа через специальную вкладку «Восстановить пароль». Переход к этой вкладке будет осуществляться прямо с экрана авторизации, что сделает процесс интуитивно понятным и доступным. На открывшейся странице пользователю будет предлагаться ввести адрес электронной почты, связанный с его аккаунтом. После подтверждения система отправит на указанный адрес инструкции по восстановлению пароля в виде ссылки. Такой подход обеспечит безопасность и в то же время позволяет восстановить доступ без лишних сложностей. Макет вкладки «Восстановление пароля» представлен на рисунке 1.9.

The image shows a mobile application interface for password recovery. At the top, the title "Восстановить пароль" (Reset password) is displayed in blue. Below the title, the word "Почта" (Email) is shown. There is a light blue input field with a mail icon and the placeholder text "Введите почту" (Enter email). Below the input field is a solid blue button with the text "Восстановить пароль" (Reset password). At the bottom of the screen, there is a link that says "Уже есть аккаунт? Войти" (Already have an account? Log in).

Рисунок 1.9 – Макет вкладки «Восстановить пароль»

Целевая аудитория мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет включать людей, стремящихся повысить свою эффективность и улучшить общее благополучие. Это могут быть занятые профессионалы, студенты, предприниматели, родители, а также все, кто хочет оптимизировать свои рабочие процессы и получать поддержку в повседневной жизни.



## 2 ПРОЕКТИРОВАНИЕ ЗАДАЧИ

### 2.1 Алгоритм решения задачи

Разрабатываемое мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» будет представлять собой удобное средство для взаимодействия с искусственным интеллектом, предназначенное для поддержки пользователей в различных аспектах повседневной жизни. Приложение будет выполнено с интуитивно понятным интерфейсом, что обеспечит простоту и доступность использования. Оно будет разделено на несколько функциональных блоков, каждый из которых отвечает за выполнение конкретных задач: авторизация и регистрация пользователей, управление историей диалогов, восстановление доступа, отправка и получение сообщений от ИИ. Блок-схема мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» представлена на рисунке 2.1.

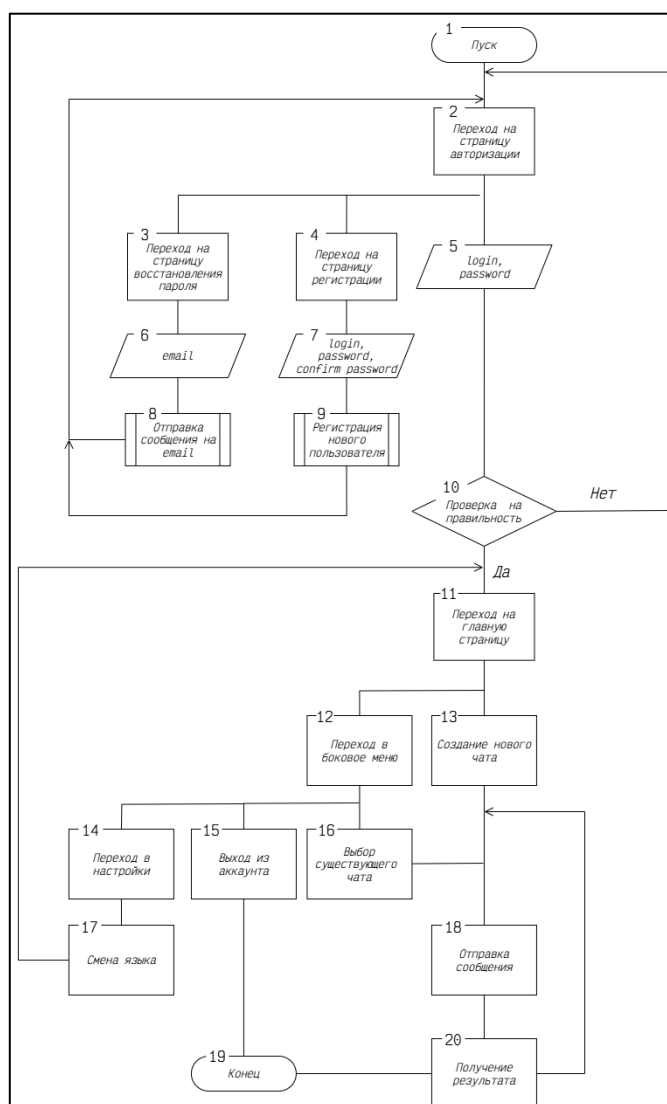


Рисунок 2.1 – Блок-схема мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»

При первом запуске приложения пользователю отобразится экран авторизации, который включает:

- поле для ввода логина и пароля;
- кнопку «Войти» для авторизации;
- кнопку «Зарегистрироваться» для создания нового аккаунта.

Если пользователь еще не зарегистрирован, он может пройти процесс регистрации, указав логин, пароль, подтверждение пароля и, при необходимости, тип профиля. После успешной регистрации данные сохраняются в базе данных, и пользователь перенаправляется обратно на экран авторизации для входа.

После ввода логина и пароля система проверяет их на корректность:

- при совпадении данных пользователь попадает на главный экран приложения;
- при ошибке авторизации отображается соответствующее сообщение;
- при нескольких неудачных попытках система предлагает восстановление пароля.

После успешного входа в систему пользователь попадает на главный экран приложения, где отображается визуальное представление ИИ (персонаж-ассистент), текстовое поле для ввода сообщений и кнопка отправки. Это позволяет незамедлительно начать общение. После отправки сообщения приложение автоматически переключает пользователя на вкладку чата, где отображается полный диалог с ИИ в хронологическом порядке, создавая ощущение живого общения.

В приложении будет реализован раздел «Боковая панель», предоставляющий доступ к списку предыдущих диалогов. Каждый разговор сохраняется в виде отдельной сессии и может быть открыт повторно для просмотра или продолжения. Это упростит навигацию по истории общения, позволяя отслеживать важные темы и не терять контекст.

Приложение будет поддерживать множество полезных функций для улучшения пользовательского опыта:

- при выборе языка пользователь сможет адаптировать интерфейс приложения под свои предпочтения, используя кнопку смены языка в настройках с возможностью выбора из нескольких языков (русский, английский и другие);
- при поиске чатов по ключевому слову в разделе истории диалогов пользователю будет предоставляться возможность быстро найти нужный чат, если в нем обсуждалась важная тема или содержится необходимая информация;
- при выборе чата в боковой панели с историей чатов пользователь сможет открыть любой сохранённый чат для продолжения разговора или для просмотра предыдущих обсуждений;
- при выходе из аккаунта пользователь сможет завершить текущую сессию и вернуться на экран авторизации, используя кнопку «Выход» в настройках приложения.

Если пользователь забыл пароль, он сможет воспользоваться вкладкой восстановления доступа. Для этого необходимо ввести адрес электронной почты, на который будет отправлено письмо с инструкциями по восстановлению пароля, что позволит без труда вернуть доступ к учетной записи.

Для обеспечения стабильной и безопасной работы приложения будут реализованы следующие механизмы:

- при проверке корректности ввода данных приложение будет выводить предупреждения или запросы на исправление, если пользователь пытается выполнить некорректное действие (например, отправить пустое сообщение);
- при логировании ошибок приложение будет записывать критические ошибки, что помогает быстро реагировать на возникающие проблемы;
- при шифровании данных все пользовательские данные, включая пароли и личную информацию будут шифроваться для обеспечения высокого уровня защиты и конфиденциальности.

Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» предоставит пользователям удобный инструмент для взаимодействия с искусственным интеллектом, поможет решить повседневные задачи и получить поддержку в интерактивной форме. Оно будет предлагать функционал для работы с чатом, историю диалогов, поиск, по ключевым словам, и смену языка интерфейса. Приложение также обеспечит высокий уровень безопасности, защиты персональной информации и удобство в использовании. Это решение способствует улучшению цифровой грамотности и помогает пользователям эффективно использовать ИИ для решения различных задач.

## **2.2 Логическое моделирование**

Диаграмма деятельности представляется в виде графа, где узлы представляют собой состояния действий, а стрелки – переходы между этими состояниями. Отображает логику или последовательность выполнения действий, где акцент сделан на результате этих действий [4]. Графическое представление помогает описать порядок выполнения задач или операций в процессе, подчеркивая последовательность и завершенность каждого действия для достижения определенного результата. Компонентами диаграммы деятельности являются:

- состояния действия;
- переходы;
- дорожки;
- символы слияния и ветвления;
- символы разделения и слияния параллельных потоков управления;
- объекты.

Диаграмма деятельности обладает следующими преимуществами. Во-первых, она обеспечивает визуализацию процесса, позволяя наглядно представить

последовательность действий и поток управления в системе или процессе. Во-вторых, она способна отобразить альтернативные и параллельные потоки, что помогает лучше понять различные варианты выполнения действий и параллельные процессы. В-третьих, диаграмма деятельности представлена в графическом виде, что делает ее удобной для восприятия и понимания.

Однако у диаграммы деятельности есть некоторые недостатки. Во-первых, она может сокращать детали и сложности процесса, описывая его на более высоком уровне. Это может привести к упрощению и потере некоторых важных деталей. Во-вторых, диаграмма не предоставляет точной информации о временных ограничениях и продолжительности выполнения действий. Это может ограничить ее способность полноценно отразить аспекты, связанные со временем.

Диаграмма деятельности применяется для моделирования бизнес-процессов, анализа и оптимизации работы системы, визуализации последовательности действий и потока управления, а также для обеспечения понимания и коммуникации между участниками проекта.

В целом, диаграмма деятельности является полезным инструментом для визуализации последовательности действий и потока управления в системе или процессе, но ее использование должно сопровождаться дополнительной документацией и детализацией для полного понимания процесса.

Диаграмма деятельности отражает последовательность пользовательских действий в процессе работы с приложением. Работа начинается с запуска приложения. Далее пользователь либо нажимает кнопку регистрации, либо переходит к авторизации. В случае регистрации пользователь вводит почту, затем пароль и подтверждает его, после чего переходит к следующему этапу. При выборе авторизации предусмотрен сценарий восстановления пароля. Если пользователь нажимает соответствующую кнопку, он переходит на страницу восстановления, вводит почту и возвращается к авторизации.

После успешной регистрации или авторизации пользователь попадает на главный экран. Здесь он может либо нажать кнопку «Создать диалог», либо открыть боковое меню, где доступна история диалогов и возможность выбрать нужный диалог. В обоих случаях осуществляется переход к диалоговому окну, где пользователь отправляет запрос и получает ответ. После отправки сообщения возможен выход из приложения через боковое меню. Завершение работы отображается конечной точкой диаграммы.

Диаграмма иллюстрирует типичную логику пользовательского взаимодействия с системой, обеспечивая наглядное представление потока управления и облегчая анализ бизнес-процесса.

Диаграмма деятельности представлена на плакате МРКК.508909.001ПЛ.

Диаграмма показывает, как пользователь может перемещаться между различными вкладками приложения для доступа к основным функциям.

Само приложение в будущем будет предоставлять пользователю удобный и интуитивно понятный интерфейс для общения с ИИ. Пользователи смогут регистрироваться, авторизовываться, восстанавливать доступ к учетной записи, создавать новые диалоги и просматривать историю общения. Интерфейс будет направлен на обеспечение простоты и логичности навигации, а также на поддержание постоянного взаимодействия с искусственным интеллектом.

Диаграмма вариантов использования в UML – это графическое представление, которое демонстрирует взаимосвязь между актерами и прецедентами. Является частью модели прецедентов в UML и используется для описания системы на концептуальном уровне. Эта диаграмма позволяет увидеть, как пользователи взаимодействуют с системой через ее функциональные компоненты и какие задачи или сценарии они выполняют [5].

Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
- сформулировать общие требования к функциональному поведению проектируемой системы;
- разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Основными компонентами диаграммы вариантов использования являются:

- варианты использования;
- актеры;
- интерфейсы;
- примечания;
- отношения.

Диаграмма вариантов использования имеет ряд преимуществ. Во-первых, она обеспечивает визуализацию функциональности системы и ее взаимодействия с актерами. Это позволяет наглядно представить, как система будет использоваться и какие требования она должна удовлетворять. Во-вторых, диаграмма способствует понятности и облегчает коммуникацию между участниками проекта. Участники могут лучше понять, как система будет функционировать и какие задачи она будет выполнять. Кроме того, диаграмма вариантов использования помогает определить основные функциональные требования и приоритеты системы, что позволяет сосредоточиться на наиболее важных аспектах.

Однако у диаграммы вариантов использования есть и некоторые недостатки. Во-первых, она ограничена и может не учитывать все детали и сложности системы. Так как она сконцентрирована на высокоуровневом описании функциональности, могут быть упущены некоторые важные детали. Во-вторых, диаграмма не отображает временную последовательность действий или временные ограничения, связанные с каждым вариантом использования. Это может быть недостаточно для полного понимания системы. Наконец, диаграмма не заменяет полного описания требований и может быть недостаточной для подробного изучения и понимания системы.

Диаграмма вариантов использования применяется для анализа требований, визуализации функциональности системы, уточнения требований и обеспечения коммуникации и согласования между участниками проекта.

Диаграмма вариантов использования представлена на плакате МРКК.508909.002ПЛ.

Приложение будет включать функции авторизации и регистрации, что позволит пользователям создавать свои аккаунты и обеспечит безопасность личных данных. При необходимости пользователи смогут восстановить забытые пароли, следуя простым инструкциям для получения доступа к своим учетным записям.

После успешного входа в аккаунт пользователи смогут легко начинать новые беседы с ИИ на главной странице, которая обеспечит интуитивно понятный интерфейс для общения и получения ответов в реальном времени.

Главная страница приложения будет представлять собой интуитивно понятный и дружелюбный интерфейс, предназначенный для начала взаимодействия с ИИ. В верхней части экрана будет отображаться название приложения и кнопка меню, открывающая доступ к дополнительным функциям. Центральное место займет изображение симпатичного робота, создающего ощущение персонального помощника и вызывающего доверие у пользователя. В нижней части разместится поле для ввода текста и кнопка отправки в виде стрелки, которая позволит немедленно начать беседу. Такой дизайн сделает использование приложения простым и доступным для широкой аудитории. После отправки сообщения интерфейс автоматически переключится на вкладку чата, где будет отображаться текущий диалог с искусственным интеллектом. Это обеспечит непрерывность общения и позволит удобно отслеживать ход беседы.

Вкладка чата будет служить основным пространством для взаимодействия с ИИ. Здесь будут отображаться все сообщения пользователя и ответы искусственного интеллекта в формате привычной переписки. Интерфейс будет устроен таким образом, чтобы пользователь мог легко продолжать разговор, не теряя контекста. После каждой отправки сообщения пользователь останется в окне чата, что повысит удобство и будет способствовать более эффективному взаимодействию.

Раздел «Диалоги» будет реализован в виде боковой панели, предоставляющей пользователю доступ к истории предыдущих взаимодействий с ИИ. Диалоги будут представлены списком, что позволит быстро находить нужные разговоры, просматривать их содержание и при необходимости возвращаться к обсуждению. Такая организация данных сохранит важную информацию и сделает взаимодействие с приложением более осмысленным и продуктивным.

Такое логическое моделирование сыграет важную роль в более глубоком осмыслении и анализе всего процесса взаимодействия пользователя с разработанным приложением, а также в более детальном понимании всех функциональных возможностей, которые оно будет предлагать. Этот процесс моделирования будет способствовать не только структурированию ключевых компонентов программы, но и позволит ясно и четко описать их взаимосвязи, что поможет сформировать более полное представление о логике и архитектуре приложения. В результате это позволит не только оптимизировать процессы разработки, но и обеспечит лучшее понимание принципов работы системы, ее взаимодействий и потенциала для будущих улучшений.

### **2.3 Описание инструментов разработки**

Инструменты разработки – это набор сред разработки и других технических средств, которые используются для создания программного решения. Для разработки мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» были использованы такие инструменты разработки как язык программирования C#, интегрированная среда разработки «Visual Studio 2022», графический редактор «Figma», «.NET Firebase SDK», база данных «Firebase Authentication», фреймворк «Xamarin».

C# – это высокоуровневый объектно-ориентированный язык программирования, разработанный компанией Microsoft. C# является одним из основных языков разработки приложений для платформы «.NET Framework» и других сред, таких как .NET Core и «Xamarin». C# сочетает в себе элементы языков C и C++ с удобством и эффективностью языков высокого уровня, таких как Java.

C# предназначен для создания широкого спектра программных приложений, включая десктопные приложения, веб-приложения, мобильные приложения, игры и многое другое. Он обладает мощными средствами для работы с объектами, событиями, обработкой исключений, асинхронным программированием и многими другими возможностями. Кроме того, C# имеет большую и активную сообщество разработчиков, что обеспечивает обширную поддержку и доступ к разнообразным библиотекам и фреймворкам, упрощающим процесс разработки и увеличивающим производительность [6].

В рамках разработки мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI», C# будет использоваться для написания бизнес-

логики, пользовательского интерфейса и взаимодействия с внешними сервисами, обеспечивая эффективное и надежное функционирование всех его компонентов.

«Visual Studio» является одной из наиболее популярных и мощных интегрированных сред разработки для разработки программного обеспечения. Эта среда обладает рядом значительных преимуществ.

«Visual Studio» обеспечивает широкий набор инструментов для создания различных типов приложений, таких как консольные программы, веб-приложения и мобильные приложения, повышая эффективность и качество работы разработчиков. Его интеграция с другими инструментами и платформами, такими как Git и отладчик, облегчает процесс разработки и совместную работу в команде.

Однако «Visual Studio» может быть сложным для новичков из-за мощных функций и интерфейса, требующего времени для освоения. Кроме того, платная лицензия может стать преградой для отдельных разработчиков, а ориентация на платформу Windows может ограничить возможности кроссплатформенной разработки [7].

«Visual Studio» был выбран в качестве интегрированной среды разработки для создания мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» благодаря его обширным возможностям и интеграции с платформой «Xamarin».

«Figma» – это современный онлайн-инструмент для проектирования пользовательских интерфейсов и прототипирования, который активно используется в процессе разработки цифровых продуктов. Его ключевым преимуществом является возможность работы прямо в браузере, что избавляет пользователей от необходимости устанавливать дополнительное программное обеспечение и делает платформу доступной с любых устройств и операционных систем. Благодаря этому обеспечивается гибкость и удобство при подключении новых участников команды и организации удалённой совместной работы.

Особенностью «Figma» является поддержка режима одновременного редактирования, при котором несколько пользователей могут параллельно работать над одним и тем же проектом, видеть действия друг друга в реальном времени и оперативно обмениваться обратной связью. Это значительно ускоряет процессы обсуждения и принятия решений, повышает прозрачность рабочих процессов и способствует более слаженной командной работе.

«Figma» предоставляет полноценный набор инструментов для создания визуального дизайна, макетов экранов, векторной графики и интерактивных прототипов, что делает её универсальным решением для дизайнеров и разработчиков. Встроенные функции позволяют создавать переиспользуемые интерфейсные компоненты и единые библиотеки стилей, благодаря чему удаётся легко поддерживать визуальную целостность и согласованность в рамках всего проекта.



В разработке мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI И AI» «Figma» используется как основной инструмент для проектирования пользовательского интерфейса. С её помощью создаются макеты экранов, прорабатывается структура навигации, формируются интерактивные прототипы, демонстрирующие поведение элементов в реальном взаимодействии. Это позволяет ещё на этапе планирования оценить удобство использования приложения и внести необходимые изменения до начала разработки. Таким образом, «Figma» играет важную роль в обеспечении качества, целостности и визуальной привлекательности интерфейса приложения.

«Firebase Authentication» – это удобный облачный сервис от компании Google, который значительно упрощает процесс аутентификации пользователей в мобильных и веб-приложениях. Его основная задача – избавить разработчиков от необходимости создавать сложные механизмы входа и управления пользователями вручную. Благодаря своей продуманной архитектуре и широкому функционалу, «Firebase Authentication» обеспечивает простой, но в то же время надёжный способ реализации системы входа в приложение. Он поддерживает множество методов аутентификации, включая стандартный вход с использованием электронной почты и пароля, а также авторизацию через популярные социальные сети, вход по номеру телефона и даже анонимную аутентификацию. Такой подход позволяет гибко адаптировать процесс входа под разные сценарии использования и предпочтения пользователей.

Сервис легко интегрируется с остальными продуктами экосистемы «Firebase», что даёт возможность быстро развернуть готовое решение и сосредоточиться на разработке самого приложения, не тратя время на реализацию базовой инфраструктуры. Одним из важных преимуществ «Firebase Authentication» является использование современных стандартов безопасности – например, аутентификация на основе токенов, что позволяет надёжно защищать пользовательские данные. Разработчики также получают в своё распоряжение удобные инструменты для управления аккаунтами, восстановления доступа, проверки подлинности и отслеживания состояния сеанса.

«Firebase Authentication» одинаково хорошо работает на разных платформах – будь то веб, Android или iOS – что особенно важно в условиях кроссплатформенной разработки. Это делает сервис универсальным решением, которое помогает ускорить процесс создания приложений и при этом повысить их надёжность и безопасность.

Кроме того, в рамках разработки мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» используется «.NET Firebase SDK» – это мощный и гибкий набор инструментов, предназначенный для интеграции с облачными сервисами Firebase при создании приложений на платформе .NET. SDK предоставляет доступ ко всем ключевым возможностям «Firebase», таким как аутентификация пользователей, хранение данных в облаке, отправка уведомлений,

работа с базами данных в реальном времени и многое другое. Благодаря наличию хорошо документированных API и высокой степени совместимости с различными средами разработки, такими как «ASP.NET», «Xamarin», «Unity» и другими .NET-платформами, этот инструмент значительно облегчает процесс интеграции и делает разработку более эффективной.

«Xamarin» – это открытый фреймворк для разработки мобильных приложений, который позволяет разработчикам использовать язык программирования C# и среду разработки .NET для создания кроссплатформенных приложений под «Android», «iOS» и другие платформы [9].

Основным преимуществом «Xamarin» является возможность создания приложений, использующих общий код для нескольких платформ. Вместо того чтобы писать отдельные версии приложений для каждой платформы, разработчики могут использовать один и тот же код на языке C# для создания приложений, которые могут быть запущены на разных устройствах и операционных системах.

«Xamarin» позволяет разработчикам использовать все преимущества языка C#, включая его современные возможности, такие как LINQ, асинхронное программирование и управление памятью. Он также интегрируется с экосистемой .NET, что обеспечивает доступ к широкому набору библиотек и инструментов для разработки приложений.

Кроме того, «Xamarin» обеспечивает высокую степень повторного использования кода благодаря концепции общего кода, позволяя разработчикам разделять логику приложения между различными платформами, при этом поддерживая возможность создания интерфейса, адаптированного для каждой платформы.

В целом, «Xamarin» представляет собой мощный инструмент для разработки кроссплатформенных мобильных приложений на базе .NET, обеспечивая высокую производительность, доступ к современным возможностям языка C# и широкие возможности повторного использования кода.

«Xamarin» будет использоваться в разработке мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» для создания общей кодовой базы приложения. Этот фреймворк будет применяться во всех частях приложения, включая логику бизнес-процессов, взаимодействие с базой данных, управление пользовательским интерфейсом и обработку событий.

## 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

### 3.1 Физическая структура

Физическая структура программного обеспечения представляет собой организацию компонентов и их взаимодействие на физическом уровне. Она описывает расположение, структуру и взаимосвязь файлов, каталогов и ресурсов, необходимых для функционирования программы на локальном компьютере или сервере. Правильная организация этих элементов критична для обеспечения эффективной работы приложения, его поддержки и масштабируемости.

Весь проект приложения содержится в папке с именем «RealTalkAI» и включает в себя ключевые папки. Главный каталог мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» представлен на рисунке 3.1.

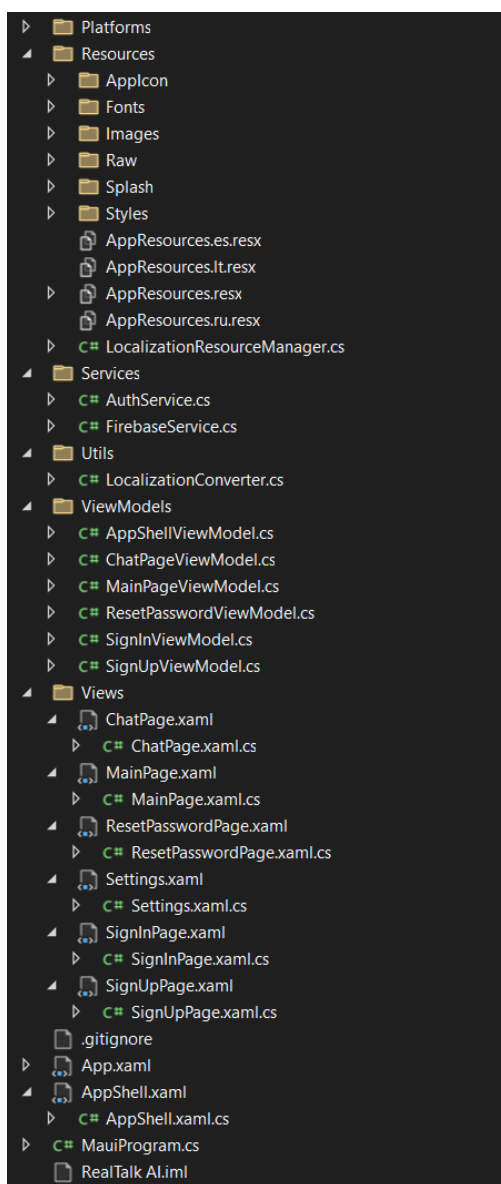


Рисунок 3.1 – Главный каталог

Папка «Assets» содержит все графические материалы и ресурсы, используемые в приложении, такие как изображения, иконки и другие медиафайлы. Эти ресурсы позволяют создать визуально привлекательный интерфейс и улучшают пользовательский опыт.

В папке «Models» находятся файлы, определяющие модели данных, которые используются в приложении. Модели представляют собой структуры данных, которые описывают основные объекты приложения и их свойства, обеспечивая единый интерфейс для работы с данными.

Папка «Services» включает файлы, ответственные за бизнес-логику приложения и взаимодействие с внешними сервисами. Сервисы управляют операциями, такими как обращение к API, работа с базами данных и выполнение других задач, связанных с обработкой данных.

В папке «ViewModel» содержатся файлы, которые связывают модели данных и представления. ViewModels обеспечивают логику, необходимую для отображения данных в пользовательском интерфейсе, и управляют взаимодействием пользователя с приложением.

В папке «Views» находятся файлы с расширениями .xaml и .xaml.cs, которые определяют внешний вид и логику отображения страниц приложения. Эти файлы отвечают за пользовательский интерфейс и взаимодействие с пользователем.

В корневой части проекта расположены важные файлы, которые определяют глобальные ресурсы, стили и точку входа в приложение:

Файлы «App.xaml» и «App.xaml.cs» определяют глобальные ресурсы и настройки приложения, такие как цвета, шрифты и стили, а также задают точку входа в приложение. «App.xaml» содержит XAML-разметку, обеспечивающую доступ к ресурсам, а «App.xaml.cs» содержит код для инициализации приложения и обработки его жизненного цикла.

Файлы «AppShell.xaml» и «AppShell.xaml.cs» определяют основную структуру пользовательского интерфейса приложения с использованием Shell. Shell является контейнером для различных страниц и представлений, определяя навигацию и внешний вид приложения. «AppShell.xaml» описывает разметку, а «AppShell.xaml.cs» управляет логикой навигации и функциональностью интерфейса.

Таким образом, структура проекта организована логично и включает в себя файлы интерфейсов, их обработчики, конфигурационные файлы, а также базу данных, обеспечивающую хранение информации.

### **3.2 Описание разработанных модулей**

Модуль в программировании – это независимая часть кода, которая выполняет конкретную задачу или предоставляет определенные функции. Он включает в себя связанные функции, переменные и другие элементы, объединенные для решения определенных проблем.

Использование модулей помогает организовать и структурировать код, делает его проще для сопровождения и повторного использования. Модули позволяют разбивать большие программы на меньшие и логически связанные части, что облегчает их понимание и разработку.

Для реализации функционала проекта были разработаны различные модули.

Для переключения между окнами был реализован AppShell, а также возможность возврата на предыдущее окно с помощью кнопки «Назад». С помощью AppShell можно перейти на вкладку «Главная», а также на вкладку «Настройки», где можно сменить язык приложения. В боковой панели можно найти чат по ключевому слову либо перейти на уже существующий чат через историю чатов.

Для мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» был разработан несколько ключевых программных модулей, каждый из которых выполняет свою функцию, способствуя реализации основной логики приложения. Один из таких модулей – AuthService, который управляет процессом авторизации и хранения пользовательских данных. Основные функции данного модуля:

- GetAuthToken(), получает токен авторизации текущего пользователя;
- SetAuthToken(string token), устанавливает токен авторизации;
- IsUserAuthenticated(), проверяет, авторизован ли пользователь на основе наличия токена;
- Logout(), выполняет выход из системы, удаляя все сохраненные данные;
- SetUserEmail(string email), сохраняет email пользователя;
- GetUserEmail(), получает email пользователя;
- SetUserId(string userId), сохраняет идентификатор пользователя;
- GetUserId(), получает идентификатор пользователя;
- SetUsername(string nickname), сохраняет никнейм пользователя;
- GetUsername(), получает никнейм пользователя.

Модуль FirebaseService интегрируется с базой данных Firebase и обеспечивает сохранение истории чатов пользователей. Он также поддерживает создание и обновление чатов, а также извлечение данных о чатах из базы. Основные функции данного модуля: SaveChatHistoryAsync(), сохраняет историю чатов пользователя, включая сообщения и информацию о пользователе, также поддерживает возможность сохранения данных как для новых, так и для уже существующих чатов.

Модуль LocalizationConverter используется для работы с локализацией и переводами в приложении. Он предоставляет возможность конвертации значений в зависимости от выбранного языка приложения. Основная цель этого модуля –

обеспечить многоязычность в интерфейсе, что позволяет приложению быть доступным для более широкой аудитории.

Модули ViewModel в приложении отвечают за управление логикой представления и обновление интерфейса в зависимости от изменений состояния или локализации. Они обеспечивают связь между моделью данных и пользовательским интерфейсом, обновляя отображаемую информацию в реальном времени. Каждый из этих модулей следит за состоянием данных и предоставляет локализованные строки для различных элементов интерфейса, таких как текстовые поля, кнопки и другие элементы управления, в зависимости от выбранного языка. Модели обновляют UI, когда происходит изменение данных или локализации, обеспечивая адаптивность интерфейса.

Модуль AppShell отвечает за управление пользовательским интерфейсом на главной странице приложения. Он настраивает и обновляет UI, включая обработку локализации, отображение информации о пользователе, а также управление навигацией между страницами приложения. Основные функции модуля:

- LoadChatTitles(), загружает список чатов пользователя из базы данных и отображает их на главной странице. Чаты отображаются с учётом локализации и могут быть отфильтрованы по теме;
- OnChatSelected(), обрабатывает выбор чата пользователем, осуществляет переход на страницу чата и сохраняет текущее состояние чата перед навигацией;
- SetTitle(), устанавливает заголовок для главной страницы в зависимости от текущей страницы;
- Logout(), выполняет выход из системы, очищая сохраненные данные и перенаправляя пользователя на страницу входа;
- OnTapped(), обрабатывает нажатие на элемент меню, выполняя действия по возвращению на главную страницу и загрузке данных;
- OnSearchClicked(), выполняет поиск по чату на основе введённого запроса, фильтруя чаты и отображая совпадения;
- OnSearchCompleted(), обрабатывает завершение поиска, обновляя список чатов в соответствии с результатами поиска.

Этот модуль координирует взаимодействие с базой данных, пользовательским интерфейсом и процессами навигации в приложении. Он позволяет динамически обновлять отображаемую информацию и поддерживать актуальность данных на всех страницах.

Модуль SignUpPage отвечает за процесс регистрации нового пользователя в приложении. Он управляет аутентификацией с помощью Firebase, сохранением данных о пользователе в базе данных и предоставляет интерфейс для ввода информации. Основные функции модуля:

- OnSignUpClicked(), обрабатывает клик по кнопке регистрации. Проверяет корректность введенных данных (почта, пароль, подтверждение пароля), выполняет

регистрацию пользователя через Firebase, генерирует случайный никнейм и сохраняет данные о пользователе в базе данных;

- `GenerateRandomUsername()`, генерирует случайный никнейм для пользователя, комбинируя прилагательные и существительные с числовым значением;

- `SaveUserToDatabase()`, сохраняет данные нового пользователя (email и сгенерированный никнейм) в Firebase Realtime Database;

- `OnSignInClicked()`, выполняет переход на страницу входа для пользователя, если тот уже зарегистрирован.

Модуль `SignInPage` отвечает за процесс входа пользователя в приложение с использованием Firebase для аутентификации и получения данных о пользователе. Основные функции модуля:

- `OnSignInClicked()`, обрабатывает клик по кнопке входа, выполняет проверку данных, осуществляет аутентификацию через Firebase и сохраняет токен и email пользователя для дальнейшего использования в системе;

- `FetchAndSaveUserInfo()`, загружает дополнительные данные о пользователе (например, никнейм) из Firebase Realtime Database и сохраняет их с помощью `AuthService`;

- `OnEyeTapped()`, управляет видимостью пароля при вводе (открывает/скрывает пароль);

- `OnForgotPasswordTapped()`, выполняет переход на страницу восстановления пароля;

- `OnSignUpClicked()`, выполняет переход на страницу регистрации для нового пользователя.

Модуль `ResetPasswordPage` отвечает за функциональность восстановления пароля пользователя, включая отправку письма для сброса пароля через Firebase. Основные функции модуля:

- `OnSignInClicked()`, перенаправляет пользователя на страницу входа в приложение;

- `OnResetPasswordClicked()`, выполняет сброс пароля, отправляя пользователю письмо для его восстановления через Firebase.

Модуль `Settings` отвечает за управление настройками языка приложения. Он позволяет пользователю изменять язык интерфейса на один из доступных вариантов, таких как русский, английский, литовский и испанский. Основные функции модуля:

- `OnRussianTapped()`, изменяет язык интерфейса на русский;

- `OnEnglishTapped()`, изменяет язык интерфейса на английский;

- `OnLithuanianTapped()`, изменяет язык интерфейса на литовский;

- `OnSpanishTapped()`, изменяет язык интерфейса на испанский;

- `SetLanguage()`, устанавливает выбранный язык для приложения, обновляет интерфейс и сохраняет предпочтение языка в локальное хранилище с

использованием Preferences, также обновляет видимость галочек для выбранного языка;

- OnDisappearing(), при закрытии страницы обновляет список чатов на главной странице.

Модуль MainPage управляет главной страницей приложения, обеспечивая взаимодействие с пользователем на уровне ввода сообщений и навигации. Основные функции модуля:

- OnAppearing(), метод, который вызывается при появлении страницы, он устанавливает заголовок на главной странице с помощью метода SetShellTitleAsync, передавая текст «SmartTalk AI»;

- SetShellTitleAsync(), асинхронный метод для установки заголовка на верхней панели оболочки (Shell) приложения, этот метод обновляет название с задержкой, чтобы убедиться, что оболочка (Shell) готова к изменениям;

- OnSendMessage(), метод, обрабатывающий отправку сообщения от пользователя, когда пользователь вводит сообщение в текстовое поле и нажимает кнопку отправки, происходит следующее;

- 1) проверяется, что сообщение не пустое;
- 2) очищается поле ввода;
- 3) создается новый пустой чат (список с кортежами sender и message);
- 4) сериализуется история чата в JSON-формат;
- 5) генерируется новый уникальный идентификатор чата;
- 6) снимается фокус с поля ввода;
- 7) выполняется переход на страницу чата с передачей параметров: идентификатора чата, истории сообщений и отправленного сообщения пользователя.

Модуль ChatPage отвечает за отображение и управление диалогом между пользователем и искусственным интеллектом в приложении. Он реализует отображение сообщений, отправку запросов к модели LLaMA 3.3 (через Together.ai API), а также сохранение и загрузку истории чатов из Firebase. Пользователь может начать новый чат или продолжить существующий. Основные функции модуля:

- OnAppearing(), инициализирует загрузку чата при открытии страницы и запускает обработку первого сообщения, если оно задано;

- SendMessage(), отправляет сообщение пользователя, очищает поле ввода и запускает обработку ответа от AI;

- HandleIncomingMessage(), добавляет сообщение пользователя, визуализирует ответ AI и сохраняет диалог в истории;

- GetTogetherAIResponse(), формирует запрос к Together.ai на основе истории сообщений и получает ответ модели;

- AnimateDots(), показывает анимацию «Lama думает...» перед выводом ответа;



- `AnimateResponse()`, поочерёдно выводит символы ответа AI с задержкой, создавая эффект набора текста;
- `TrySaveCurrentChatAsync()`, сохраняет текущую переписку пользователя в Firebase с уникальным ID и темой;
- `LoadChat()`, загружает переписку и тему из Firebase по указанному ID чата;
- `ClearChatState()`, сбрасывает текущее состояние чата, очищает UI и историю сообщений;
- `UpdateChatTitle()`, устанавливает заголовок чата на основе первых слов первого сообщения;
- `AddMessageToChat()`, визуально добавляет сообщение в чат с форматированием и цветовой схемой в зависимости от отправителя.

Разработка мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» продемонстрировала эффективное применение модульного подхода, благодаря которому удалось достичь высокой степени структурированности, логической разделённости и масштабируемости системы.

Каждый программный модуль отвечает за строго определённую функциональность, что упрощает поддержку, расширение и тестирование приложения. Логика работы с авторизацией, хранением и загрузкой данных, локализацией, пользовательским интерфейсом и навигацией была разделена между специализированными модулями, что обеспечивает чистую архитектуру и упрощает взаимодействие между компонентами. Такая организация позволяет легко адаптировать приложение под новые требования, добавлять новые языки, подключать другие базы данных или изменять внешний вид без необходимости вмешательства в основную бизнес-логику.

Интеграция с Firebase, продуманная работа с локализацией и реализация динамического обновления интерфейса сделали приложение гибким и удобным как для разработчиков, так и для конечных пользователей. В результате модульная архитектура обеспечила надёжность, читаемость кода и лёгкость сопровождения, а также создала прочную основу для дальнейшего развития проекта.

## 4 ТЕСТИРОВАНИЕ

Тестирование – это ключевой процесс проверки программного обеспечения или системы, направленный на обеспечение их соответствия требованиям и ожиданиям пользователей. Основная цель тестирования заключается в выявлении ошибок, дефектов и недочетов, что позволяет их исправить и улучшить общее качество продукта. Тестирование проводится на различных этапах жизненного цикла программного обеспечения [10].

Методы тестирования:

- метод белого ящика, тестировщик имеет полное представление о внутренней структуре и логике программы. Это позволяет ему проверять каждый блок кода, включая ветвления и циклы, для обеспечения их корректного выполнения;
- метод черного ящика, в этом подходе тестировщик не имеет информации о внутренней структуре программы, он тестирует программу как единое целое и оценивает ее функциональность, не уделяя внимания внутренним деталям;
- метод серого ящика, сочетает элементы обоих предыдущих методов, тестировщик обладает частичным знанием о внутренней структуре и может проверять определенные блоки кода на корректность выполнения, без глубокого понимания всей логики программы;
- метод функционального тестирования сосредоточен на проверке функциональности программы, тестировщик оценивает, соответствует ли программа заявленным требованиям и правильно ли выполняются ее функции;
- метод юнит-тестирования, проверяет отдельные модули или компоненты программы независимо друг от друга, обеспечивая их корректность и функциональность;
- метод интеграционного тестирования, анализирует взаимодействие между различными модулями программы, чтобы убедиться, что они работают совместно и корректно передают данные;
- метод системного тестирования, программа проверяется как единое целое, включая все ее компоненты, тестировщик удостоверяется в корректной работе программы в различных условиях и при взаимодействии с другими системами.

Тестирование также может включать разнообразные виды тестов, зависящие от целей и требований проекта. Например, тесты на соответствие требованиям проверяют, выполняет ли программа все заданные условия. Тесты на надежность оценивают стабильность и отказоустойчивость системы, тесты на производительность измеряют скорость и эффективность работы, а тесты на безопасность проверяют защиту системы от угроз и атак.

Тестирование является важным этапом разработки программного обеспечения, который помогает выявить и устранить проблемы, а также повысить надежность и качество продукта.

При тестировании мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» применялся ручной метод тестирования. Этот процесс включает выполнение тестовых сценариев вручную и проверку их результатов. Основные этапы и характеристики ручного метода тестирования:

- определение целей и задач, разработка тестовых сценариев и планов тестирования;
- ручное выполнение разработанных тестов в соответствии с планом;
- фиксация результатов выполнения тестов, включая найденные ошибки и непредвиденное поведение системы;
- исправление ошибок и повторное выполнение тестов для проверки исправлений;
- анализ достигнутого покрытия и выявление пробелов в тестировании;
- подготовка отчетов о тестировании, найденных ошибках и рекомендациях по улучшению системы.

Ручное тестирование требует активного участия тестировщика и позволяет выявлять не только технические ошибки, но и проблемы, с которыми могут столкнуться пользователи. Оно обладает гибкостью и адаптивностью к изменениям в требованиях.

Преимущества ручного тестирования включают отсутствие необходимости в сложной автоматизации и возможность использовать интуитивный подход тестировщика. Однако этот метод также имеет недостатки: он зависит от квалификации тестировщика, требует больше времени и ресурсов по сравнению с автоматизированным тестированием и может привести к пропуску ошибок.

Для систематизации тестовых проверок был разработан чек-лист – список критериев и проверок, предназначенных для оценки качества цифрового продукта [11].

Чек-лист был составлен и представлен в таблице 4.1, включая необходимые тесты для оценки различных аспектов программного средства.

Таблица 4.1 – Чек-лист мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»

Checklist ID	Title	Status
1	Авторизация и регистрация пользователя	OK
1.1	Открытие окна регистрации	OK
1.1.1	Ввод корректного email	OK
1.1.2	Ввод валидного пароля	OK
1.1.3	Подтверждение пароля	OK
1.1.4	Переход в главное окно	OK

Продолжение таблицы 4.1

Checklist ID	Title	Status
1.2	Открытие окна входа	OK

1.2.1	Ввод корректных учетных данных	OK
1.3	Восстановление пароля	OK
1.3.1	Переход на вкладку восстановления пароля	OK
1.3.2	Ввод email для восстановления	OK
1.3.3	Получение письма со ссылкой на сброс пароля	OK
2	Главная страница и интерфейс	OK
2.1	Отображение названия приложения	OK
2.2	Наличие иконки меню	OK
2.3	Отображение изображения робота	OK
2.4	Отображение текстового поля ввода сообщений	OK
2.5	Нажатие на кнопку отправки сообщений	OK
3	Чат с искусственным интеллектом	OK
3.1	Ввод текста и отправка	OK
3.1.1	Очистка поля после отправки	OK
3.1.2	Отображение сообщения пользователя	OK
3.2	Отображение ответа от ИИ	OK
3.2.1	Отображение анимация ожидания («AI думает...»)	OK
3.2.2	Отображение эффекта печати по символам	OK
3.2.3	Автопрокрутка вниз при длинных ответах	OK
4	Работа с историей чатов (боковая панель)	OK
4.1	Открытие боковой панели	OK
4.2	Отображение списка сохраненных диалогов	OK
4.3	Переход к выбранному чату	OK
4.4	Продолжение беседы из истории	OK
5	Сохранение и загрузка диалогов	OK
5.1	Генерация темы по первым сообщениям	OK
5.2	Загрузка диалога при открытии существующего чата	OK
6	Работа со сменой языка	OK
6.1	Переход в настройки	OK
6.2	Выбор нового языка	OK
6.3	Применение выбранного языка в интерфейсе	OK
7	Работа с кнопкой выхода из аккаунта	OK
7.1	Переход на боковую панель	OK
7.2	Нажатие на кнопку «Выход из аккаунта»	OK
7.3	Проверка выхода и возврата на экран авторизации	OK

Тестирование должно начинаться сразу после создания новых модулей и систематически совершенствоваться в течение всего процесса разработки. Все тесты должны быть выполнены, используя встроенные инструменты разработки и структуры проекта.

Тест-кейс представляет собой комплекс тестовых данных, условий выполнения и предполагаемых результатов, разработанный с определенной целью, такой как проверка соответствия конкретному требованию [12].

Метод «черного ящика» был использован при проверке разработанного программного продукта. Результаты тестирования представлены в таблице 4.2.

Таблица 4.2 – Тест-кейсы мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»

Чек-лист ID	Модуль тестируемой операции	Тестируемая операция	Шаги	Ожидаемый результат	Статус
1	2	3	4	5	6
1.1	Регистрация пользователя	Открытие окна регистрации	1. Открыть приложение. 2. Нажать на кнопку «Зарегистрироваться». 3. Подождать загрузки окна	Открылось окно с полями Email, Пароль, Подтверждение пароля	ОК
1.2	Регистрация пользователя	Ввод валидных данных	1. Ввести email в формате user@example.com 2. Ввести пароль от 8 символов. 3. Повторно ввести пароль. 4. Нажать «Зарегистрироваться».	Пользователь переходит в главное окно приложения	ОК
1.3	Регистрация пользователя	Проверка сохранения сессии	1. Пройти регистрацию. 2. Перезайти в приложение. 3. Открыть вкладку «Главная».	Пользователь остаётся в системе, не требуется повторный вход	ОК
2.1	Авторизация	Ввод корректных учетных данных	1. Нажать «Вход». 2. Ввести email и пароль. 3. Нажать "Войти". 4. Переключиться на другую вкладку и обратно.	Авторизация выполнена, пользователь возвращается в основное окно.	ОК

Продолжение таблицы 4.2

1	2	3	4	5	6
---	---	---	---	---	---

3.1	Восстановление пароля	Процесс восстановления	1. Нажать "Забыли пароль?". 2. Ввести email. 3. Нажать «Отправить». 4. Открыть почту и перейти по ссылке из письма.	Пользователь перенаправлен на форму сброса пароля.	ОК
4.1	Главная страница	Отображение интерфейса	1. Авторизоваться. 2. Перейти на главную. 3. Перезагрузить страницу.	Отображаются название приложения, меню, робот, поле ввода и кнопка отправки	ОК
5.1	Чат	Отправка сообщения	1. Ввести «Привет» в текстовое поле. 2. Нажать кнопку «Отправить». 3. Переключиться на другой чат и вернуться.	Сообщение отображается, поле очищено	ОК
5.2	Чат	Ответ ИИ и прокрутка	1. Отправить длинный вопрос. 2. Дождаться ответа. 3. Не прокручивать вручную.	Ответ прокручивается вниз автоматически, с эффектом печати	BUG
6.1	История чатов	Работа с боковой панелью	1. Нажать на иконку меню. 2. Выбрать сохранённый чат. 3. Ввести новое сообщение.	Продолжение беседы в выбранном чате, чат активен	ОК
7.1	Смена языка	Изменение языка интерфейса	1. Перейти в «Настройки». 2. Выбрать «English». 3. Подтвердить. 4. Вернуться на главную.	Интерфейс отображается на выбранном языке	ОК
8.1	Выход из аккаунта	Завершение сессии	1. Открыть меню. 2. Нажать «Выйти». 3. Перезагрузить страницу. 4. Попробовать вернуться назад.	Пользователь перенаправлен на экран авторизации	ОК

В процессе тестирования было проверено множество функциональных сценариев, охватывающих регистрацию, авторизацию, восстановление пароля,

отображение интерфейса, чат с искусственным интеллектом, работу с историей диалогов, смену языка и выход из аккаунта. Почти все тестируемые функции продемонстрировали стабильную и предсказуемую работу. Ошибка с прокруткой и ответом от ИИ была исправлена. Окна регистрации и входа корректно открываются, поля работают согласно требованиям, и при вводе валидных данных пользователь успешно проходит аутентификацию. Механизм восстановления пароля функционирует должным образом: ссылка из письма работает, форма сброса открывается.

Интерфейс главной страницы корректно отображает все ключевые элементы – заголовок, меню, иконку робота, текстовое поле и кнопку отправки сообщений. Поведение чата соответствует ожиданиям: пользовательские сообщения отправляются, отображаются, очищаются из поля ввода, а ответы от ИИ сопровождаются анимацией и печатью текста, с автоматической прокруткой. История чатов сохраняется, доступна через боковую панель, и позволяет пользователю возвращаться к прежним диалогам и продолжать беседу.

Также была подтверждена корректная работа системы смены языка: выбранный язык применяется к интерфейсу после изменения настроек. Функция выхода из аккаунта завершает сессию, возвращает пользователя на экран авторизации и предотвращает несанкционированный доступ при попытке возврата назад.

В результате тестирования не было выявлено серьезных проблем, нарушающих логику или работоспособность системы. Все функциональные требования были успешно реализованы и протестированы. Приложение продемонстрировало высокую стабильность и готово к эксплуатации в рабочем окружении.

## **5 ПРИМЕНЕНИЕ**

### **5.1 Назначение и условия применения**

Мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» предоставляет пользователям возможность взаимодействовать с искусственным интеллектом для генерации диалогов и получения ответов на вопросы. Приложение разработано с учетом удобства использования и доступности для пользователей всех возрастов.

Условия применения мобильного приложения:

- для использования приложения необходимо мобильное устройство на базе операционной системы Android или iOS;
- рекомендуется пользователям старше 12 лет, так как некоторые функции могут требовать базовых знаний о взаимодействии с ИИ;
- вся информация, предоставленная в приложении, является исключительно ознакомительной и не должна восприниматься как профессиональные советы.

Процесс установки и запуска мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»:

- загрузите приложение из официального магазина приложений Google Play (для устройств на базе Android) или App Store (для устройств на базе iOS);
- найдите загруженное приложение в списке установленных приложений и нажмите на него, чтобы запустить (для устройств на базе Android);
- после завершения загрузки приложения найдите его значок на главном экране вашего устройства и нажмите на него, чтобы запустить (для устройств на базе iOS);
- для удобного доступа вы можете создать ярлык на главном экране вашего устройства.

После выполнения этих шагов пользователь сможет запускать мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» через его значок на главном экране своего мобильного устройства.

### **5.2 Руководство пользователя**

Руководство пользователя – это документ, который предлагает информацию и инструкции по правильному использованию программного обеспечения, устройства или системы. В нем содержится детальное описание функций, возможностей и настроек продукта, а также пошаговые указания по его установке, настройке и эксплуатации.

Такое руководство помогает пользователям ознакомиться с продуктом, понять его основные принципы работы и эффективно использовать его функционал для достижения поставленных целей.



Сразу после установки и запуска приложения пользователю предлагается зарегистрироваться. Страница регистрации предоставляет поля для ввода электронной почты, пароля и его подтверждения, а также реализует проверку корректности введенных данных, рисунок Б.1. В случае успешной регистрации для пользователя автоматически создается уникальный идентификатор и никнейм, после чего происходит сохранение данных в базе и вход в систему. Этот этап обеспечивает защиту данных и персонализацию работы с приложением.

Если у пользователя уже есть аккаунт, он может перейти на экран авторизации, где потребуется ввести электронную почту и пароль для входа, рисунок Б.2. Авторизация необходима для доступа к ранее сохраненной истории чатов и настройкам. Интерфейс лаконичен и понятен, а также содержит ссылки на восстановление пароля и регистрацию нового аккаунта. При необходимости пользователь может восстановить пароль, воспользовавшись соответствующей функцией: в этом случае на указанную почту будет отправлено письмо со ссылкой для сброса пароля, рисунок Б.3.

После успешного входа пользователь попадает на главную страницу приложения, где можно сразу начать новый диалог с ИИ или воспользоваться боковой панелью для навигации, рисунок Б.4. Центральная часть экрана отведена под поле ввода сообщения, а также список предыдущих диалогов. Интерфейс поддерживает быструю отправку сообщений и мгновенный переход к диалогу без лишних действий.

Страница чата служит основным рабочим пространством приложения, где пользователь ведёт переписку с искусственным интеллектом, рисунок Б.5. Интерфейс включает визуальное оформление сообщений, разделение по отправителю, а также динамическую анимацию, имитирующую процесс набора текста ИИ-собеседником. Также реализованы функции сохранения истории, генерации заголовка чата и продолжения диалога в любой момент.

Слева доступна боковая панель, открывающая дополнительные возможности навигации, рисунок Б.6. Пользователь может быстро перейти к нужному чату по ключевым словам, просмотреть историю переписок, вернуться на главный экран, изменить настройки или выйти из системы. Панель упрощает управление контентом и предоставляет удобный доступ к ключевым функциям без необходимости покидать активную сессию.

В разделе настроек пользователь может выбрать язык интерфейса из доступных вариантов: русский, английский, литовский и испанский, рисунок Б.7. Язык можно сменить в любой момент – приложение мгновенно адаптирует весь интерфейс без необходимости перезапуска. Настройки сохраняются локально и автоматически подгружаются при следующем входе, обеспечивая стабильный пользовательский опыт.

## ЗАКЛЮЧЕНИЕ

В ходе курсового проектирования была успешно реализована функциональная и удобная платформа, предназначенная для эффективного взаимодействия пользователей с искусственным интеллектом. Практическая значимость данной разработки заключается в создании инструмента, который помогает не только развивать навыки общения и повышать продуктивность, но и предлагает инновационные способы использования ИИ для решения разнообразных задач. Приложение представляет собой универсальный инструмент, который можно использовать для обучения, тренировки и создания новых идей в контексте общения с виртуальными собеседниками.

Разработанное приложение имеет ряд преимуществ, таких как кроссплатформенная доступность, интуитивно понятный интерфейс и возможность настройки параметров взаимодействия с ИИ. Весь процесс разработки приложения позволил углубить знания в области ИИ и диалоговых систем, а также обеспечить удобство использования для конечного пользователя. Все функциональные требования были выполнены, включая авторизацию, историю чатов, переключение языков и другие ключевые возможности.

Перспективы развития мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI» видятся в нескольких направлениях:

- внедрение более сложных и многоуровневых алгоритмов ИИ, что позволит создавать более реалистичные и контекстуально обоснованные диалоги. Можно интегрировать модели машинного обучения для адаптивного улучшения ответов ИИ в зависимости от предпочтений пользователя и контекста диалога;
- разработка функционала, который позволит ИИ «запоминать» стиль общения и предпочтения каждого пользователя, что обеспечит персонализированный подход и более глубокие и осмысленные взаимодействия. ИИ сможет не только отвечать на вопросы, но и помогать развивать определённые навыки в зависимости от интересов пользователя;
- возможность интеграции с внешними сервисами и платформами, такими как социальные сети, системы планирования задач, и даже инструменты для создания контента. Это расширит круг пользователей и сделает приложение более многозадачным и гибким;
- включение более широкого спектра тем и контекстов для общения, например, специализированные диалоги для профессиональных сфер, психологической помощи или обучения новым навыкам. Это откроет новые ниши для пользователей и добавит больше вариативности в общение с ИИ;
- добавление функции голосового ввода и голосовых ответов ИИ, что значительно улучшит пользовательский опыт и сделает приложение более

доступным для людей с ограниченными возможностями или тех, кто предпочитает голосовые интерфейсы;

- применение более продвинутых технологий в области обработки естественного языка (NLP), таких как нейросетевые модели с глубоким обучением, для повышения качества диалогов и более точного понимания контекста;

- внедрение возможностей для создания и обмена диалогами между пользователями. Это позволит создать сообщество, где люди смогут делиться своим опытом взаимодействия с ИИ, создавать совместные сценарии общения и улучшать навыки через коллективный обмен;

- разработка инструментов для отслеживания прогресса пользователя в его навыках общения с ИИ. Приложение может собирать данные о частоте использования, предпочтениях в темах и стиле общения, а также предоставлять отчёты и рекомендации для улучшения опыта взаимодействия.

Цель проекта была успешно выполнена, и все задачи были достигнуты. Приложение стало не только функциональным инструментом, но и важным шагом в направлении улучшения взаимодействия между человеком и ИИ. В будущем мобильное приложение «Генератор диалогов с ИИ с использованием .NET MAUI и AI» может значительно расширить свои возможности, предоставляя пользователям ещё более персонализированные, удобные и эффективные способы общения с искусственным интеллектом.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Соколова, В. Разработка мобильных приложений: учебное пособие для среднего профессионального образования / В. Соколова. – М: Юрайт, 2024. – 234 с.
- [2] Типы мобильных приложений [Электронный ресурс]. – Режим доступа: <https://polygant.net/ru/blog/chto-takoe-mobilnoe-prilozhenie/>. – Дата доступа: 10.03.2025.
- [3] Входные данные и выходные данные [Электронный ресурс]. – Режим доступа: <https://studwood.net/1706755/informatika/>. – Дата доступа: 13.03.2025.
- [4] Диаграмма Деятельности [Электронный ресурс]. – Режим доступа: <https://eumk.mrk-bsuir.by/eumk>. – Дата доступа: 14.03.2025.
- [5] Диаграмма Вариантов использования [Электронный ресурс]. – Режим доступа: <https://eumk.mrk-bsuir.by/eumk>. – Дата доступа: 15.03.2025.
- [6] Васильев, А. Программирование на C# для начинающих. Основные сведения / А. Васильев. – М.: Бомбора, 2023. – 142 с.
- [7] Amann, S. A Study of Visual Studio Usage in Practice / S. Amann, S. Nadi, M. Mezini. – Hachette Book Group / FR, 2023. – 47 p.
- [8] Fabio, S. Designing and Prototyping Interfaces with Figma / S. Fabio. – Harper Collins / NY, 2022. – 228 p.
- [9] Умрихин, Е. Разработка Android-приложений на C# с использованием Xamarin с нуля / Е. Умрихин. – М: БХВ, 2022. – 24 с.
- [10] Блэк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование / Р. Блэк. – Спб.: Питер, 2022. – 800 с.
- [11] Чек-лист: что это такое, как составлять – [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/723948/> – Дата доступа: 14.03.2025
- [12] Тест-кейс: задачи, правила создания – [Электронный ресурс]. – Режим доступа: <https://software-testing.ru/library/testing/testing-for-beginners/1991-test-case-writing> – Дата доступа: 16.03.2025.

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Текст программы

```

using Firebase.Database;
using System.Collections.ObjectModel;
using Firebase.Database;
using Firebase.Database.Query;
using System.Diagnostics;
using static System.Net.Mime.MediaTypeNames;
using Newtonsoft.Json;
namespace RealTalk_AI
{
    public partial class AppShell : Shell
    {
        private ViewModels.AppShellViewModel ViewModel =>
BindingContext as ViewModels.AppShellViewModel;
        private ObservableCollection<string> _chatTitles;
        private static readonly FirebaseClient _databaseClient =
new FirebaseClient("https://realtalk-ai-default-
rtdb.firebaseio.com/");
        private Dictionary<string, string> chatTitlesDict;
        private bool isNavigating;
        public ObservableCollection<string> ChatTitles
        {
            get => _chatTitles;
            set
            {
                _chatTitles = value;
                OnPropertyChanged(nameof(ChatTitles));
            }
        }
        public AppShell()
        {
            InitializeComponent();
            BindingContext = new
ViewModels.AppShellViewModel();
            ChatTitles = new ObservableCollection<string>();
            LoadChatTitles();
            TitleMain.Text = "SmartTalk AI";
            Username.Text = AuthService.GetUsername() ??
"Имя";
            PageTitle = ViewModel.Home;
            Routing.RegisterRoute("ChatPage", typeof(ChatPage));
            Routing.RegisterRoute(nameof(MainPage),
typeof(MainPage));
            Routing.RegisterRoute(nameof(Settings),
typeof(Settings));
        }
        private string _pageTitle;
        public string PageTitle
        {
            get => _pageTitle;
            set
            {
                _pageTitle = value;
                OnPropertyChanged(nameof(PageTitle));
            }
        }
        public async void LoadChatTitles()
        {
            try

```

```

        {
            var userEmail =
AuthService.GetUserEmail().Replace("@", "_").Replace(".",
"_");
            var userChatsRef =
_databaseClient.Child("users").Child(userEmail).Child("chats"
);
            var chats = await
userChatsRef.OnceAsync<dynamic>();
            if (chats == null || !chats.Any())
            {
                Debug.WriteLine("No chats found.");
                return;
            }
            chatTitlesDict = new Dictionary<string, string>();
            foreach (var chat in chats)
            {
                Debug.WriteLine($"Processing chat:
{chat.Key}");
                dynamic chatData = chat.Object;
                if (chatData != null && chatData.Topic != null)
                {
                    chatTitlesDict[chat.Key] =
$" {ViewModel.ChatTopic} " + (string)chatData.Topic;
                    Debug.WriteLine($"Added title: {chat.Key} -
{chatData.Topic}");
                }
                else
                {
                    Debug.WriteLine("No Topic found in the
chat.");
                }
            }
            ChatTitlesCollectionView.ItemsSource =
chatTitlesDict.Select(kvp => kvp.Value).ToList();
            Debug.WriteLine($"Loaded chat titles: {string.Join(",
", chatTitlesDict.Select(kvp => kvp.Value))}");
        }
        catch (Exception ex)
        {
            Debug.WriteLine($"Error loading chat titles:
{ex.Message}");
        }
    }
    private async void OnChatSelected(object sender,
EventArgs e)
    {
        if (isNavigating) return;
        if (sender is Label label && label.BindingContext is
string selectedTopic)
        {
            var chatId = chatTitlesDict.FirstOrDefault(kvp =>
kvp.Value == selectedTopic).Key;
            if (!string.IsNullOrEmpty(chatId))
            {
                try
                {
                    isNavigating = true;

```

```

        var currentChatPage =
Shell.Current.CurrentPage as ChatPage;
        if (currentChatPage != null)
        {
            Debug.WriteLine("[AppShell] Saving current
chat before navigation.");
            await
currentChatPage.TrySaveCurrentChatAsync();
        }
        await
Shell.Current.GoToAsync($"//ChatPage?chatId={Uri.EscapeD
ataString(chatId)}");
        Shell.Current.FlyoutIsPresented = false;
    }
    catch (Exception ex)
    {
        Debug.WriteLine($"[ERROR] Navigation error:
{ex.Message}");
    }
    finally
    {
        {
            isNavigating = false;
            LoadChatTitles();
        }
    }
}
}
public void SetTitle(string title)
{
    PageTitle = title;
    TitleMain.Text = PageTitle;
}
private async void Logout(object sender, EventArgs e)
{
    AuthService.Logout();
Microsoft.Maui.Controls.Application.Current.MainPage = new
SignInPage();
    if (Shell.Current.CurrentPage is ChatPage chatPage)
    {
        await chatPage.TrySaveCurrentChatAsync();
    }
}
private async void OnTapped(object sender, EventArgs e)
{
    if (Shell.Current.CurrentPage is ChatPage chatPage)
    {
        await chatPage.TrySaveCurrentChatAsync();
        chatPage.ClearChatState();
        chatPage.ChatId = null;
    }
    await Shell.Current.GoToAsync("//MainPage");
    await Task.Delay(100);
    SetTitle("RealTalk AI");
    LoadChatTitles();
    Shell.Current.FlyoutIsPresented = false;
}
private async void OnTappedSettings(object sender,
EventArgs e)
{
    await Shell.Current.GoToAsync(nameof(Settings));
    Shell.Current.FlyoutIsPresented = false;
}
}

```

```

private async void OnSearchClicked(object sender,
EventArgs e)
{
    {
        string query = InputEntry.Text?.Trim();
        if (string.IsNullOrEmpty(query))
        {
            LoadChatTitles(); // Показать последние чаты
            return;
        }
        var userEmail =
AuthService.GetUserEmail().Replace("@", "_").Replace(".",
"_");
        var chatsRef =
_databaseClient.Child("users").Child(userEmail).Child("chats"
);
        try
        {
            ChatTitles.Clear();
            if (string.IsNullOrEmpty(query))
            {
                LoadChatTitles();
                return;
            }
            var userChatsRef =
_databaseClient.Child("users").Child(userEmail).Child("chats"
);
            var chats = await
userChatsRef.OnceAsync<dynamic>();
            var matchedTitles = new Dictionary<string, string>();
            string loweredQuery = query.ToLowerInvariant();
            foreach (var chat in chats)
            {
                string chatId = chat.Key;
                dynamic chatData = chat.Object;
                if (chatData?.Messages == null)
                    continue;
                foreach (var message in chatData.Messages)
                {
                    string messageText = message?.message;
                    if (!string.IsNullOrEmpty(messageText) &&
messageText.ToLowerInvariant().Contains(loweredQuery))
                    {
                        string topic = chatData?.Topic ?? "Без темы";
                        matchedTitles[chatId] =
$"{{ViewModel.ChatTopic}} {topic}";
                        break; // Найдено совпадение — дальше в
этом чате не ищем
                    }
                }
            }
            if (matchedTitles.Any())
            {
                chatTitlesDict = matchedTitles;
ChatTitlesCollectionView.ItemsSource =
matchedTitles.Values.ToList();
            }
            else
            {
                ChatTitlesCollectionView.ItemsSource = new List<string> {
"Чаты не найдены" };
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        Debug.WriteLine($"[Search Error] {ex.Message}");
    }
}

private async void OnSearchCompleted(object sender,
EventArgs e)
{
    string query = InputEntry.Text?.Trim();

    if (string.IsNullOrEmpty(query))
    {
        LoadChatTitles();
        return;
    }

    var userEmail =
AuthService.GetUserEmail().Replace("@", "_").Replace(".",
"_");

    var chatsRef =
_databaseClient.Child("users").Child(userEmail).Child("chats"
);

    try
    {
        ChatTitles.Clear();
        var userChatsRef =
_databaseClient.Child("users").Child(userEmail).Child("chats"
);

        var chats = await
userChatsRef.OnceAsync<dynamic>();

        var matchedTitles = new Dictionary<string, string>();
        string loweredQuery = query.ToLowerInvariant();
        foreach (var chat in chats)
        {
            string chatId = chat.Key;
            dynamic chatData = chat.Object;
            if (chatData?.Messages == null)
                continue;
            foreach (var message in chatData.Messages)
            {
                string messageText = message?.message;
                if (!string.IsNullOrEmpty(messageText) &&
messageText.ToLowerInvariant().Contains(loweredQuery))
                {
                    string topic = chatData?.Topic ?? "Без темы";
                    matchedTitles[chatId] =
${ViewModel.ChatTopic} {topic}";
                    break;
                }
            }
        }
        if (matchedTitles.Any())
        {
            chatTitlesDict = matchedTitles;
            ChatTitlesCollectionView.ItemsSource =
matchedTitles.Values.ToList();
        }
        else
        {
            ChatTitlesCollectionView.ItemsSource = new List<string> {
"Чаты не найдены" };
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        Debug.WriteLine($"[Search Error] {ex.Message}");
    }
}

}

using Firebase.Auth;
using Firebase.Auth.Providers;
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Threading.Tasks;

namespace RealTalk_AI
{
    public partial class SignUpPage : ContentPage
    {
        private readonly FirebaseAuthClient _authClient;
        private readonly FirebaseClient _databaseClient;
        private bool firstIsPasswordVisible = false;
        private bool isPasswordVisible = false;
        public SignUpPage()
        {
            InitializeComponent();
            BindingContext = new
ViewModels.SignUpViewModel();
            NavigationPage.SetHasNavigationBar(this, false);
            _authClient = new FirebaseAuthClient(new
FirebaseAuthConfig
            {
                ApiKey = "AlzaSyCdTJg_iSeWoX2Ete3-
8emKdqBVnY71AIA",
                AuthDomain = "realtalk-ai.firebaseio.com",
                Providers = new FirebaseAuthProvider[] { new
EmailProvider() }
            });
            _databaseClient = new FirebaseClient("https://realtalk-
ai-default-rtdb.firebaseio.com/");
        }

        private async void OnSignUpClicked(object sender,
EventArgs e)
        {
            string email = emailEntry.Text;
            string password = passwordEntry.Text;
            string confirmPassword = confirmPasswordEntry.Text;
            if (string.IsNullOrEmpty(email) ||
string.IsNullOrEmpty(password))
            {
                await DisplayAlert("Ошибка", "Введите почту и
пароль!", "OK");
                return;
            }
            if (password != confirmPassword)
            {
                await DisplayAlert("Ошибка", "Пароли не
совпадают!", "OK");
                return;
            }
            try
            {

```

```

        var userCredential = await
_authClient.CreateUserWithEmailAndPasswordAsync(email,
password);
        string randomUsername =
GenerateRandomUsername();
        await SaveUserToDatabase(email,
randomUsername);
        await DisplayAlert("Успех", "Аккаунт создан!",
"OK");
        await Navigation.PushAsync(new SignInPage());
    }
    catch (FirebaseAuthException ex)
    {
        await DisplayAlert("Ошибка регистрации",
ex.Reason.ToString(), "OK");
    }
    catch (Exception ex)
    {
        await DisplayAlert("Ошибка", ex.Message, "OK");
    }
}
private string GenerateRandomUsername()
{
    string[] adjectives = { "Fast", "Happy", "Smart",
"Brave", "Cool", "Funny", "Lucky", "Clever" };
    string[] nouns = { "Tiger", "Panda", "Eagle", "Wolf",
"Dolphin", "Fox", "Hawk", "Lion" };
    Random rnd = new Random();
    return
$" {adjectives[rnd.Next(adjectives.Length)]} {nouns[rnd.Next(n
ouns.Length)]} {rnd.Next(1000, 9999)}";
}
private async Task SaveUserToDatabase(string email,
string username)
{
    string encodedEmail = email.Replace("@",
" _").Replace(".", " _");
    var user = new
    {
        Email = email,
        Username = username,
    };
    await _databaseClient
        .Child("users")
        .Child(encodedEmail)
        .PutAsync(user);
}
private void OnFirstEyeTapped(object sender, EventArgs
e)
{
    passwordEntry.IsPassword = !firstIsPasswordVisible;
    ((Image)sender).Source = !firstIsPasswordVisible ?
"eye_close.png" : "eye_open.svg";
    firstIsPasswordVisible = !firstIsPasswordVisible;
}
private void OnEyeTapped(object sender, EventArgs e)
{
    confirmPasswordEntry.IsPassword =
!isPasswordVisible;

```

```

        ((Image)sender).Source = isPasswordVisible ?
"eye_close.png" : "eye_open.svg";
        isPasswordVisible = !isPasswordVisible;
    }
    private async void OnSignInClicked(object sender,
EventArgs e)
    {
        await Navigation.PushAsync(new SignInPage());
    }
}
using Firebase.Auth;
using Firebase.Auth.Providers;
using Firebase.Auth.Repository;
using Firebase.Database;
using Firebase.Database.Query;
using Microsoft.Maui.Controls;
using System;
using System.Threading.Tasks;
using System.Text.Json;
namespace RealTalk_AI
{
    public partial class SignInPage : ContentPage
    {
        private FirebaseAuthClient authClient;
        private bool isPasswordVisible = false;
        public SignInPage()
        {
            InitializeComponent();
            BindingContext = new
ViewModels.SignInViewModel();
            NavigationPage.SetHasNavigationBar(this, false);
            NavigationPage.SetHasBackButton(this, false);
            var config = new FirebaseAuthConfig
            {
                ApiKey = "AIzaSyCdTJg_iSeWoX2Ete3-
8emKdqBVnY71AIA",
                AuthDomain = "realtalk-ai.firebaseio.com",
                Providers = new FirebaseAuthProvider[] { new
EmailProvider() },
                UserRepository = new
FileUserRepository("RealTalk_AI")
            };
            authClient = new FirebaseAuthClient(config);
        }
        private async void OnSignInClicked(object sender,
EventArgs e)
        {
            string email = emailEntry.Text;
            string password = passwordEntry.Text;
            if (string.IsNullOrEmpty(email) ||
string.IsNullOrEmpty(password))
            {
                await DisplayAlert("Ошибка", "Введите почту и
пароль!", "OK");
                return;
            }
            try
            {

```



```

        var authResult = await
authClient.SignInWithEmailAndPasswordAsync(email,
password);
        var user = authResult.User;
        string idToken = await user.GetIdTokenAsync();
        AuthService.SetAuthToken(idToken);
        AuthService.SetUserEmail(user.Info.Email);
        await FetchAndSaveUserInfo(user.Info.Email);
        await DisplayAlert("Успех", $"Добро пожаловать,
{user.Info.Email}", "OK");

        Application.Current.MainPage = new
NavigationPage(new AppShell());
    }
    catch (FirebaseAuthException ex)
    {
        await DisplayAlert("Ошибка входа",
ex.Reason.ToString(), "OK");
    }
}
private async Task FetchAndSaveUserInfo(string email)
{
    try
    {
        var httpClient = new HttpClient();
        string encodedEmail = email.Replace("@",
" _").Replace(".", " _");
        var response = await
httpClient.GetAsync($"https://realtalk-ai-default-
rtbd.firebaseio.com/users/{encodedEmail}.json");

        if (response.IsSuccessStatusCode)
        {
            var json = await
response.Content.ReadAsStringAsync();
            var userInfo =
JsonSerializer.Deserialize<UserInfo>(json);
            if (userInfo != null)
            {
                AuthService.SetUsername(userInfo.Username);
                Console.WriteLine($"User's username is:
{userInfo.Username}");
            }
            else
            {
                await DisplayAlert("Ошибка", "Информация о
пользователе не найдена.", "OK");
            }
        }
        else
        {
            await DisplayAlert("Ошибка", "Не удалось
загрузить данные из Firebase.", "OK");
        }
    }
    catch (FirebaseException firebaseEx)
    {
        await DisplayAlert("Ошибка", $"Ошибка загрузки
данных из Firebase: {firebaseEx.Message}", "OK");
    }
    catch (Exception ex)
    {

```

```

        await DisplayAlert("Ошибка", $"Ошибка загрузки
данных: {ex.Message}", "OK");
    }
}
private void OnEyeTapped(object sender, EventArgs e)
{
    if (isPasswordVisible)
    {
        passwordEntry.IsPassword = true;
        ((Image)sender).Source = "eye_close.png";
    }
    else
    {
        passwordEntry.IsPassword = false;
        ((Image)sender).Source = "eye_open.svg";
    }
    isPasswordVisible = !isPasswordVisible;
}
private async void OnForgotPasswordTapped(object
sender, EventArgs e)
{
    await Navigation.PushAsync(new
ResetPasswordPage());
}
private async void OnSignUpClicked(object sender,
EventArgs e)
{
    await Navigation.PushAsync(new SignUpPage());
}
}
public class UserInfo
{
    public string Id { get; set; }
    public string Username { get; set; }
}
using Microsoft.Maui.Controls;
using Newtonsoft.Json;
using RealTalk_AI.ViewModels;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading.Tasks;
namespace RealTalk_AI
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            BindingContext = new MainPageViewModel();
            NavigationPage.SetIconColor(this,
Color.FromHex("305FD5"));
        }
        protected override async void OnAppearing()
        {
            base.OnAppearing();
            await SetShellTitleAsync("SmartTalk AI");
        }
        private async Task SetShellTitleAsync(string title)
        {
            await Task.Delay(100);
            if (Shell.Current is AppShell appShell)

```

```

    {
        appShell.SetTitle(title);
    }
    else
    {
        Debug.WriteLine("AppShell is null");
    }
}
private async void OnSendMessage(object sender,
EventArgs e)
{
    string userMessage = InputEntry.Text;
    if (string.IsNullOrEmpty(userMessage)) return;
    InputEntry.Text = string.Empty;
    var emptyChat = new List<(string sender, string
message)>();
    string serializedChatHistory =
JsonConvert.SerializeObject(emptyChat);
    string newChatId = Guid.NewGuid().ToString();

    InputEntry.Unfocus();
    /
    await
Shell.Current.GoToAsync($"//ChatPage?chatId={Uri.EscapeD
ataString(newChatId)}&chatHistory={Uri.EscapeDataString(se
rializedChatHistory)}&userMessage={Uri.EscapeDataString(u
serMessage)}");
}
}
}
using System.Text;
using Newtonsoft.Json;
using Firebase.Database;
using Firebase.Database.Query;
using System.Diagnostics;
namespace RealTalk_AI
{
    [QueryProperty(nameof(ChatHistory), "chatHistory")]
    [QueryProperty(nameof(UserMessage), "userMessage")]
    [QueryProperty(nameof(ChatId), "chatId")]
    public partial class ChatPage : ContentPage,
IQueryAttributable
    {
        private ViewModels.ChatPageViewModel ViewModel =>
BindingContext as ViewModels.ChatPageViewModel;
        private const string ApiKey =
"tgp_v1_yhnsDXnowgpVXFZPXauX1 1R2zGYOvwCF4EIDO
PDP3To";
        private const string ApiUrl =
"https://api.together.ai/v1/chat/completions";
        private const string DefaultChatTitle = "Новый чат";
        private CancellationTokenSource
cancellationTokenSource;
        private bool isSending;
        private string chatTitle = DefaultChatTitle;
        private string _chatId;
        private string firstMessage;
        private List<(string sender, string message)> chatHistory
= new();
        public string ChatTitle
        {
            get => chatTitle;

```

```

            set { chatTitle = value; OnPropertyChanged(); }
        }
        public string ChatId
        {
            get => _chatId;
            set
            {
                if (!string.IsNullOrEmpty(value))
                {
                    _chatId = Uri.UnescapeDataString(value);
                    LoadChat(_chatId);
                }
            }
        }
        public string ChatHistory
        {
            get => JsonConvert.SerializeObject(chatHistory);
            set
            {
                if (!string.IsNullOrEmpty(value))
                {
                    chatHistory =
JsonConvert.DeserializeObject<List<(string sender, string
message)>>(>
                    Uri.UnescapeDataString(value)) ?? new();
                }
                OnPropertyChanged();
            }
        }
        public string UserMessage
        {
            get => firstMessage;
            set { firstMessage = value; OnPropertyChanged(); }
        }
        public ChatPage()
        {
            InitializeComponent();
            BindingContext = new
ViewModels.ChatPageViewModel();
        }
        public void ApplyQueryAttributes(IDictionary<string,
object> query)
        {
            if (query.TryGetValue("chatId", out var chatIdObj) &&
chatIdObj is string chatIdStr)
            {
                ChatId = Uri.UnescapeDataString(chatIdStr);
                Debug.WriteLine($"[ApplyQueryAttributes] chatId =
{ChatId}");
                LoadChat(ChatId);
            }
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
            if (string.IsNullOrEmpty(ChatId))
            {
                ClearChatState();
            }
            else if (!chatHistory.Any())
            {
                LoadChat(ChatId);
            }
        }
    }
}

```

```

    }
    if (!string.IsNullOrEmpty(firstMessage))
    {
        cancellationTokenSource = new();
        HandleIncomingMessage(firstMessage,
cancellationTokenSource.Token);
    }
    UpdateChatTitle();
}
protected override async void OnDisappearing()
{
    base.OnDisappearing();
    if (await TrySaveCurrentChatAsync())
    {
        ClearChatState();
        if (Shell.Current is AppShell appShell)
        {
            appShell.LoadChatTitles();
            appShell.SetTitle("RealTalk AI");
        }
    }
}
public async Task<bool> TrySaveCurrentChatAsync()
{
    if (!chatHistory.Any()) return false;
    string userEmail = AuthService.GetUserEmail();
    string userId = AuthService.GetUserId();
    string nickname = AuthService.GetUsername();
    string topic =
GetChatTopic(chatHistory.First().message);
    string effectiveChatId = ChatId;
    if (string.IsNullOrEmpty(effectiveChatId))
    {
        effectiveChatId = Guid.NewGuid().ToString();
        ChatId = effectiveChatId;
    }
    await
FirebaseService.SaveChatHistoryAsync(userEmail, nickname,
chatHistory, topic, chatId: effectiveChatId);
    Debug.WriteLine($"? Чат сохранён под ID:
{effectiveChatId}");
    return true;
}
public void ClearChatState()
{
    chatHistory.Clear();
    ChatLayout.Children.Clear();
    ChatTitle = DefaultChatTitle;
    InputEntry.Text = string.Empty;
((AppShell)Shell.Current)?.SetTitle(ChatTitle);
}
private void UpdateChatTitle()
{
    if (chatHistory.Count > 0)
    {
        var topic =
GetChatTopic(chatHistory.First().message);
        ChatTitle = $"{ViewModel.ChatTopic} {topic}";
        ((AppShell)Shell.Current)?.SetTitle(ChatTitle);
    }
}
private string GetChatTopic(string message)

```

```

{
    var words = message.Split(new[] { ' ', '!', ',', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);
    return words.Length > 3 ? string.Join(" ",
words.Take(3)) : message;
}
private async void OnSendOrStopMessage(object sender,
EventArgs e)
{
    if (isSending)
        StopSending();
    else
        await SendMessage();
}
private async void OnSendMessage(object sender,
EventArgs e)
{
    await SendMessage();
}
private async Task SendMessage()
{
    string userMessage = InputEntry.Text;
    await Task.Delay(50);
    if (string.IsNullOrEmptyOrWhiteSpace(userMessage)) return;
    InputEntry.Text = string.Empty;
    cancellationTokenSource = new();
    await HandleIncomingMessage(userMessage,
cancellationTokenSource.Token);
}
private void StopSending()
{
    cancellationTokenSource?.Cancel();
    isSending = false;
    UpdateSendButton();
}
private void UpdateSendButton()
{
    SendButton.Source = isSending ? "stop_icon.svg" :
"send_icon.svg";
}
private async Task HandleIncomingMessage(string
message, CancellationToken token)
{
    InputEntry.Unfocus();
    if (chatHistory.Exists(m => m.message == message))
return;
    isSending = true;
    UpdateSendButton();
    AddMessageToChat(ViewModel.You, message,
Color.FromRgb(48, 95, 213), Colors.White);
    chatHistory.Add((ViewModel.You, message));
    await ScrollToBottom();
    var aiFrame = AddMessageToChat(ViewModel.Lama,
"", Colors.Black, Color.FromRgba(0, 0, 0, 20));
    var aiLabel =
((StackLayout)aiFrame.Content).Children[1] as Label;
    if (aiLabel != null)
    {
        await AnimateDots(aiLabel, token);
        string response = await
GetTogetherAIResponse(token);
        await AnimateResponse(aiLabel, response, token);
    }
}

```

```

        await ScrollToBottom();
        chatHistory.Add((ViewModel.Lama, response));
    }
    isSending = false;
    UpdateSendButton();
}
private async Task AnimateDots(Label label,
CancellationTokentoken)
{
    string[] variants = { "Lama думает.", "Lama думает..",
"Lama думает..." };
    int i = 0;
    int max = 30;
    while (!token.IsCancellationRequested && i < max)
    {
        label.Text = variants[i % 3];
        await Task.Delay(500);
        i++;
    }
    if (i >= max) label.Text = "Lama задумалась...";
}
private async Task<string>
GetTogetherAIResponse(CancellationTokentoken)
{
    try
    {
        using HttpClient client = new() { Timeout =
TimeSpan.FromSeconds(15) };
        client.DefaultRequestHeaders.Add("Authorization", "Bearer "
+ ApiKey);
        var messages = chatHistory.Select(m => new
        {
            role = m.sender == ViewModel.You ? "user" :
"assistant",
            content = m.message
        });
        var request = new
        {
            model = "meta-llama/Llama-3.3-70B-Instruct-
Turbo-Free",
            messages,
            temperature = 0.7,
            max_tokens = 6000
        };
        var response = await client.PostAsync(ApiUrl,
new
StringContent(JsonConvert.SerializeObject(request),
Encoding.UTF8, "application/json"), token);
        var result = await
response.Content.ReadAsStringAsyncAsync();
        dynamic json =
JsonConvert.DeserializeObject(result);
        return
json?.choices?[0]?.message?.content?.ToString() ?? "Ошибка:
пустой ответ от AI.";
    }
    catch (TaskCanceledException)
    {
        return "Ответ AI занял слишком много времени.";
    }
    catch
    {

```

```

        return "Ошибка сети. Попробуйте еще раз.";
    }
}
private async Task AnimateResponse(Label label, string
message, CancellationTokentoken)
{
    label.Text = "";
    double previousHeight = label.Height;
    foreach (char c in message)
    {
        if (token.IsCancellationRequested) return;
        label.Text += c;
        await Task.Delay(20);
        await Task.Yield();
        if (Math.Abs(label.Height - previousHeight) > 1)
        {
            previousHeight = label.Height;
            await ScrollToBottom();
        }
    }
    await ScrollToBottom();
}
public async void LoadChat(string chatId)
{
    if (string.IsNullOrEmpty(chatId)) return;
    try
    {
        var userEmail =
AuthService.GetUserEmail().Replace("@", "_").Replace(".",
"_");
        var chatRef = new FirebaseClient("https://realtalk-ai-
default-rtdb.firebaseio.com/")
.Child("users").Child(userEmail).Child("chats").Child(chatId);
        var chatData = await
chatRef.OnceSingleAsync<dynamic>();
        if (chatData?.Messages == null) return;
        chatHistory.Clear();
        ChatLayout.Children.Clear();
        foreach (var msg in chatData.Messages)
        {
            string sender = msg.sender;
            string message = msg.message;
            chatHistory.Add((sender, message));
            var isUser = sender == ViewModel.You;
            AddMessageToChat(sender, message,
isUser ? Color.FromRgb(48, 95, 213) :
Colors.Black,
isUser ? Colors.White : Color.FromRgba(0, 0, 0,
20));
        }
        ChatTitle = $"{ViewModel.ChatTopic}
{chatData.Topic ?? "Без темы"}";
        ((AppShell)Shell.Current)?.SetTitle(ChatTitle);
        await ScrollToBottom();
    }
    catch (Exception ex)
    {
        Debug.WriteLine($"Ошибка при загрузке чата:
{ex.Message}");
    }
}

```

```

private Frame AddMessageToChat(string sender, string
message, Color textColor, Color backgroundColor)
{
    var senderLabel = new Label
    {
        Text = sender,
        FontFamily = "Nunito-ExtraBold",
        TextColor = textColor,
        FontSize = 14
    };
    var messageLabel = new Label
    {
        Text = string.IsNullOrEmpty(message) ? "Lama
думает..." : message,
        FontFamily = "Nunito-Light",
        TextColor = textColor,
        FontSize = 16
    };
    var frame = new Frame
    {
        BackgroundColor = backgroundColor,
        BorderColor = Color.FromRgba(0, 0, 0, 20),
        CornerRadius = 15,
        Padding = 10,
        Margin = new Thickness(10, 2),
        Content = new StackLayout { Spacing = 2, Children
= { senderLabel, messageLabel } },
        HorizontalOptions = sender == ViewModel.You ?
LayoutOptions.End : LayoutOptions.Start,
        HasShadow = true
    };
    ChatLayout.Children.Add(frame);
    return frame;
}
private async Task ScrollToBottom()
{
    await Task.Delay(100);
    Device.BeginInvokeOnMainThread(async () =>
    {
        await Task.Delay(100);
        await ChatScrollView.ScrollToAsync(ChatLayout,
ScrollToPosition.End, true);
    });
}
}
using Firebase.Auth;
using Firebase.Auth.Providers;
using Firebase.Auth.Repository;

namespace RealTalk_AI;

public partial class ResetPasswordPage : ContentPage
{
    private FirebaseAuthClient authClient;
    public ResetPasswordPage()
    {
        InitializeComponent();
        BindingContext = new
ViewModels.ResetPasswordViewModel();
        NavigationPage.SetHasNavigationBar(this, false);
    }
}

```

```

var config = new FirebaseAuthConfig
{
    ApiKey = "AIzaSyCdTJg_iSeWoX2Ete3-
8emKdqBVnY71AIA",
    AuthDomain = "realtalk-ai.firebaseio.com",
    Providers = new FirebaseAuthProvider[]
    {
        new EmailProvider()
    },
    UserRepository = new
FileUserRepository("RealTalk_AI")
};

authClient = new FirebaseAuthClient(config);
}
private async void OnSignInClicked(object sender,
EventArgs e)
{
    await Navigation.PushAsync(new SignInPage());
}
private async void OnResetPasswordClicked(object sender,
EventArgs e)
{
    string email = emailEntry.Text;

    if (string.IsNullOrEmpty(email))
    {
        await DisplayAlert("Ошибка", "Введите почту",
"OK");
        return;
    }
    try
    {
        await authClient.ResetEmailPasswordAsync(email);
        await DisplayAlert("Успех", "Письмо для сброса
пароля отправлено!", "OK");
        await Navigation.PushAsync(new SignInPage());
    }
    catch (FirebaseAuthException ex)
    {
        await DisplayAlert("Ошибка", $"Ошибка сброса
пароля: {ex.Reason}", "OK");
    }
    catch (Exception ex)
    {
        await DisplayAlert("Ошибка", $"Произошла ошибка:
{ex.Message}", "OK");
    }
}
}
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

public static class FirebaseService
{
}

```

```

private static readonly FirebaseClient _databaseClient = new
FirebaseClient("https://realtalk-ai-default-
rtdb.firebaseio.com/");

public static async Task SaveChatHistoryAsync(string
userEmail, string nickname,
List<(string sender, string message)> chatHistory, string
chatTopic, string chatId = null)
{
    var formattedEmail = userEmail.Replace("@",
" _").Replace(".", "_");
    var dbRef = _databaseClient
        .Child("users").Child(formattedEmail).Child("chats");

    Debug.WriteLine($"Saving chat with ChatId: {chatId}");

    var chatData = new
    {
        Username = nickname,
        Topic = chatTopic,
        Messages = chatHistory.Select(m => new { sender =
m.sender, message = m.message }).ToList(),
        Timestamp =
DateTimeOffset.UtcNow.ToUnixTimeSeconds() // <-
Сохраняем текущее время в секундах
    };

    if (!string.IsNullOrEmpty(chatId))
    {
        await dbRef.Child(chatId).PutAsync(chatData);
    }
    else
    {
        await dbRef.PostAsync(chatData);
    }
}

public static class AuthService
{
    private const string TokenKey = "AuthToken";
    private const string EmailKey = "UserEmail";
    private const string UserIdKey = "UserId";
    private const string NicknameKey = "UserNickname";

    public static string GetAuthToken()
    {
        return Preferences.Get(TokenKey, string.Empty);
    }

    public static void SetAuthToken(string token)
    {
        Preferences.Set(TokenKey, token);
    }

    public static bool IsUserAuthenticated()
    {
        return !string.IsNullOrEmpty(GetAuthToken());
    }

    public static void Logout()
    {
        Preferences.Remove(TokenKey);

```

```

        Preferences.Remove(EmailKey);
        Preferences.Remove(UserIdKey);
        Preferences.Remove(NicknameKey);
    }

    public static void SetUserEmail(string email)
    {
        Preferences.Set(EmailKey, email);
    }

    public static string GetUserEmail()
    {
        return Preferences.Get(EmailKey, string.Empty);
    }

    public static void SetUserId(string userId)
    {
        Preferences.Set(UserIdKey, userId);
    }

    public static string GetUserId()
    {
        return Preferences.Get(UserIdKey, string.Empty);
    }

    public static void SetUsername(string nickname)
    {
        Preferences.Set(NicknameKey, nickname);
    }

    public static string GetUsername()
    {
        return Preferences.Get(NicknameKey, string.Empty);
    }
}

using System;
using System.Globalization;
using Microsoft.Maui.Controls;
using RealTalk_AI.Resources;

namespace RealTalk_AI.Utils
{
    public class LocalizationConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            if (parameter is string key)
            {
                return LocalizationResourceManager.Instance[key];
            }

            return string.Empty;
        }


        public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}

```



**ПРИЛОЖЕНИЕ Б**  
(обязательное)  
**Результат работы мобильного приложения**

## Создать аккаунт



Почта

 Введите почту

Пароль

 Введите пароль 

Повторите пароль

 Повторите пароль 

Создать аккаунт


Уже есть аккаунт?

Войти



Рисунок Б.1 – Вкладка «Страница регистрации» мобильного приложения  
«Генератор диалогов с ИИ с использованием .NET MAUI и AI»

# Войти в аккаунт

Почта

 Введите почту

Пароль

 Введите пароль 

Забыли пароль?

Войти

Нет аккаунта?

Создать аккаунт

Рисунок Б.2 – Вкладка «Страница авторизации» мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»



**Восстановить пароль**

Почта

✉ Введите почту

**Восстановить пароль**

Уже есть аккаунт?

**Войти**

Рисунок Б.3 – Вкладка «Восстановить пароль» мобильного приложения  
«Генератор диалогов с ИИ с использованием .NET MAUI и AI»

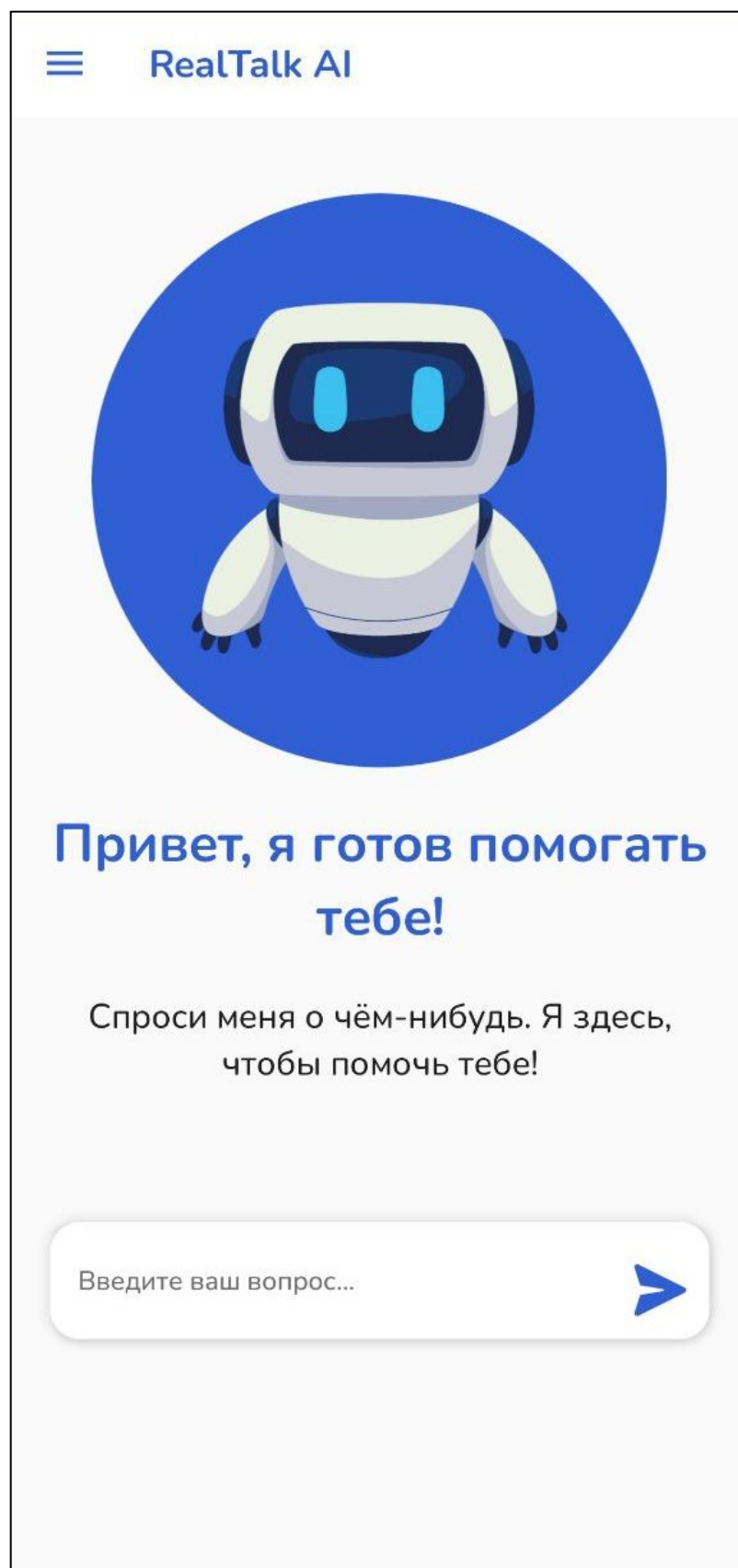


Рисунок Б.4 – Вкладка «Главный экран» мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»

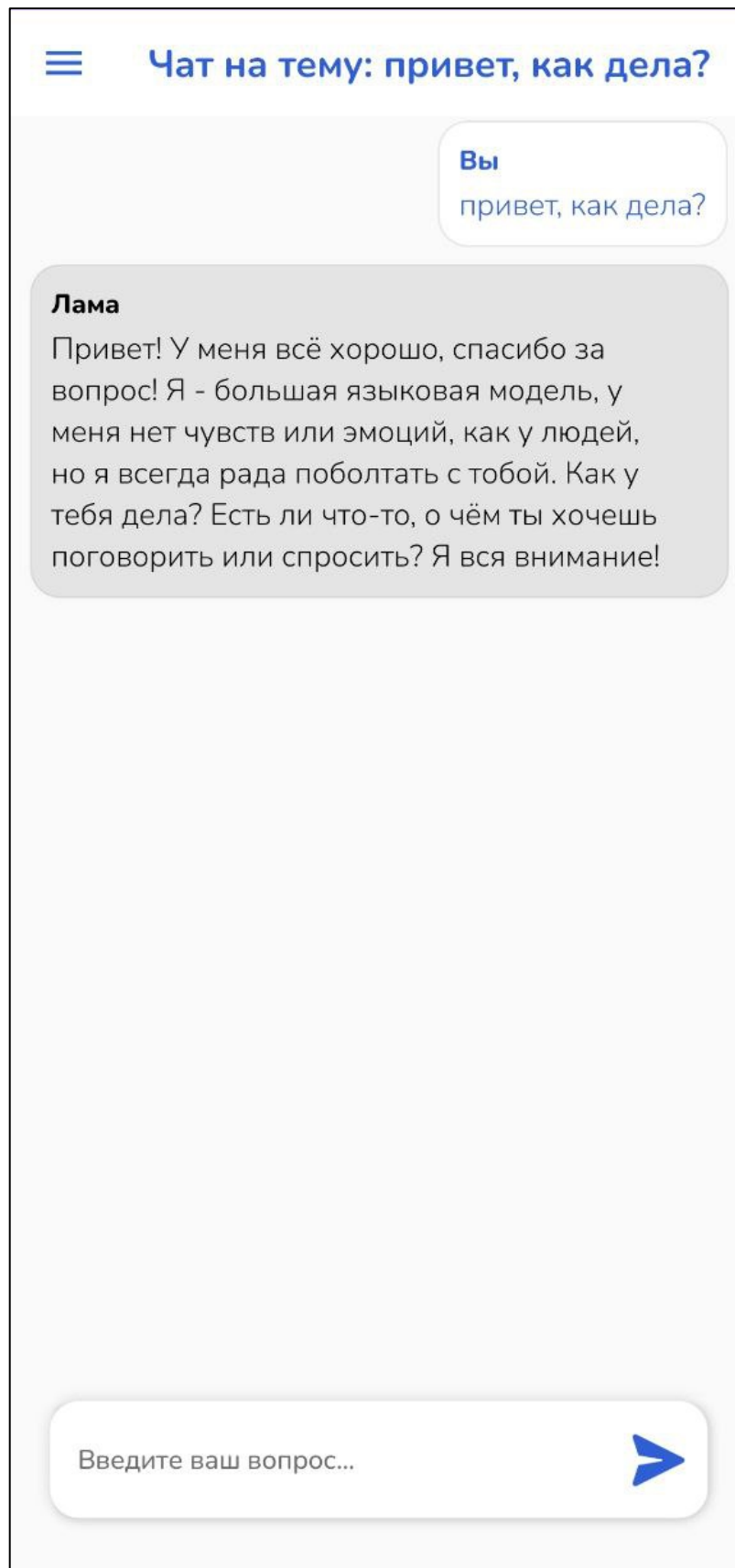


Рисунок Б.5 – Вкладка «Диалог» мобильного приложения  
«Генератор диалогов с ИИ с использованием .NET MAUI и AI»

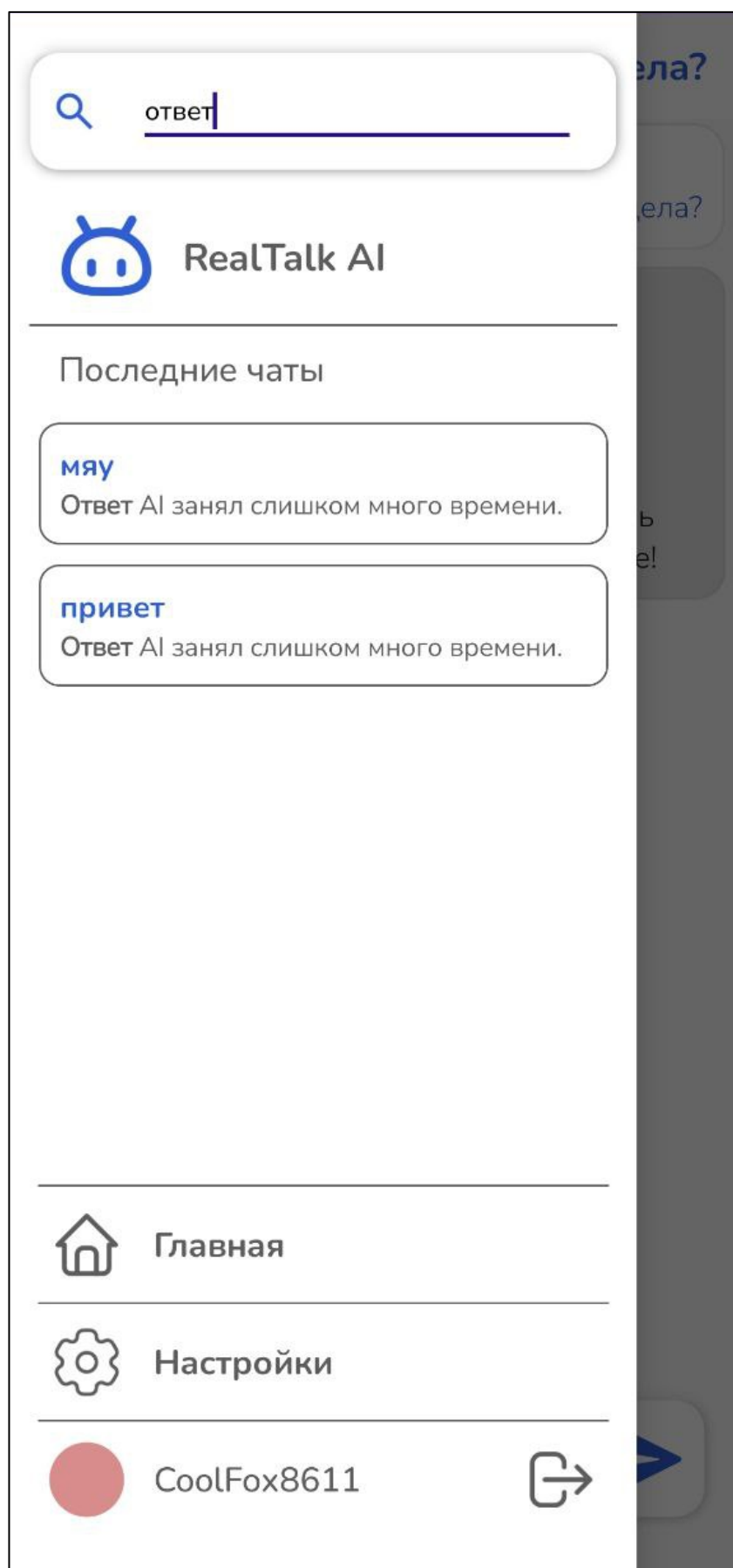


Рисунок Б.6 – Вкладка «Боковая панель» мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»

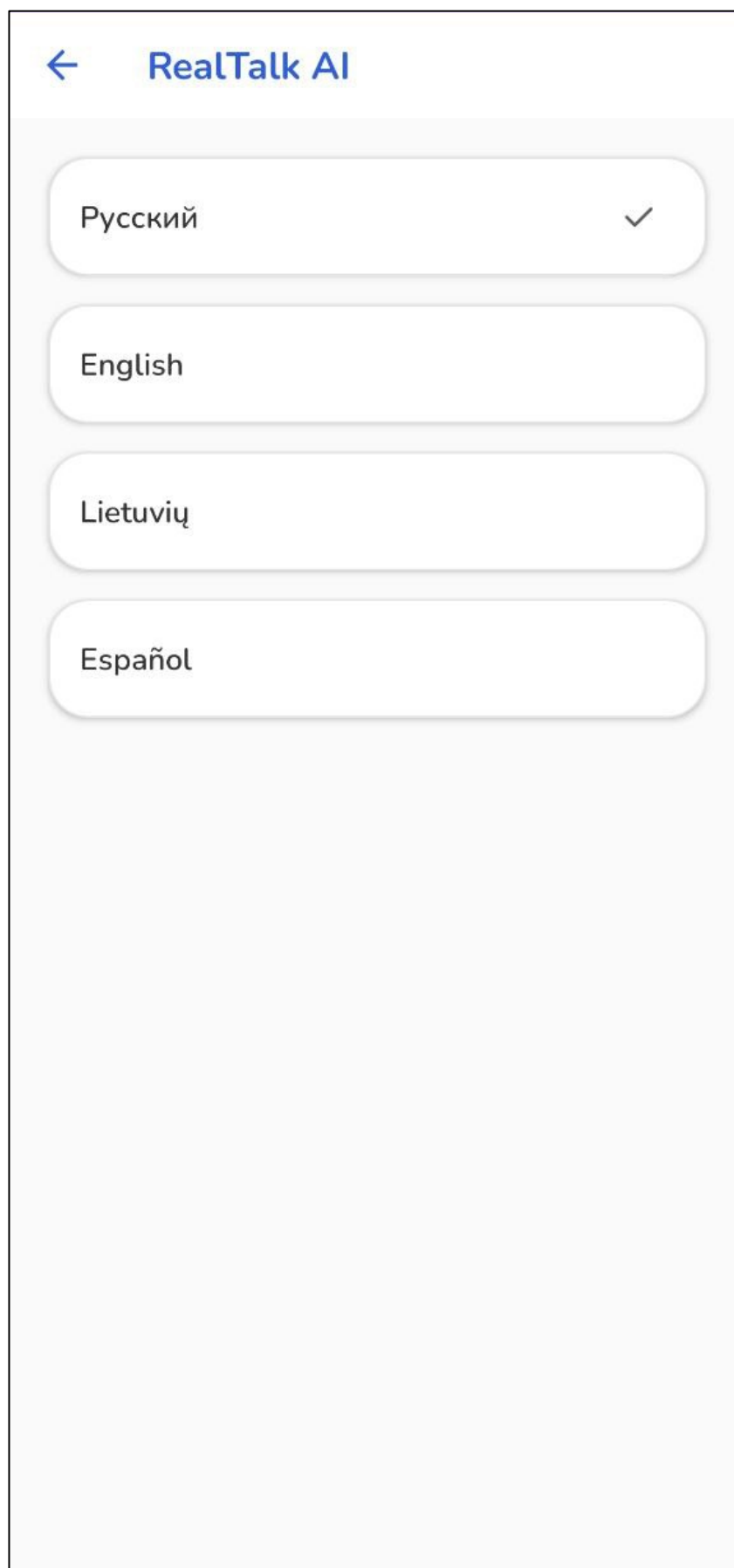


Рисунок Б.7 – Вкладка «Настройки» мобильного приложения «Генератор диалогов с ИИ с использованием .NET MAUI и AI»