

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

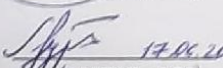
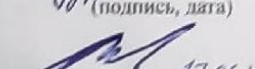
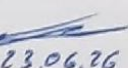
Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

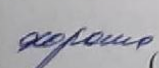
на тему

**РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ
СИСТЕМЫ ПРИНЯТИЯ РЕШЕНИЙ ДЛЯ
ОБРАБОТКИ ФИНАНСОВЫХ ТРАНЗАКЦИЙ**

Студент	 (подпись, дата)	Щеваев Михаил Владимирович (ФИО полностью)
Руководитель	 (подпись, дата)	Л.В. Гурьянов (фамилия, инициалы)
Нормоконтролёр	 (подпись, дата)	А.С. Бычков (фамилия, инициалы)
Рецензент	Доцент ПензГТУ  (должность, место работы) (подпись, дата)	С.В.Печерский (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

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

Работа защищена с отметкой 
(протокол заседания ГЭК от 23.06.2026 № 34)

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

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 20 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Щеваев Михаил Владимирович гр. 24ВВИм1 факультета ВТ

2. Тема работы Разработка автоматизированной системы принятия решений для обработки финансовых транзакций

Тема утверждена приказом ПГУ № 920/0 от " 20 " 09 20 24 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры МО и ПЭВМ ПГУ Гурьянов Л.В.

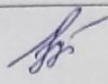
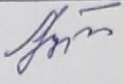
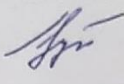
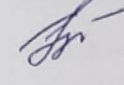
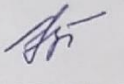
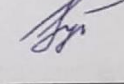
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Провести исследование научной области и создать автоматизированную программу принятия решений для обработки финансовых транзакций, на базе языковых моделей. Разработать AI-агента, распознающего текстовые запросы от пользователя, для формирования и изменения отчетов о финансовых переводах. Основываясь на базе бакалаврской работы, программный модуль должен быть совместим с мобильной версией приложения банка. Итоговая программа должна представлять систему на основе AI-агента, отвечающего запросам пользователя с клиентской стороны приложения.

5. Перечень подлежащих разработке вопросов

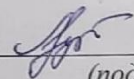
Необходимо проанализировать существующие сервисы, работающие в сфере транзакций, чтобы выявить их недостатки и преимущества. Проверить работающие языковые модели на пригодность к взаимодействию с финансами. Сделать выводы о возможных сценариях коммуникации приложения с клиентами, на базе Python, LangChain и GigaChat. Выполнить анализ и настроить способы передачи данных от пользователя до AI-агента. Выразить конкретные команды для реализации вывода информации о переводах, платежах и других операциях. Настроить систему, передав право выбора текущего решения от программиста к языковой модели. Сформулировать подходящий план для поставленной задачи и получить работоспособный продукт.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Постановка задачи и анализ предметной области	10%	20.09.24-15.11.2024	
2.	Изучение литературы и существующих решений	15%	19.11.2024-28.12.2024	
3.	Создание программного обеспечения	25%	29.12.2024-17.04.2025	
4.	Реализация итоговой структуры	30%	18.04.2025-15.09.2025	
5.	Выявление и устранение неполадок	15%	16.09.2025-10.12.2026	
6.	Оформление пояснительной записки	5%	01.05.2026-20.05.2026	

Дата выдачи задания " 20 " 09 2024 г.

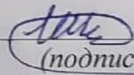
Руководитель магистерской диссертации



(подпись, дата)

Гурьянов Л.В.

Задание к исполнению принял студент



(подпись, дата)

Щеваев М.В.

Работу к защите допустить

Декан факультета



23.06.2026

Фионова Л.Р.

РЕФЕРАТ

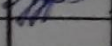
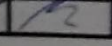
Пояснительная записка содержит здесь до приложения 85 лист, 31 рисунок, 10 источников, 2 приложения.

ИНТЕЛЛЕКТУАЛЬНЫЙ АГЕНТ, ФИНАНСОВЫЕ ПЕРЕВОДЫ, LLM,
БОЛЬШАЯ ЯЗЫКОВАЯ МОДЕЛЬ, РАСПРЕДЕЛЕННАЯ АРХИТЕКТУРА,
РАСПОЗНАВАНИЕ ТЕКСТА, МУЛЬТИМОДАЛЬНЫЙ

Главной целью работы является создание помощника на основе искусственного интеллекта для принятия решений по обработке финансовых транзакций. Создаваемое приложение призвано разгрузить персонал финансовых учреждений, заменив человека на систему, различающую человеческие команды. Разработка была направлена на анализ предметной области, программирование системы, реализацию ее графического интерфейса и тестирование.

В конце работы была создана распределенная архитектура на базе интеллектуального агента, языка программирования Python, LangChain и мультимодальной модели GigaChat, для совершения продвинутых диалогов с пользователем. Графическая часть была спроектирована при помощи фоторедактора Photoshop.

Большое внимание было направлено на формулирование запросов, понятных большой языковой модели, создание пользовательского интерфейса и анализ результатов работы интеллектуального агента.

ПГУ 1.090401.13.001 ПЗ				
Изм.	Лист	№ докум	Подпись	Дата
Разраб.		Щеваев М.В.		4.06.26
Провер.		Гурьянов Л.В.		4.06.26
Н. Контр.		Бычков А.С.		4.06.26
Утв.		Митрохин М.А.		4.06.26
Разработка автоматизированной системы принятия решений для обработки финансовых транзакций Пояснительная записка				
		Лит.	Лист	Листов
			2	68
ФВТ, каф. ВТ гр. 24ВВИм1				

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ.....	7
ВВЕДЕНИЕ	8
1 Анализ предметной области.....	10
1.1 Анализ требований.....	10
1.2 Анализ больших языковых моделей.....	12
1.2.1 RuGPT-3.5	13
1.2.2 Kandinsky 2.1.....	15
1.2.3 RuCLIP.....	16
1.2.4 FRED-T5	16
1.2.5 Вывод.....	17
2 Архитектура языковых моделей	18
2.1 Цель Encoder	19
2.2 Задача Decoder	22
2.3 Архитектура Decoder-only Transformer	22
3 Анализ аналогов	26
4 Архитектура приложения	28
4.1 Процессы работы со статистикой	29
4.2 Архитектура клиент-сервер	31
4.3 Проектирование структуры базы данных.....	32
4.4 Проектирование программных средств.....	33
4.5 Проектирование пользовательского интерфейса	35
5 Контроль качества программного продукта	51
5.1 Класс эквивалентности.....	51
5.1.1 Граничные значения.....	51
5.1.2 Тест-кейсы.....	52
5.1.3 Функциональное тестирование белым ящиком	52
5.2 Общий сценарий программных средств.....	54
5.2.1 Алгоритм SPAM	55
5.2.2 Альфа-алгоритм.....	56

5.2.3 Эвристический алгоритм.....	58
5.2.4 Реализация эвристического алгоритма	60
5.2.5 Реализация индуктивного алгоритма.....	60
5.2.6 Формирование ключевых особенностей построения алгоритма	62
ЗАКЛЮЧЕНИЕ.....	64
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	65
ПРИЛОЖЕНИЕ А – ЛИСТИНГ ПРОГРАММ	66
ПРИЛОЖЕНИЕ Б - ПРЕЗЕНТАЦИЯ.....	74

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В этой работе применяются следующие сокращения и обозначения.

ИИ – Искусственный интеллект

AI (Artificial Intelligence) – Искусственный интеллект

API (Application Programming Interface) – Программный интерфейс приложения

BERT (Bidirectional Encoder Representation from Transformers) – Двухнаправленный кодер из трансформера

CORS (Cross-Origin Resource Sharing) – Механизм совместного использования ресурсов между разными источниками

DL (Deep Learning) – Глубокое обучение

LSTM (Long Short-Term Memory) – Рекуррентная сеть с долгой краткосрочной памятью

LLM (Large Language Model) – Большая языковая модель

ML (Machine Learning) – Машинное обучение

REST (Representational State Transfer) – Архитектурный стиль взаимодействия web-сервисов

RNN (Recurrent Neural Network) – Рекуррентная нейронная сеть

RNNT (Recurrent Neural Network Transducer) – Архитектура нейросетевой модели для распознавания речи

URL (Uniform Resource Locator) – Единый указатель ресурса

ВВЕДЕНИЕ

С развитием современных технологий, а именно графических процессоров, ведущие лидеры цифровых сервисов начали внедрение систем искусственного интеллекта. Облачные технологии вынесли мощности вычислительных машин далеко за рамки привычного использования людьми персональных компьютеров. Интеллектуальные агенты принялись за вытеснение реальных сотрудников, перенимая их навыки под чутким руководством программистов нейронных сетей. Лучше всего большие языковые модели зарекомендовали себя в распознавании текстовых сообщений. Системы обучаемые на миллиардах абзацев текста, созданных людьми за годы жизни человечества, в текущий момент времени могут без труда дописать запрос в поисковике и в ту же секунду отправить краткую выжимку по вопросу. Данные технологии позволяют экономить часы работы, денежные вложения для текущих и новоиспеченных проектов. Всё это открывает небывалые возможности для информационных решений, что позволяет возводить быстрые и эффективные системы, опираясь на многолетний опыт разработки программного обеспечения.

Рост объемов потребляемых данных языковыми моделями сказывается на их мощностях, формируя бесплатные решения мультимодальных систем для общего использования. Одним из таких решений является модель GigaChat, оптимизированная под русских пользователей, от компании Сбербанк.

Модель от российского банка отвечает на базовые запросы, такие как: ведение диалога; формирование и редактирование текста, создание изображений по описанию; написание и анализирование программного кода. Качество выполнения задач моделью растет с ходом развития удаленных систем, что позволяет улучшить текущий уровень обслуживания в розничной и оптовой деятельности.

В 2025 году аналитики агентства Smart RanKing зафиксировали, рост рынка технологий искусственного интеллекта увеличился на 25% [1]. С 2023

года выход на рынок новоиспеченных российских компаний привел к усилению конкуренции в среде. Поддержка отечественных разработок, обусловленная повышением спроса на новые технологии, привела к росту индивидуального подхода в клиентской сфере. Большие языковые модели переключились с написания текста на естественном человеку языке и углубились в написании программного кода. Привычные логические операции системы разбирают и исполняют с высокой долей успеха, но без человеческого участия результат может быть далек от желаемого. Но даже такого упрощения достаточно, чтобы сформулировать новые задачи к сервису, занимающемуся обработкой финансовых транзакций. Основные задачи магистерской диссертации в этой связи включают в себя:

- анализ ключевых моделей, которые анализируют и формируют текстовые решения на основе больших данных,
- исследование текущих структур в сфере финансов по теме платежей и переводов,
- выбор оптимального алгоритма искусственного интеллекта для оптимизации повседневных задач банковской сферы,
- анализ итогов.

Целью выпускной квалификационной работы является создание решения с использованием больших языковых моделей для обработки финансовых операций.

1 Анализ предметной области

Перед началом работы следует провести теоретический анализ доступных на рынке цифровых технологий, представляющих системы обработки больших данных, содержащих в себе следующие возможности: дробление целей на фракталы подзадач, распределение архитектуры, структурирование потоков информации, систематизирование последовательностей модулей логики, понятное массовому потребителю графическое представление возможностей приложения. Все эти аспекты должны совмещаться в программном устройстве, обеспечивая заданные требования к микросервису.

Для выбора языковой модели потребуется исследовать основные задачи, решаемые сложными структурами, обрабатывающими массивы данных, что должно сказаться на обосновании типа модели.

1.1 Анализ требований

Из всего многообразия, учитывая необходимые требования, можно отобрать подходящую модель, которая будет отвечать поставленной задаче. Такой подход к выбору архитектуры предполагает следующие требования:

- бесплатный доступ к сервису со стороны клиентов,
- местное использование программы для просмотра важной информации,
- модульная структура,
- достаточная вычислительная мощность модели,
- низкая степень ошибки при выводе ожидаемого результата,
- шаблонный анализ запросов пользователя с возможностью гибкого изменения ошибок и вывода релевантного ответа,
- способность к формированию отчета о переводах и пополнениях ввиду структуры шаблонных ответов,
- вывод ошибочных значений в формате ключ-ошибка,
- работа с лицензионным программным обеспечением от разработчика большой языковой модели в рамках законодательства,

– реализация понятного интерфейса со стороны пользователя, отвечающие на базовые требования к системе.

Большая языковая модель была неоднократно протестирована на стороне сервиса GigaChat от Сбербанка. Что дало полный спектр в анализе возможностей удаленного решения.

Стоит упомянуть авторские права, распространяющиеся на подобного типа программное обеспечение. Открытый фреймворк LangChain позволяет разрабатывать приложения на базе больших языковых моделей в бесплатном контексте. В случае с GigaChat, система сама предоставляет первый пакет токенов для домашней разработки приложения, а для юридических лиц пакеты можно докупать. При соблюдении простых правил, владельцы эти структур позволяют коммерческое проектирование и создание архитектур на базе их разработок.

1.2 Анализ больших языковых моделей

Анализ языковых моделей стоит начать с определения, касающегося нейросетей. Они представляют математическую модель, способную имитировать деятельность нейронных сетей мозга человека. За передачу данных в них отвечают нейроны. Эти модели призваны обучаться на массивах информации и решать задачи различной структуры. Нейросетевые модели имеют широкий спектр применения, они затрагивают следующие ниши обработки данных: работа с изображениями; использование последовательных данных; обработка естественных языков; генерация и анализ информации; решение задач классификации. Типовой пример работы нейросети показан на рисунке 1.

Искусственный нейрон

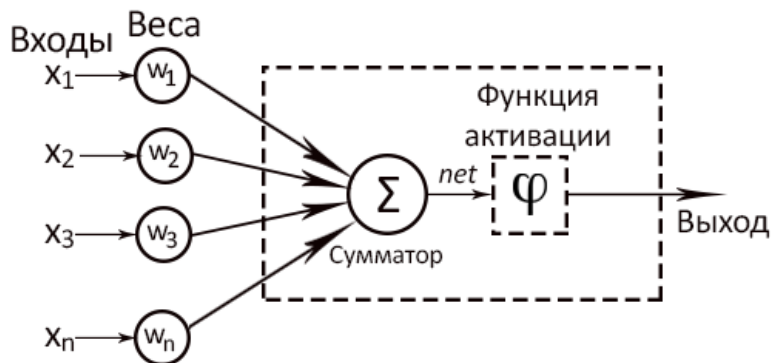


Рисунок 1 – Процесс работы искусственного нейрона

Модель нейрона содержит в себе три основных компонента:

- синапсы – отвечают за входные сигналы, каждый из них обладает своим весом,
- сумматор – с учетом значений весов, складывает входные сигналы,

– функция активации – компонент, отвечающий за разброс выходного сигнала, это позволяет нейрону держать выход на отрезке от 0,1 до -1,1.

Основываясь на базовом примере работы нейрона в искусственной сети можно сделать вывод, что структура демонстрирует высокую производительность в классификации объектов, разделяя их на группы рисков. Однако в ситуациях с высокой степенью неопределенности ее прогнозы могут быть неточными. В этой связи надежность можно дополнить экспертной оценкой факторов риска, переобучая модель на новых массивах данных.

Понимание работы нейросетей лежит в основе анализа рыночных аналогов, это позволяет оценить риски и подготовиться к нежелательным последствиям. Данные знания будут применены в магистерской диссертации.

1.2.1 RuGPT-3.5

RuGPT-3.5 – это большая языковая модель, предназначенная для создания и генерации текста на русском языке. Базируется на технологиях OpenAI, при поддержке Института искусственного интеллекта AIRI. Она легла в основании GigaChat, пройдя дообучение на базе знаний сотрудников подразделения. Система является декодерной нейросетью-трансформером, базу в ней заложили архитектурой GPT-3 инженеры OpenAI. Модель настроена на работу с русским языком, обгоняя зарубежные аналоги, обученные на английском языке.

RuGPT-3.5 содержит 13 миллиардов параметров, с длиной контекста в 2048 токенов. На первом этапе в нее загружали научные статьи, энциклопедии, книги, содержимое социальных сетей из открытых источников. Второй этап вобрал в себя датасет из The Stack, измененные статьи из Википедии и дела из юриспруденции [2].

Сообщество любителей доработало версию до квантового решения, что позволило описать команды для обработки и комбинирования семантических элементов, понятным образом для модели. Этот подход помог снизить нагрузку на вычислительные устройства, увеличив количество пользователей, заинтересованных в развитии данной нейросети.

Отличительной особенностью `ruGPT-3.5` является оптимизация под русскоязычный сегмент. Ключевым нюансом в использовании модели является ее распространение под лицензией MIT, она же в свою очередь, предполагает свободную доработку и передачу решения с минимальными ограничениями.

1.2.2 Kandinsky 2.1

Kandinsky 2.1 – это генеративная нейросеть, сутью которой является воссоздание изображений по текстовому описанию. Модель может работать с 101 языком, имея в своем распоряжении палитру стилизации выходной картины по нескольким темам. Обучали нейросеть на сложной для восприятия выборке датасетов, состоящей из иллюстраций человеческих лиц.

С формированием схемотехники изображения в высоком качестве, модели помогает увеличенное число параметров и дополнительное обучение на 170 миллионах пар, состоящих из текста и связанных с ним картинок. Система имеет несколько вариантов использования в сети, от сайта с удобным интерфейсом до мобильного приложения. Художники вместе с дизайнерами и энтузиастами могут использовать ее в качестве мольберта для набросков и быстрых зарисовок. Авторы не рекомендуют формировать запросы со сложными позами, подобное использование нейросети может привести к сдвигу силуэтов лиц и общему снижению качества выходного результата [3].

Модель содержит энкодер (XLM-Roberta-Large-Vit-L-14), блок `image prior`, который согласует текст и вектора многомерного пространства, а также латентную диффузию U-Net с MoVQ (энкодер/декодер). Распространяемая под лицензией Apache-2.0, модель представляет интерес в контексте транзакций из-за своей склонности связывать текст и цвет, как это делает пара ключ-значение в базах данных. Исходя из этого можно назначить оттенок для каждой транзакции, отсортировав их по степени важности на текущий момент времени. Такого вида наработки могут существенно упростить формирование финансовых отчетов, снизив риски, упустить важные события в контексте обработки платежей и переводов.

1.2.3 RuCLIP

RuCLIP – это модульное решение в контексте русской культуры. Аналогом является ее старший брат от OpenAI, но ее приспособили к работе с кириллицей. Разработка велась в команде SberDevices и Sber AI. Структуру модели делят два компонента (image encoder и text encoder). Image Encoder преобразует изображение в векторные слепки. Text Encoder кодирует описание, состоящие из текста, на русском языке. Ее трансформеры отличаются от версии к версии. Вероятность неверных комбинаций снижается на фоне максимизации близости эмбеддингов правильных пар. Главной задачей эмбеддингов является сохранение смысловой связи объектов: в многомерном пространстве соседствующие элементы получают схожие значения, это видно на рисунке 2.



Рисунок 2 – Оценка работы модели ruCLIP

На датасете из 240 миллионов пар экосистемы Сбера обучение новой модели занимает 12 дней. К главным особенностям системы можно отнести решение вопросов компьютерного зрения в режиме zero-shot и few-shot. Программа справляется с ранжирование изображений в контексте

пользовательских запросов, созданием простого описания к продуктам в интернет-ресурсах, классификацией изображений без дополнительного обучения и простым поиском картинок [4].

Точность выводимого изображения в мультимодальных системах может опускаться из-за небольшого изменения текстового описания. Только введение новых тестов для ранжирующих моделей может снизить количество их ошибок, сказавшись на массовости программного обеспечения.

1.2.4 FRED-T5

FRED-T5 – это языковая модель, совмещающая кодирующие и декодирующие функциональные блоки, в отличие от моделей GPT. Содержит 1,7 миллиарда параметров, что повышает скорость обработки естественных языков. Лидер среди тестов в русском сегменте, возвышается над другими нейросетями, уступая в точности только человеку. Способна на воссоздание утраченных элементов текста. Этот подход предложили архитекторы компании Google в модели UL2, но команда разработки Сбербанка доработала это решение. Ключевые элементы системы показаны на рисунке 3.

```
from transformers import T5ForConditionalGeneration, AutoTokenizer

path_to_model = "ai-forever/FRED-T5-large-spell"
model = T5ForConditionalGeneration.from_pretrained(path_to_model)
tokenizer = AutoTokenizer.from_pretrained(path_to_model, eos_token="</s>")

prefix = "Исправь: "
sentence = "прийдя в МГТУ я был удивлен никого не обнаружив там.."
sentence = prefix + sentence
encodings = tokenizer(sentence, return_tensors="pt")
generated_tokens = model.generate(
    **encodings, eos_token_id=tokenizer.eos_token_id, early_stopping=True)
answer = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
print(answer)

# ["прийдя в МГТУ я был удивлен никого не обнаружив там.. «при»"]
```

Рисунок 3 – Демонстрация части кода FRED-T5

Модель способна на внесение правок, неплохо справляясь с ошибками ввода, пунктуационными недочетами и опечатками. Она разбивает блок теста на логически модули для корректного восприятия информации. Рассмотрим

элементы команд в контексте небольшого блока программы от языковой модели FRED-T5:

- sentences – содержит предложение, в котором требуется выявить ошибку,

- batch_size – отвечает за хранение подвыборки входящих данных,

- prefix – отвечает за направление анализа для языковой модели,

- generation_params – хранит параметры для работы метода модели.

Ясно выделяет даты, имена из контекста, названия учреждений, культурных объектов. Используется, как модульное решение, но способна самостоятельно принимать решения, основываясь на входных и выходных параметрах, снижая риски на просчет [5].

FRED-T5 показывает своей архитектурой, что прогресса можно достичь не только путем расширений программных инструментов и числом параметров.

1.2.5 Вывод

Рассмотренные модели, в большинстве своем, верны ключевым параметрам, сформулированным к разработке программного обеспечения. Почти все из них имеют свободный программный код, могут работать вне микросервисной архитектуры, обладают достаточным количеством токенов для первых шагов в сфере больших языковых моделей. Финальное решение будет принято только после практического тестирования лучших кандидатов, на основании точности предлагаемых моделями решений.

2 Архитектура языковых моделей

Основой большинства языковых моделей являются трансформеры, их чаще всего именуют архитектурой. В привычном понимании LLM способна анализировать и создавать человеческий язык. Это стало возможным за счет увеличения пропускной способности шин видеоадаптеров и росту массивов данных. Основываясь на выявленных закономерностях, система учится предсказывать вероятность успешного решения или неудачи. После всех логических операций наиболее успешный блок текста пойдет по конвейеру далее. Именно способность к самоанализу происходящего наделяет модели незаменимыми навыками в работе. За ключевой прорыв в сфере LLM отвечает архитектура трансформер. Ее анонсировали в 2017 году в статье «Attention Is All You Need». До нее балом правили рекуррентные сети. Им тяжело было перерабатывать длинные зависимости, что существенно тормозило работу. Механизм внимания в трансформере решил эту задачу, эта технология обрабатывает каждое слово текста параллельно входной последовательности.

Ключевыми аспектами такого подхода является многоголовое внимание, что намекает на использование нескольких считывающих модулей, вместо одного. Добавление метки с информацией по каждому новому слову формирует понимание последовательности входных данных у модели. После уточнения позиции слов в цепочках, к каждому элементу применяется полносвязный слой нейросети. В таких сетках каждый нейрон соединен со всеми предыдущими нейронами. Через функцию активации, в таком случае, проходят только сигналы с вычисленной взвешенной суммой и смещением [5].

Суть архитектуры трансформера, показана на рисунке 4, она заключается в отказе от рекуррентных и сверточных нейронных сетей, что сводит все к механизму внимания. Улучшенное распараллеливание вычислений, ускоряет обучение модели и требует учитывать зависимости между разными частями текста, а также позволяет определить, какие токены в цепи важны друг для друга.

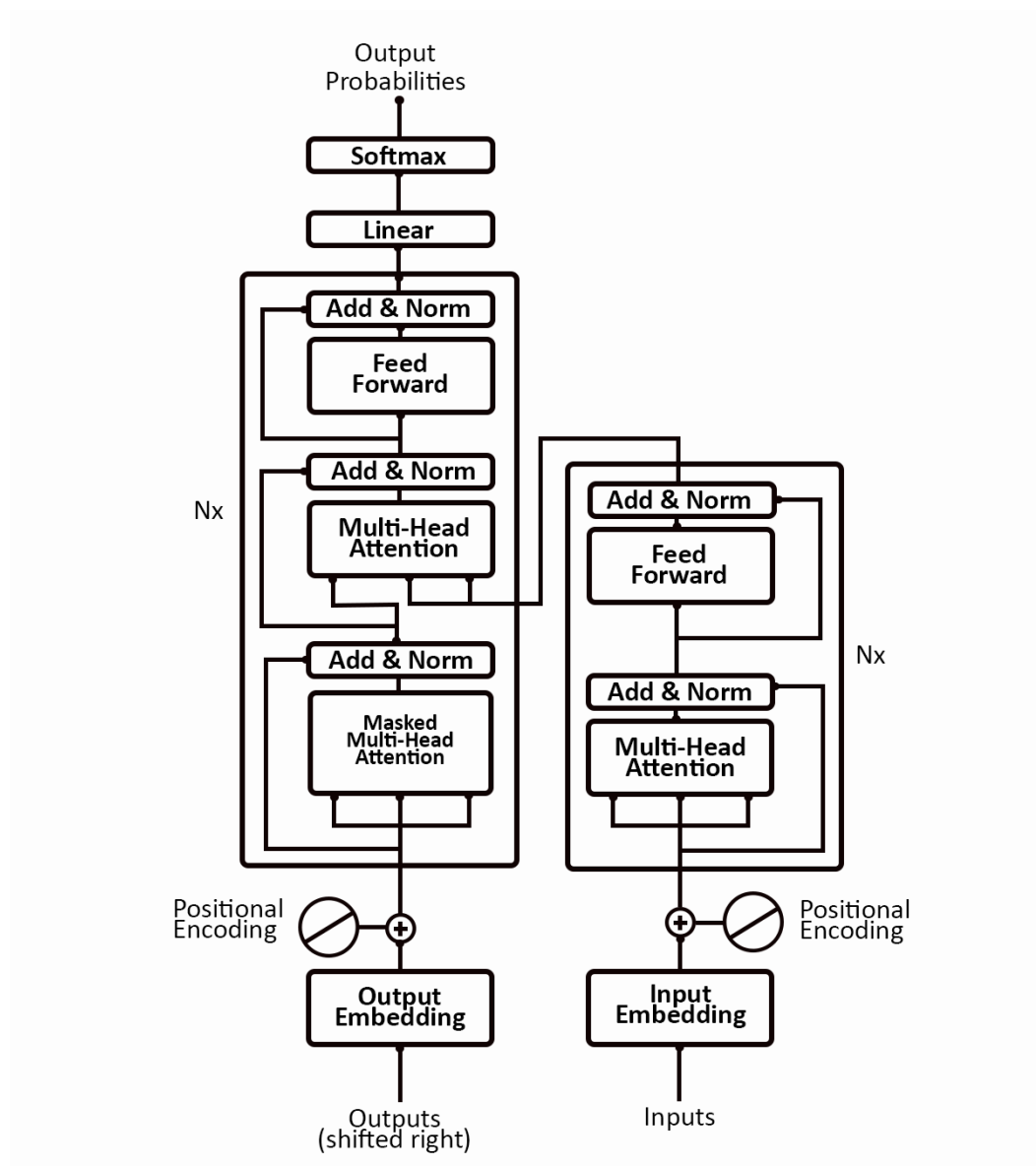


Рисунок 4 – Архитектура трансформера

Архитектуру трансформера делят энкодер и декодер. Энкодер создает из внутреннего представления последовательность токенов. Декодер, использует ранее выведенные токены, пошагово формируя выходную последовательность для переориентирования вывода на естественный язык.

2.1 Цель Encoder

Энкодер создан для структурирования входной последовательности. Отличается от RNN обработкой токены параллельно, а не постепенно. Вход энкодера заполняют токены, которые преобразуются в эмбединги – числовые векторы, представляющие смысл токенов. После они попадают в блоки энкодера.

Самовнимание или же self-attention несут основной слой энкодера. Он занимается обновлением эмбединга каждого токена с учетом ситуации. Для каждого токена суммируются веса остальных токенов, затем эти результаты проходят через функцию softmax, после этого эмбединги объединяются в новый вектор внимания. Этот вектор суммируется с исходным эмбедингом, благодаря чему токен получает данные о своем положении в системе. Но в структуре трансформера принято использовать не один слой внимания, а multi-head attention, то есть многоголобие. Его суть заключается в том, что головы могут анализировать разные цепочки токенов (1). Например, одна голова может учитывать синтаксические параллели, а другая – смысловые, третья же – может уловить связь между существительным и местоимением.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O, \quad (1)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)W^O. \quad (2)$$

где $head_i$ – отдельная голова внимания;

W_i^Q, W_i^K, W_i^V – обучаемые матрицы для формирования запросов, ключей и значений;

W^O - матрица объединения результатов.

Программа трансформера обрабатывает токены параллельно, из-за этого модель не знает их очередности, поэтому используется позиционное кодирование. Добавляя к каждому эмбедингу информацию о позиции токена в цепи. Скажем, предложения «horse runs dog» и «dog runs horse» состоят из одних и тех же слов, но имеют разный смысл, поэтому порядок без учета последовательности токенов не сформируется, это дает понимание структуре, в каком порядке расположены токены. Для этого используются синусоидальные кодировки, которые добавляют каждому токenu индекс в блоке текста (3-4):

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), \quad (3)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right). \quad (4)$$

где pos – позиция токена в последовательности;

i – номер измерения вектора;

d_{model} – размерность внутреннего представления модели.

После слоя self-attention используется блок Add & Norm, он означает суммирование начального эмбединга с результатом слоя внимания, что выгружает остаточную связь в систему. Из-за нее модель не теряет исходные сведения о токене. Контекстная информация добавляется от self-attention, но начальный образ токена тоже сохраняется. Для этого система складывает старый и новый эмбединг.

За стабилизацию отвечает часть Norm, которая выравнивает показатели нейросети. Все это сказывается на устойчивости обучения моделей, минимизируя риски. Плавное и равномерное обучение во время работы с программой гарантирует LayerNorm. То есть нормализация сначала приводит эмбединг к более стабильному масштабу, а затем с помощью параметров γ и β позволяет модели самой подобрать удобные значения (5):

$$x_i^e = \frac{x_i^e - \mu_i}{\sigma_i + \epsilon} * \gamma + \beta. \quad (5)$$

где μ_i – среднее значение координат эмбединга;

σ_i – стандартное отклонение;

ϵ – маленькое число для защиты от деления на ноль;

γ и β – обучаемые параметры.

Для каждого эмбединга сохраняется размерность входа и выхода, ее еще называют feed-forward. Результат работы блока Encoder – это

обновленные эмбединги токенов, причем каждый слой обрабатывает эмбединги параллельно другим потокам.

2.2 Задача Decoder

Декодер Transformer предназначен для генерации выходной последовательности. Во время обучения в него можно подать сразу все токены целевого предложения, используя подход *teacher forcing*. После прохождения через декодер выходные эмбединги передаются в полносвязный слой и *softmax*, после чего получается распределение вероятностей следующего токена. Во время обучения декодеру можно подать сразу все токены *target*-предложения, а затем считать ошибку через *cross-entropy loss*. Декодер состоит из трех основных частей: *masked self-attention*, *cross-attention* и *feed-forward layer*.

Во время реального использования модели декодер работает авторегрессионно: сначала получает начальный токен, затем генерирует следующий, добавляет его к входу и повторяет процесс.

Masked self-attention отличается от обычного *self-attention* тем, что токен не может получать информацию из будущих токенов. Это необходимо, потому что при генерации текста модель не должна заранее знать правильное продолжение. Если во время обучения декодер видит всю целевую последовательность сразу, тогда модель фактически подсматривала бы правильный ответ. В реальном использовании такого будущего токена еще нет, поэтому обучение было бы некорректным и в декодере запрещаются связи с будущими позициями. На рисунке 5 это изображено красными линиями.

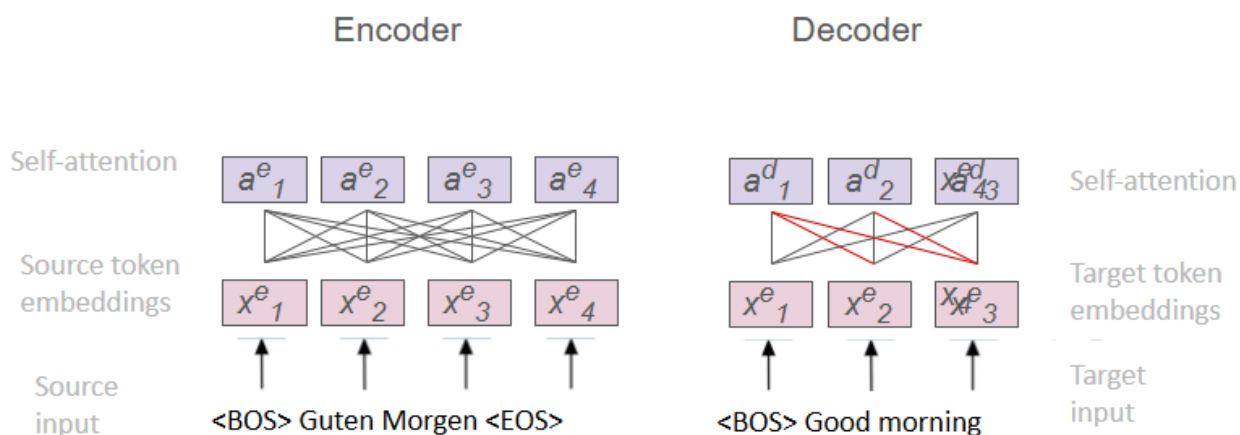


Рисунок 5 - Masked self-attention

То есть первый токен может учитывать только себя, второй – первый и второй, третий – первые три и так далее. Благодаря этому декодер обучается в тех же условиях, в которых потом будет работать при генерации текста.

Вторая важная часть декодера – cross-attention, или перекрестное внимание. Это слой, в котором декодер получает информацию от энкодера. Если masked self-attention помогает декодеру учитывать уже сгенерированные токены, то cross-attention помогает ему учитывать входное предложение.

В задаче перевода это означает следующее: декодер генерирует перевод, но на каждом шаге может обращаться к эмбедингам исходного предложения. Например, при генерации английского слова декодер может посмотреть, какие слова во входной последовательности наиболее важны для текущего шага. Работает cross-attention похоже на обычное внимание, но запросы берутся из декодера, а ключи и значения – из энкодера. То есть декодер спрашивает у энкодера, какая часть исходного предложения сейчас важна.

В Transformer используется вариант внимания, который называется QKV-attention (Query Key Value). Запрос показывает, какую информацию ищет текущий токен. Ключ показывает, какая информация содержится в другом токене. Значение содержит саму информацию, которую нужно передать дальше, если токен оказался важным.

Перед вычислением внимания эмбединги умножаются на обучаемые матрицы W^Q, W^K, W^V . Это нужно для того, чтобы модель могла извлечь из

исходных эмбедингов разные типы информации: отдельно для поиска сходства и отдельно для передачи полезного содержания (6):

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \quad (6)$$

где Q – матрица запросов;

K – матрица ключей;

V – матрица значений;

d_k – размерность ключей.

После этого применяется softmax, который превращает оценки в веса внимания. Затем веса умножаются на V , и получается итоговая информация, которую модель передает дальше.

2.3 Архитектура Decoder-only Transformer

Классическая архитектура Transformer изначально была разработана для задач машинного перевода и состояла из двух частей: энкодера и декодера. Однако современные большие языковые модели, предназначенные для генерации текста, чаще всего используют не всю архитектуру целиком, а только ее декодерную часть. Такой подход называется Decoder-only Transformer, то есть Transformer, состоящий только из декодера.

Главное отличие Decoder-only Transformer от классической архитектуры заключается в отсутствии энкодера и слоя cross-attention. В классическом Transformer энкодер обрабатывает входную последовательность, а декодер с помощью cross-attention обращается к результатам энкодера. В Decoder-only архитектуре отдельного энкодера нет, поэтому вся входная информация передается модели напрямую в виде единой текстовой последовательности. Например, в случае ассистента для проведения собеседований в одну последовательность могут быть объединены системная инструкция, описание вакансии, вопрос, ответ кандидата и требуемый формат результата.

Вместо разделения на входную и выходную последовательность Decoder-only модель работает с одним общим контекстом. Она получает уже

имеющиеся токены и предсказывает следующий токен. Поэтому такие модели называются авторегрессионными. После генерации нового токена он добавляется к исходному контексту, и модель снова предсказывает следующий токен, то есть не генерирует весь ответ сразу, а строит его последовательно, опираясь на уже имеющийся контекст.

Основным механизмом внутри Decoder-only Transformer является masked self-attention. Он необходим для того, чтобы модель при предсказании следующего токена не использовала информацию из будущих токенов. Это особенно важно при обучении, когда в модель может подаваться вся последовательность целиком. Если не использовать маску, модель могла бы видеть правильное продолжение и фактически “подсматривать” ответ. В реальном использовании будущих токенов еще не существует, поэтому обучение без маски привело бы к некорректному поведению модели.

Таким образом, классическая архитектура Transformer является основой для современных LLM, однако в большинстве генеративных языковых моделей используется именно ее декодерная разновидность – Decoder-only Transformer. Она сохраняет ключевые механизмы Transformer, такие как self-attention, multi-head attention, позиционное кодирование, нормализацию и полносвязные слои, но не использует энкодер и cross-attention. Это позволяет модели эффективно решать задачи генерации текста и работать с диалоговым контекстом.

3 Анализ аналогов

На рынке представлены решения для автоматизации финансовых операций. Исходя из быстрого развития финансового сектора и активного содействия со стороны государства можно сделать вывод, что разработка мобильного приложения, предметная область которого представляет сферу оказания услуг интернет-банкинга, является актуальной и востребованной.

Для того, чтобы решить задачу объединения разрозненных сервисов, специализирующихся на оказании платежных услуг, необходимо проанализировать вероятные варианты использования разрабатываемого приложения.

Разрабатываемое приложение относится к классу интеллектуальных ассистентов для структурирования переводов и платежей. Его основная задача заключается в упрощении юридических дел и создании итогового отчета по операциям. Поэтому в качестве аналогов целесообразно рассмотреть системы, которые частично или полностью решают похожие задачи.

Во время исследования рынка финансовых приложений были выявлены следующие программные продукты-аналоги: «СберБанк», «Т-Банк», «Альфа-Банк».

«СберБанк» является мобильным приложением, предоставляющим свои клиентам следующий набор услуг:

- просмотр информации по счетам;
- пополнение счета;
- осуществление переводов между картами и счетами;
- оплата кредитов;
- оплата услуг юридических лиц;
- оплата коммунальных услуг.

Приложение «СберБанк» входит в крупную экосистему «Сбера», которая насчитывает более 40 компаний и сервисов. Для того, чтобы пользовательский опыт взаимодействия с таким количеством услуг не

вызывал неудобств, был введен единый идентификатор клиента. Эта технология упростила обработку платежей и переводов потребителей.

«Т-Банк» – приложение для смартфонов, обладающее простым в использовании интерфейсом. Удобная лента событий отображает историю финансовых операций. Отдельного внимания заслуживает реализация бонусных предложений от партнеров банка. Чтобы получить бонусы в виде кешбэка, нужно выбрать интересующие категории спецпредложений и перейти к покупкам.

«Альфа-Банк» – решение для мобильных устройств, которое предлагает все виды финансовых услуг, включая инвестиции, накопительные счета, кредитование, управление активами и участие в партнерских программах. Присутствуют гибкие инструменты учета средств.

4 Архитектура приложения

Разрабатываемое приложение представляет систему для обработки финансовых транзакций с использованием больших языковых моделей. Приложение поддерживает два основных сценария работы: обычное AI-консультирование, при котором пользователь взаимодействует с ассистентом через панель взаимодействия, и непосредственная работа с большой языковой моделью.

Для реализации приложения была выбрана микросервисная архитектура, которая представлена на рисунке 6. Такой подход позволяет разделить систему на независимые компоненты, каждый из которых отвечает за свою область: авторизацию, формирование отчета, оценку ответов, хранение отчетов и взаимодействие с большой языковой моделью. Преимуществом микросервисной архитектуры является возможность независимо разрабатывать, тестировать, масштабировать и дорабатывать отдельные части системы [19].

Бизнес-логика приложения реализована на языке программирования Java, так как данная технология хорошо подходит для построения надежных backend-сервисов, работы с базой данных направлена в сторону языковой модели, авторизацией и REST API. Компоненты вынесены в отдельные Python-сервисы, так как Python имеет развитую экосистему библиотек для работы с большими языковыми моделями.

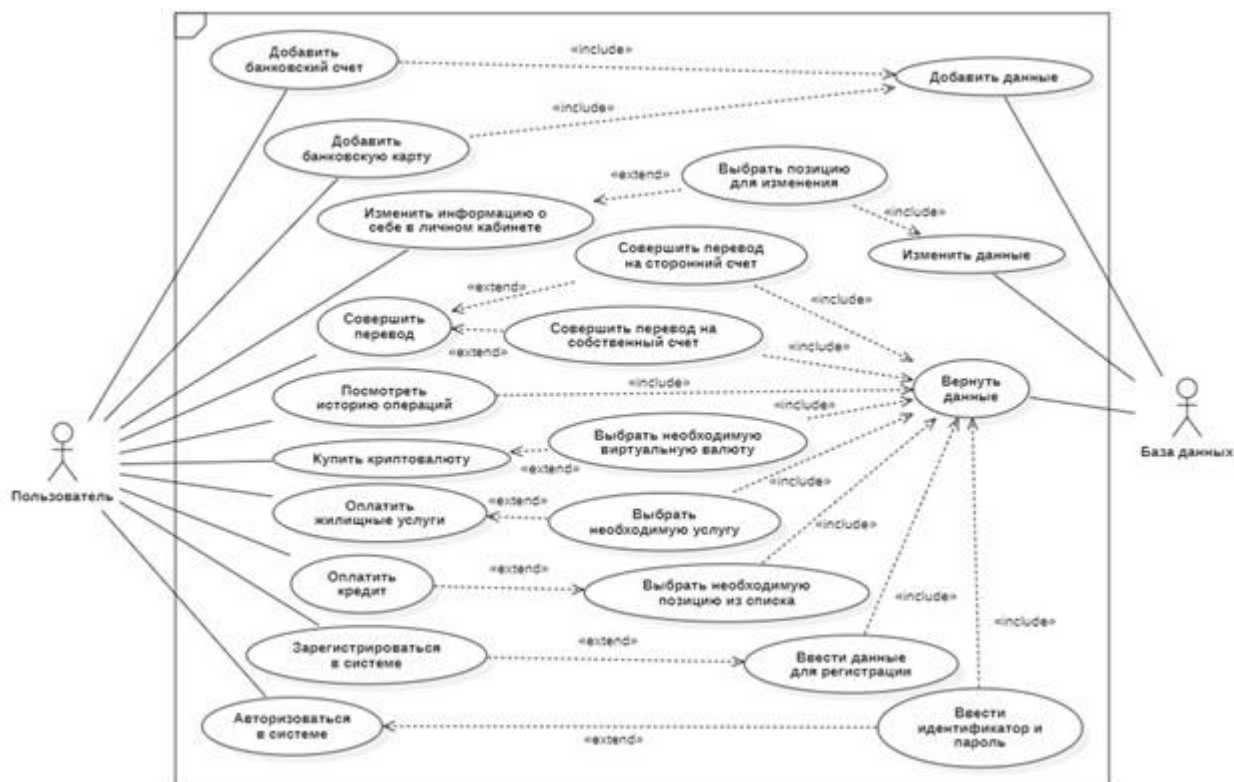


Рисунок 6 - Архитектура приложения

Последовательность выполнения ролью «Пользователь» варианта использования «Зарегистрироваться в системе» играет важную роль в реализации индивидуального подходе к клиентам. В данном приложении формы регистрации предлагают ввести пользователю данные, начиная от его полного имени до выбора вида деятельности. Подробный сбор личной информации помогает составить точный портрет потребителя, что упрощает составление индивидуальных предложений под конкретного клиента.

4.1 Процессы работы со статистикой

Рассмотрим различные сценарии использования системы. Через клиентскую часть пользователь может зарегистрироваться, авторизоваться, получить подсказки от AI-агента, ввести личные инструкции для большой языковой модели, получать обратную связь и скачать отчет по теме.

Пользовательская форма общения с системой не обращается напрямую к каждому микросервису. Все запросы отправляются через API, который выступает единой точкой входа в приложение. Это упрощает структуру клиентской части, так как интерфейс не должен знать внутреннее расположение всех компонентов.

Подробный сбор личной информации помогает составить точный портрет потребителя, что упрощает составление индивидуальных предложений под конкретного клиента. На рисунке 7 представлена последовательность выполнения ролью «Пользователь» варианта использования «Зарегистрироваться в системе»

Прецедент: Зарегистрироваться в системе.
ID: 1
Описание: Пользователь создает новую учетную запись в приложении.
Главные актеры: Пользователь.
Второстепенные актеры: Отсутствуют.
Предусловия: 1. Пользователь открыл мобильное приложение. 2. Пользователь перешел к экрану регистрации.
Основной поток: 1. Пользователь ввел все необходимые данные. 2. Пользователь нажал на кнопку регистрации.
Постусловия: 1. Приложение отображает главный экран новой учетной записи.
Альтернативные потоки: Отсутствуют.

Рисунок 7 – Специализация для варианта использования «Зарегистрироваться в системе»

На рисунке 8 представлена последовательность выполнения ролью «Пользователь» варианта использования «Совершить перевод».

Прецедент: Совершить перевод.
ID: 3
Описание: Пользователь совершает денежный перевод по номеру счета.
Главные актеры: Пользователь.
Второстепенные актеры: Отсутствуют.
Предусловия: 1. Пользователь открыл выпадающее окно перевода средств.
Основной поток: 1. Пользователь выбрал счет для перевода. 2. Пользователь выбрал с какого банковского счета будут списаны деньги. 3. Пользователь ввел сумму перевода. 4. Пользователь подтвердил перевод.
Постусловия: 1. Приложение перевело денежные средства по номеру счета. 2. Приложение задокументировало перевод денег в истории операций.
Альтернативные потоки: Отсутствуют.

Рисунок 8 – Спецификация для варианта использования «Совершить перевод»

На рисунке 9 приведено описание порядка выполнения ролью «Пользователь» варианта использования «Купить криптовалюту».

Прецедент: Купить криптовалюту.
ID: 4
Описание: Пользователь совершает покупку криптовалюты.
Главные актеры: Пользователь.
Второстепенные актеры: Отсутствуют.
Предусловия: 1. Пользователь открыл окно покупки криптовалюты.
Основной поток: 1. Пользователь выбрал необходимую ему криптовалюту. 2. Пользователь выбрал нужное количество виртуальной валюты. 3. Пользователь подтвердил приобретение.
Постусловия: 1. Приложение добавило требуемое количество криптовалюты в криптовалютный кошелек пользователя. 2. Приложение задокументировало покупку криптовалюты в истории операций.
Альтернативные потоки: Отсутствуют.

Рисунок 9 – Спецификация для варианта использования «Купить криптовалюту»

В результате разработки сценариев использования были смоделированы бизнес-процессы, позволяющие обслуживать пользователей финансовой сферы.

4.2 Архитектура клиент-сервер

При создании мобильного приложения была использована многоуровневая архитектура. Схематичное представление данной архитектуры приведено на рисунке 10.

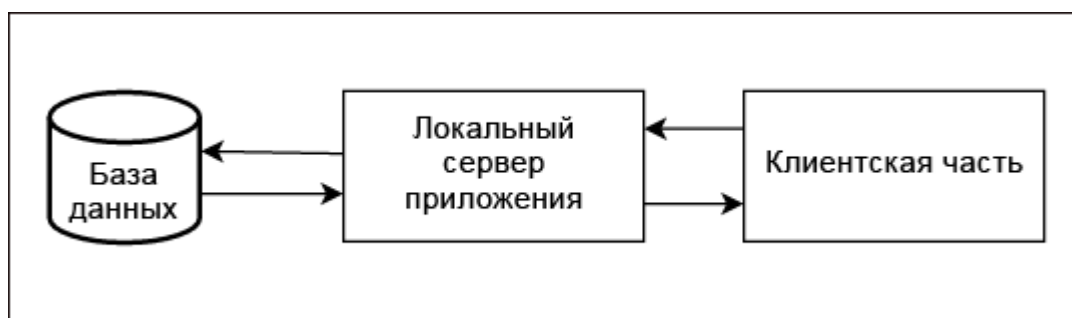


Рисунок 10 – Архитектура клиент-сервер

Многоуровневая архитектура содержит три слоя:

- презентационный слой,
- слой бизнес-логики,

– слой доступа к данным.

Презентационный слой представляет визуальный функционал приложения. При создании этого слоя разработчики программного обеспечения уделяют особое внимание дизайну пользовательского интерфейса. Работа над дизайном приложения заключается в выстраивании интуитивно понятной навигации сервиса, создании визуальных компонентов взаимодействия. Слой бизнес-логики отвечает за обработку событий и обмен данными. На этом уровне система выполняет сценарии ее использования, к ним относятся: кэширование, передача данных, обработка исключений, выполнение различных операций. Слой доступа к данным отвечает за обработку и сохранение информации. Компоненты этого слоя обеспечивают функциональный доступ к данным.

Передача состояний от презентационного слоя к слою бизнес-логики осуществляется по протоколу HTTP при помощи REST API.

4.3 Проектирование структуры базы данных

При проектировании структуры базы данных были рассмотрены возможности современных подходов к разработке хранилищ информации, отвечающих за взаимодействие с ней. К одному из таких подходов относится составление данных в виде таблиц. Базы такого типа называются реляционными. Перед использованием данного решения необходимо заранее определить схему, по которой программа будет записывать информацию. Особенность этого подхода ограничивает гибкий подход к разработке программного обеспечения, усложняет структуру кода и снижает скорость работы программы. Противоположностью данного решения являются нереляционные базы данных. В них отсутствуют ограничения формирования информации, что позволяет быстро адаптировать базу к меняющимся типам данных в конкретный момент работы приложения.

Основываясь на анализе возможностей нереляционных баз данных, выбрана платформа с открытым исходным кодом, которая называется Parse android. Этот сервис решает проблему написания серверной части вручную.

Программное решение Parse android позволяет разработчикам быстро создавать мобильные приложения с общими данными, без написания пользовательских API. Подход к составлению логически структурированных моделей данных в Parse android отличается от реляционного подхода к формированию хранилищ информации. Данная платформа позволяет добавлять произвольные поля информации без предварительной декларации изменения всей таблицы. Любое изменение состояния данных в базе сводится к редактированию структуры кода, отвечающего за взаимодействие с ней.

Parse android – это программа, которая создает интерфейс для автоматического сбора и хранения информации. Благодаря объединению инструментов разработки для мобильных устройств и гибкой серверной части Parse android обладает следующими возможностями:

- регистрация пользователя с последующей аутентификацией,
- создание, изменение и удаление данных,
- упрощение отправки уведомлений,
- загрузка файлов на сервер для общего доступа.

4.4 Проектирование программных средств

На основном экране необходимо реализовать добавление банковских счетов и функцию перевода денежных средств между ними, оформление кредитных карт, просмотр истории операций.

Для оформления кредита клиенту нужно будет ввести необходимую сумму и срок погашения займа в месяцах.

От экрана с криптовалютой требуется вывод доступных позиций виртуальной валюты, предоставление возможности покупки необходимого количества цифровых активов. Купленная криптовалюта будет привязана к данным конкретного пользователя, что упростит ее дальнейшее использование на других сервисах.

Чтобы упростить документирование оплаченных жилищно-коммунальных услуг, следует сохранять на экране оплаты счетов электронные чеки с названием услуги, суммой оплаты и датой оформления.

В личном кабинете должна присутствовать возможность изменения персональной информации. База данных, использующая хэш-таблицы, позволит быстро добавить внесенные корректировки.

На каждом этапе работы приложения с данными, функции, отдельно взятых классов, будут искать, добавлять или удалять пары данных вида «ключ-значение». Такой подход минимизирует риск потери информации при сбоях интернет-соединения. Краткая спецификация компонентов программного продукта представлена на рисунке 11.

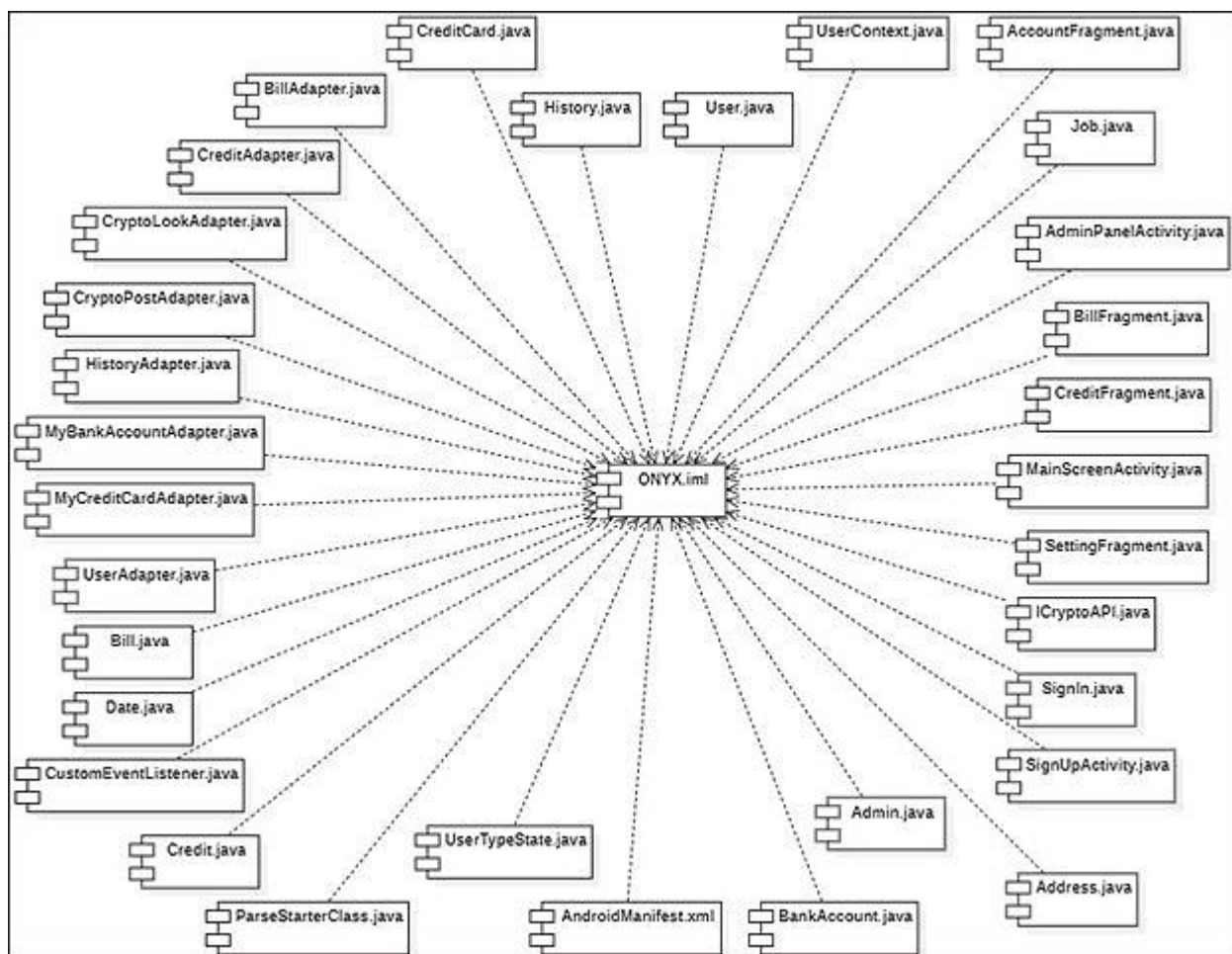


Рисунок 11 – Диаграмма компонентов

Класс SignUpActivity реализует функции записи вводимой клиентом информации. Каждому новому пользователю присваивается идентификатор учетной записи, который служит ключом ко всем остальным его данным. Класс SignIn проверяет пароль и идентификатор при входе в систему. За отображение навигационной панели внизу экрана устройства и переходы между активностями отвечает класс MainScreenActivity. Верхняя панель

главного экрана отображает полное имя пользователя, общую сумму денежных средств, текущую дату. За вывод этой информации и предоставление основных финансовых услуг отвечает класс AccountFragment. Класс CreditFragment отвечает за оформление и оплату кредита. Проверку параметров подключения к стороннему источнику информации за актуальным курсам криптовалюты отвечает класс ICryptoAPI. Формированием списка доступных криптовалют занимается класс CryptoLookAdapter. Покупку виртуальной валюты реализует класс CryptoPostAdapter. Используя логику класса BillAdapter, класс BillFragment реализует оплату и документирование жилищных услуг. За доступ к данным со стороны администрации сервиса отвечает класс AdminPanelActivity. Класс ParseStarterClass выполняет обработку и сохранение данных в базе. Работа с функционалом личного кабинета происходит в классе UserAdapter. За настройку кабинета отвечает класс UserAdapter.

4.5 Проектирование пользовательского интерфейса

Исходя из диаграммы вариантов использования, был разработан графический интерфейс приложения. Пользовательский интерфейс должен быть прост и понятен в использовании.

Для проектирования визуальной части сервиса был выбран многофункциональный редактор растровой графики Adobe Photoshop. Редактор обладает большими возможностями в сфере проектирования и верстки программных продуктов. Широкий выбор инструментов данной программы позволил спроектировать внешний вид клиентской части, еще до ее функциональной реализации, что позволило сократить время разработки.

В данном программном обеспечении знакомство с сервисом начинается с экрана авторизации и регистрации. На рисунке 12 показан экран авторизации.



Рисунок 12 – Экран авторизации

Основные элементы интерфейса, показанного на рисунке 12:

1 – поле ввода идентификатора пользователя, передающее в базу данных уникальное числовое значение учетной записи клиента,

2 – поле ввода пароля от аккаунта,

3 – кнопка, подтверждающая вход,

4 – кнопка перехода к экрану регистрации,

На рисунке 13 изображен интерфейс регистрации.



Рисунок 13 – Экран регистрации

Элементы экрана регистрации рисунка 13:

- 1 – поле ввода полного имени пользователя,
- 2 – поле ввода идентификатора учетной записи,
- 3 – поле ввода пароля,
- 4 – поле ввода номера телефона,
- 5 – выпадающий список полей ввода адреса проживания клиента,
- 6 – список, предлагающий выбрать один из видов деятельности,
- 7 – кнопка, подтверждающая регистрацию.

Главный экран приложения представлен на рисунке 14. Эта часть сервиса выполняет основные услуги интернет-банкинга. Удобное разделение функциональных частей этого экрана обеспечивает простоту использования возможностей приложения.

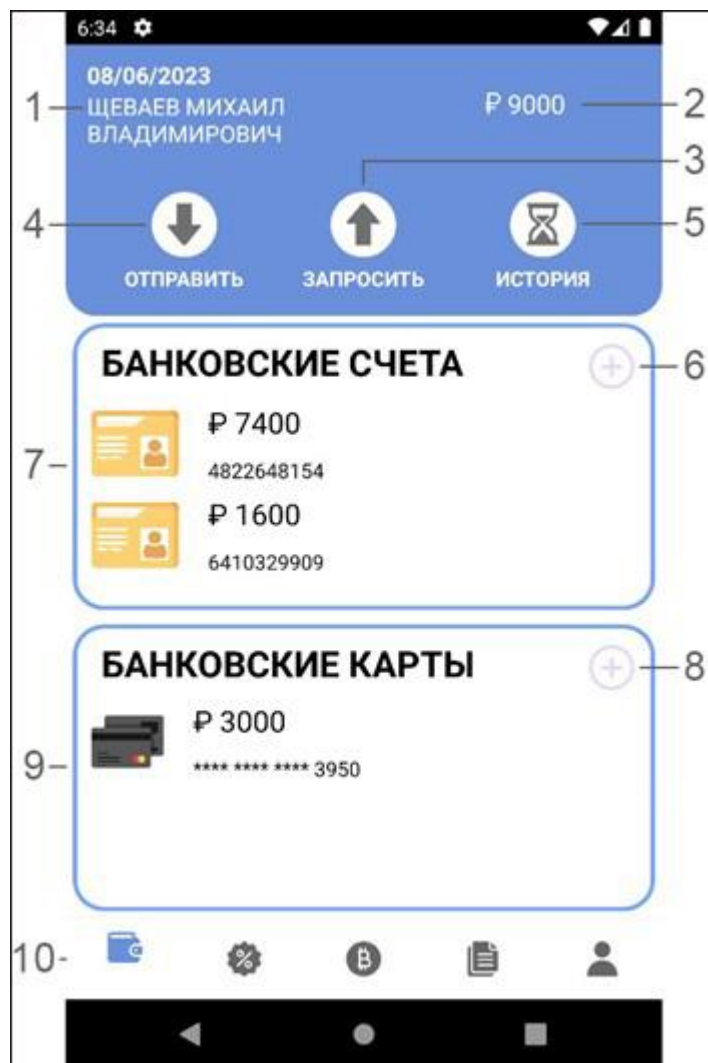


Рисунок 14 – Главный экран

К элементам главного экрана разрабатываемого приложения, представленного на рисунке 14, относятся:

- 1 – общая информация,
- 2 – совокупность денежных средств всех банковских счетов пользователя,
- 3 – кнопка запроса денежных средств,
- 4 – кнопка, позволяющая перевести деньги на собственный и сторонний счет,
- 5 – кнопка, позволяющая посмотреть историю операций,
- 6 – кнопка добавления нового банковского сета,
- 7 – раздел банковских счетов,
- 8 – кнопка, позволяющая добавить новую банковскую карту,

9 – раздел банковских карт,

10 – панель навигации по основным экранам приложения.

Рассмотрим подробное проектирование процесса перевода средств стороннему пользователю. Моделирование процессов данного функционала играет важную роль в формировании общего сценария совершения финансовых транзакций между клиентами системы.

На рисунке 15 показано окно выбора действия. Пользователь может перевести денежные средства со одного счета на другой.

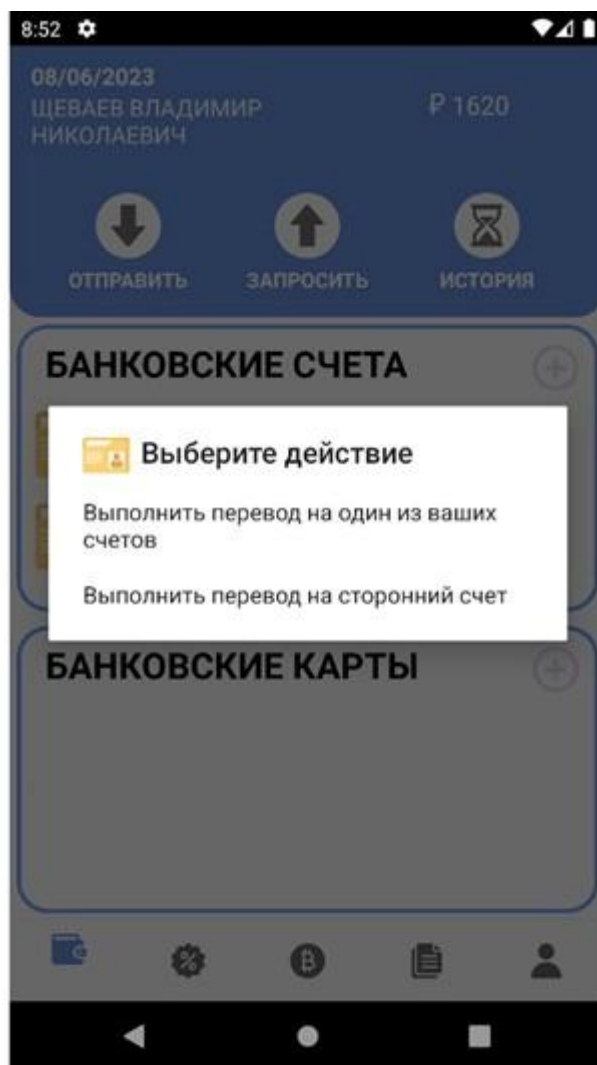


Рисунок 15 – Окно выбора действия

Программируя процесс перевода, средствами среды разработки, ограничим ввод данных цифровым значением. Внешний вид данной процедуры изображен на рисунке 16.

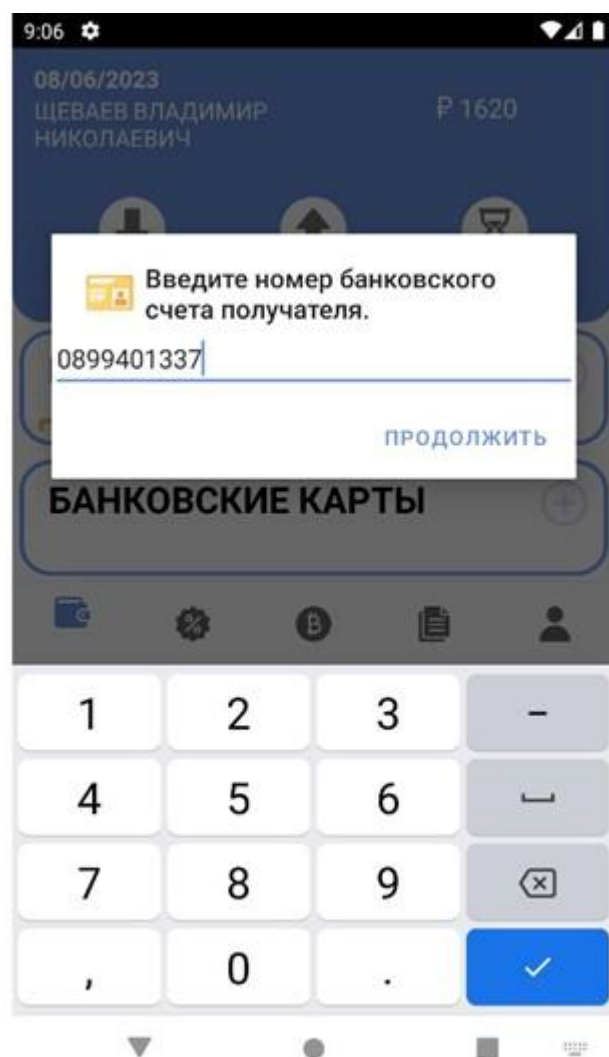


Рисунок 16 – Процесс ввода номера счета

Для улучшения эргономики, разрабатываемого интерфейса, смоделируем внешний вид выбора пользователем банковского счета. Окна цепочки событий перевода денежных активов не должны вызывать затруднений в использовании. Для этого ограничим количество необходимых действий на каждом этапе. Индивидуальный подход к клиентам можно реализовать, используя вопросительные предложения. Так приложение, построенное на жесткой логике бизнес-процессов, станет верным помощником в решении финансовых операций. Один из примеров данного подхода представлен на рисунке 17.

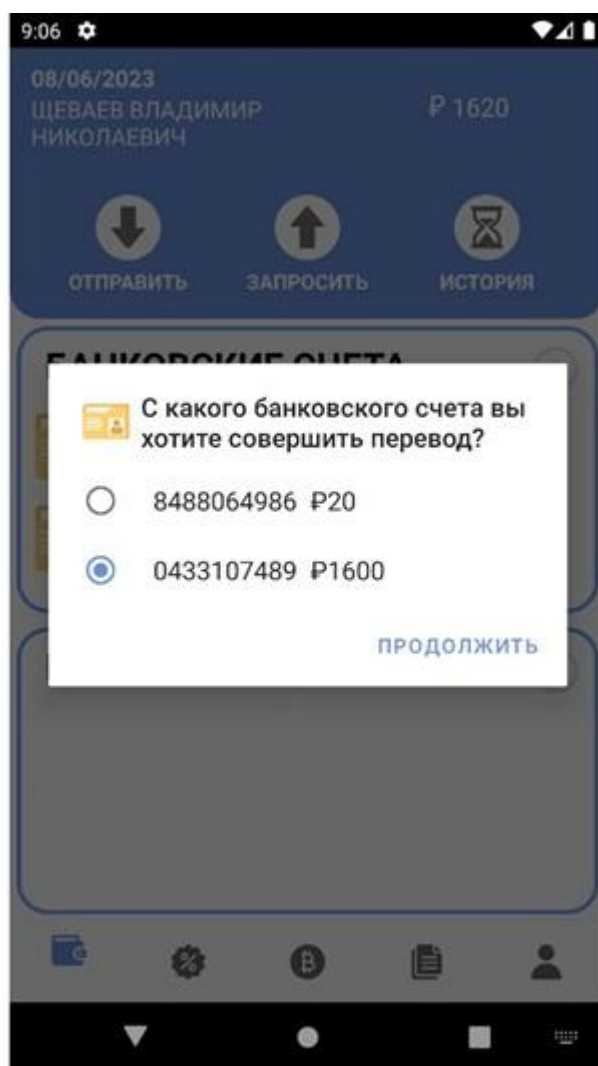


Рисунок 17 – Процесс выбора одного из счетов

Возможности среды разработки Android Studio предоставляют доступ к настройке отображения графических ресурсов. Встроенные стили и настройка компонентов создают простор для реализации шаблонов, созданных в графических редакторах. Смоделируем внешний вид процесса ввода переводимой суммы. Для этого воспользуемся ограничением количества действий пользователя на одном экране. Изобразим цифровую клавиатуру операционной системы Android, добавим поле ввода, обозначим переход к следующему действию кнопкой с соответствующей надписью. Затемним задний фон, оставив в зоне видимости основную информацию. Подробное представление проектируемого процесса показано на рисунке 18.

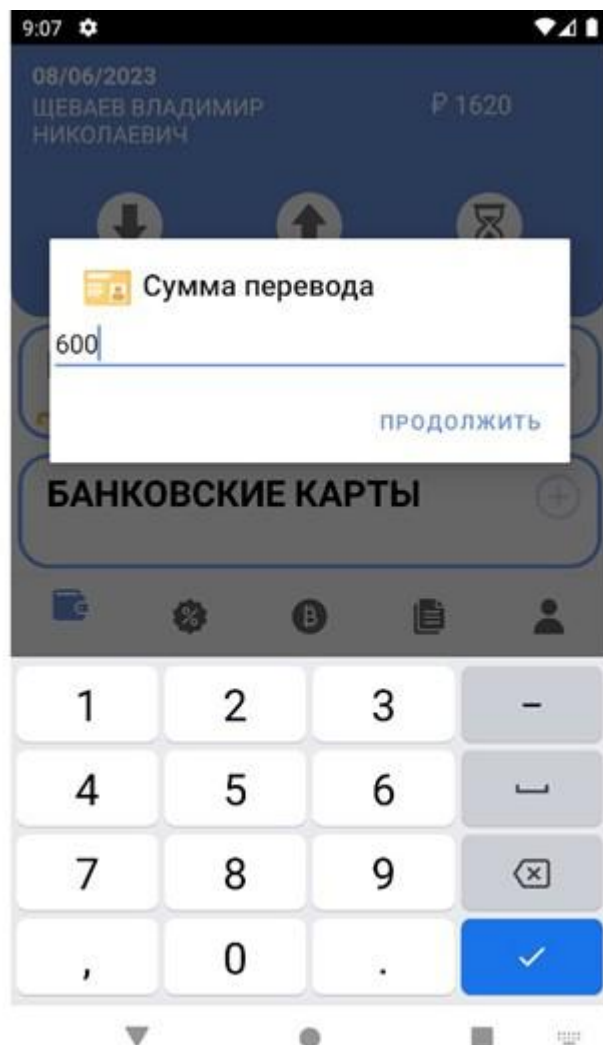


Рисунок 18 – Процесс ввода суммы перевода

На главном экране спроектируем списание денежных средств после процедуры перевода. Для этого добавим текст суммы за вычетом отправленных активов. Изменим баланс банковского счета, участвовавшего в процедуре перевода.

Важно помнить, что все процессы передачи данных аккаунта пользователя должны проходить обработку серверной частью и выводиться на стороне клиента. Одним из инструментов отображения подобной информации является окно истории операций. В этом окне должно выводиться название операции и дата ее проведения. В приложении необходимо реализовать возможность пролистывания списка истории до самой первой операции.

Окно истории изображено на рисунке 19. Пользователь может просмотреть список, интересующих его операций.



Рисунок 19 – История операций

В среде растрового редактора смоделируем результат работы клиентской и серверной части. От этой цепочки событий ожидается результат в виде зачисления денежных средств на банковский счет, номер которого был использован в переводе. Программная реализация этого графического представления должна будет заключаться в синхронизации результат операции между двумя аккаунтами. Для наглядной демонстрации успеха операции, создадим профиль получателя. Отобразим общую сумму со всех счетов данного пользователя и изобразим внешний вид окна банковских счетов. Изображение конечного этапа процесса перевода денежных средств представлено на рисунке 20.

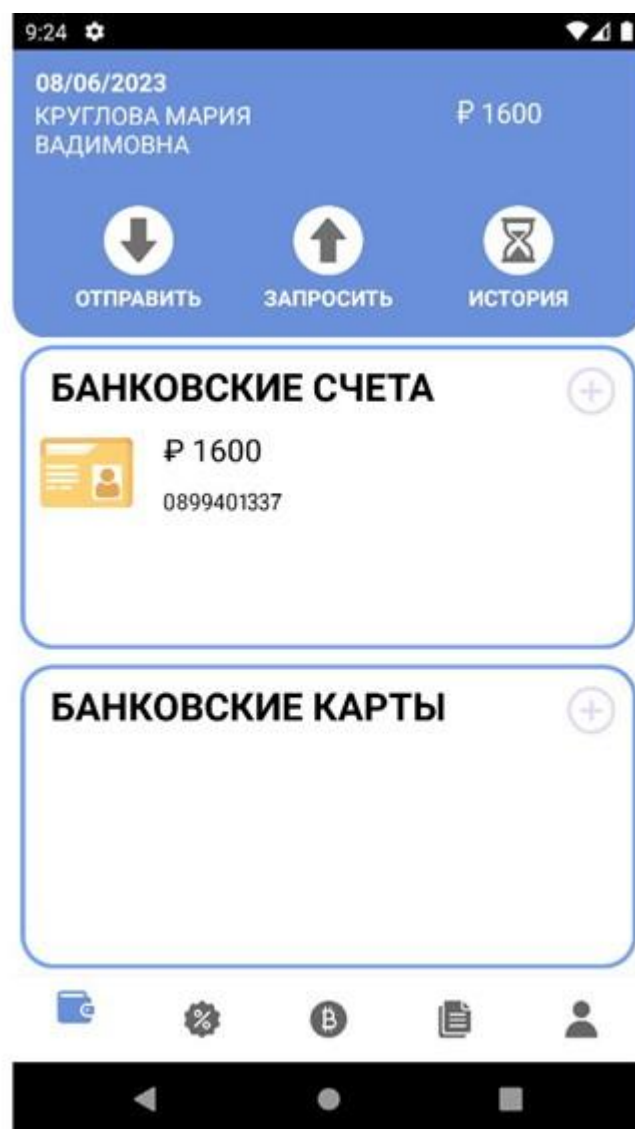


Рисунок 20 – Главный экран получателя денежного перевода

Одной из услуг разрабатываемого приложения является кредитование. Для реализации данного бизнес-процесса требуется спроектировать отдельный экран с удобной и понятной навигацией. Кнопка в правом нижнем углу данного раздела должна будет выводить окно оформления кредита. Важно отобразить максимальные значения суммы займа и срока выплаты, оформить поля ввода необходимой суммы и количества месяцев, отведенных на погашение кредита. Подтверждение оформления обозначить кнопкой. Экран кредитования изображен на рисунке 21.

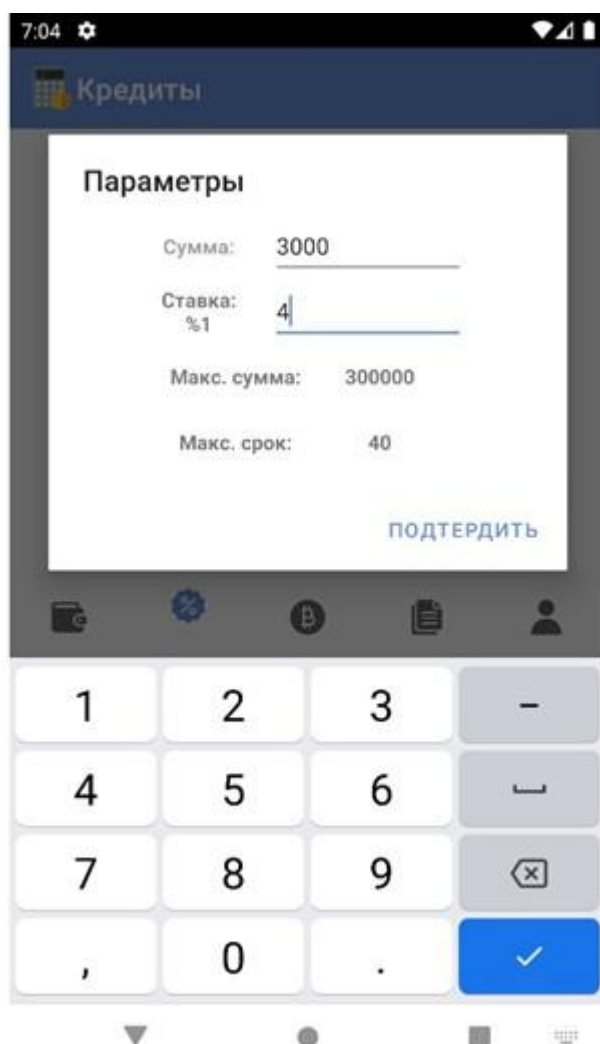


Рисунок 21 – Экран оказания услуг кредитования

Ключевой особенностью услуги кредитования является формирование общей суммы кредита, исходя из рода деятельности клиента приложения. Во время регистрации пользователь может выбрать вид текущей занятости, что определит величину процентной ставки. Если клиент сменит свой вид деятельности, то изменить данный пункт он сможет в личном кабинете. Вслед за этим значение процентной ставки тоже измениться. Данная функция удобна тем, что пользователь заранее может рассчитать свои возможности относительно заработка.

На рисунке 22 изображено окно кредитования с пронумерованными элементами графического интерфейса.

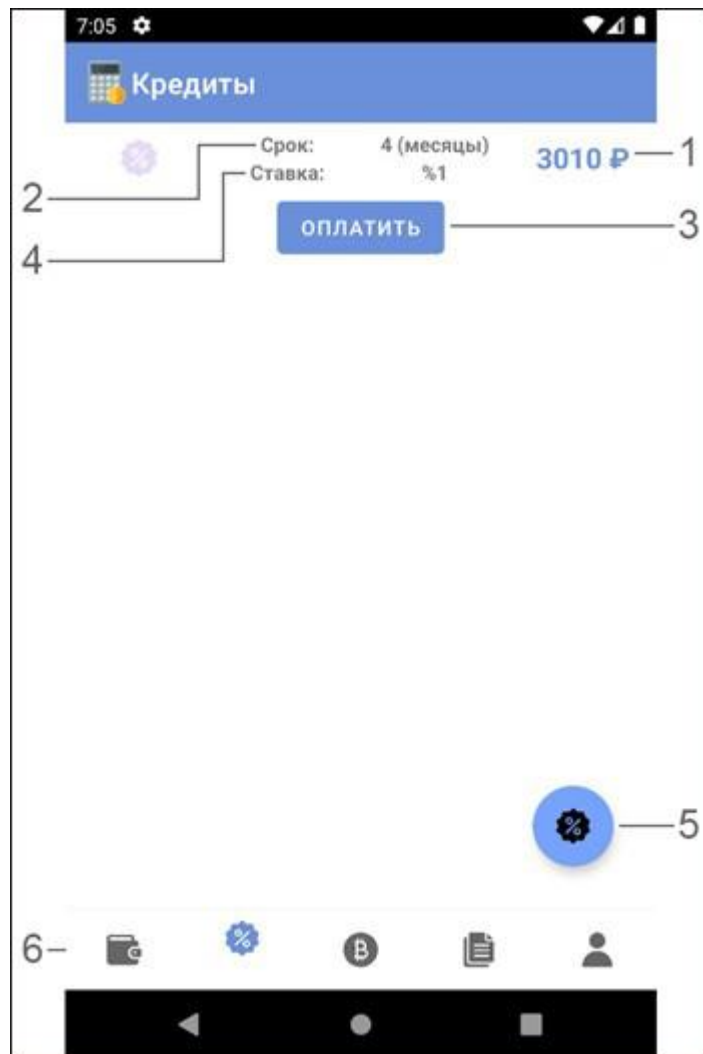


Рисунок 22 – экран кредитования

Основные элементы экрана кредитования:

- 1 – общая сумма выплаты,
- 2 – срок погашения кредита,
- 3 – кнопка оплаты кредита,
- 4 – значение процентной ставки,
- 5 – кнопка, отвечающая за оформление кредита,
- 6 – панель навигации по основным экранам приложения.

За покупку и хранение виртуальных активов отвечает функционал экрана криптовалюты. Рисунок 23 демонстрирует оформление этого раздела программы.



Рисунок 23 – экран криптовалют

Для современного клиента оплата жилищно-коммунальных услуг стала естественным процессом. Современные технологии упростили это занятие, высвободив большой объем времени. Перед реализацией данного функционала создадим графическое представление окна оплаты счетов. Спроектируем окно выбора необходимой услуги с указанием суммы оплаты, добавим кнопку для открытия списка услуг. Создадим шаблон вывода на экран оплаченных и задокументированных услуг.

Раздел оплаты и документирования коммунальных услуг представлен на рисунке 24.

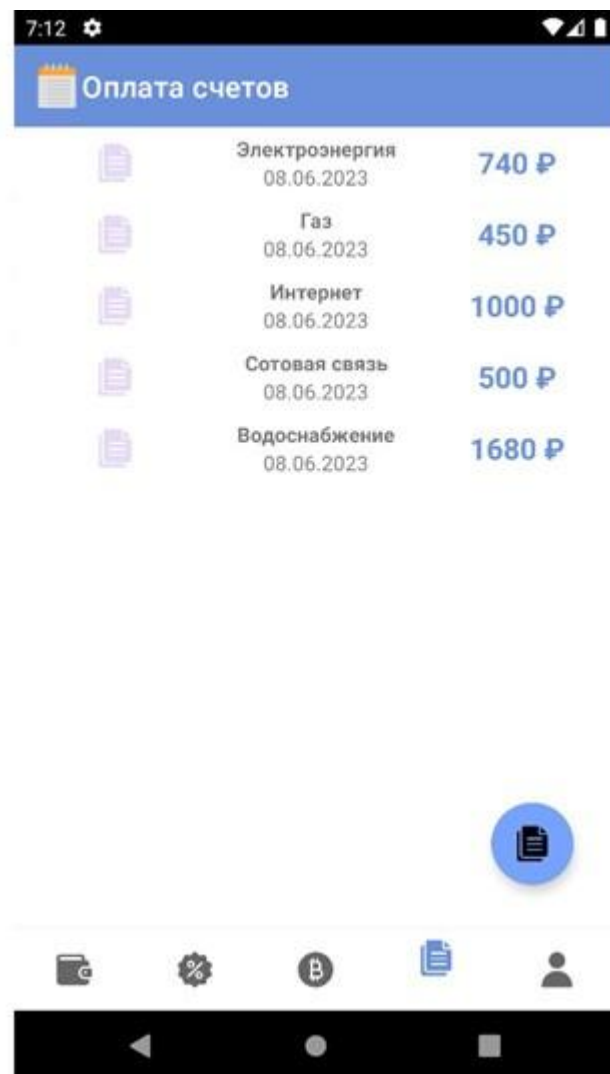


Рисунок 24 – Экран оплаты счетов

Важным элементом любого финансового приложения является личный кабинет пользователя. В нем он всегда может проверить и изменить интересующую его информацию. Данный элемент системы тесно связан с хранилищем данных. Планируется вывод информации из файлов типа JSON в соответствующие поля на экране. Редактирование будет реализовано в отдельных окнах. После изменения или отмены изменения необходимо организовать вывод информационного сообщения о результате действия. Предполагается отображение уникального идентификатора клиента на экран устройства.

На рисунке 25 представлен доступный функционал редактирования пунктов личной информации.

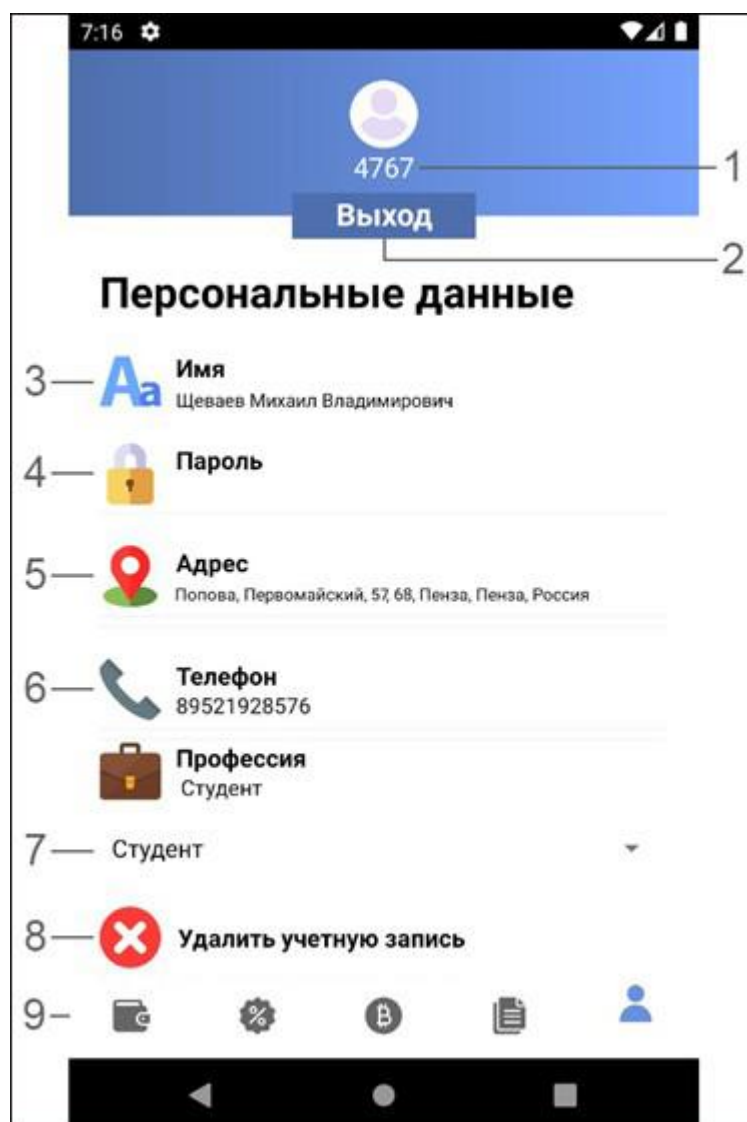


Рисунок 25 – Личный кабинет пользователя

Основные элементы экрана личного кабинета пользователя:

- 1 – поле редактирования уникального идентификатора учетной записи,
- 2 – кнопка выхода из аккаунта,
- 3 – поле редактирования полного имени пользователя,
- 4 – поле, которое служит для изменения пароля,
- 5 – поле редактирования адреса проживания,
- 6 – поле для редактирования номера телефона,
- 7 – поле выпадающего списка, которое позволяет сменить вид деятельности,
- 8 – кнопка, отвечающая за удаление учетной записи из базы данных,
- 9 - панель навигации по основным экранам приложения.

Исходя из вышеописанной реализации моделей проектирования пользовательского интерфейса, можно видеть, что графическое представление проекта реализует все функциональные требования разрабатываемого продукта большой языковой модели.

Благодаря своевременному проектированию графического интерфейса, удалось снизить трудозатраты на разработку, что значительно упростило процесс работы над клиентской частью. Использование низкоуровневых шаблонов проектирования, основанных на технологиях обработки данных платформы Parse android, позволило реализовать весь заявленный функционал.

5 Контроль качества программного продукта

Для реализации данного приложения, было произведено тестирование черным ящиком, заключающееся в том, что вместо свойств составных частей системы, проверялась ее реакция на изменение условий эксплуатации. Под стратегией данного подхода к тестированию понимается создание тестов для тестовых наборов.

5.1 Классы эквивалентности

Под классами эквивалентности понимается набор входных данных, который обрабатывается программой по одному алгоритму или приводит к единственному результату. Для разрабатываемого приложения классы эквивалентности представлены в таблице 1.

	Верные классы эквивалентности	Неверные классы эквивалентности
Длина пароля	5-12 (1)	0-4 (2) 13 - ∞ (3)
Структура пароля	Начинается со специального символа (4)	Начинается не со специального символа (5)
Содержание пароля	Цифры (6) Латинские буквы (7) Символы (8)	Остальные символы (9)

Данные в таблице 1 необходимы для корректной работы метода проверки пароля от учетной записи.

5.1.1 Граничные значения

Граничное тестирование использует принцип, при котором один класс эквивалентности переходит в другой. Именно на границе перехода чаще всего возникают ошибки. К граничным значениям относятся крайние значения, начальное-конечное, нижнее-верхнее и максимальное-минимальное. Тип и описание граничных значений для разрабатываемого продукта представлены в таблице 2.

Таблица 2 – Граничные значения

Границы	Описание
Физические	Поле не может быть пустым
Технологические	Ограничения оперативной памяти устройства
Произвольные	Пароль не более 12 символов

Эти граничные условия помогут избежать сбоев системы на переходах функционала одного класса к другому.

5.1.2 Тест-кейсы

По созданным классам эквивалентности и граничным условиям был создан набор тест-кейсов. Такой подход к тестированию представляет собой алгоритм действий, по которому проводится тестирование уже написанной программы. Тест-кейсы представлены в таблице 3.

Таблица 3 – Набор тест-кейсов

№	Класс эквивалентности	Входные данные	Выходные данные
1	Все правильные (1,4,5,8,9,10)	4767	Успешно
2	(2,4,5,8,10)	46Sq67	Недопустимый идентификатор
3	(3,4,5,8,9,10)	Wrt22@34opt9wq12Xcb	Больше 12 символов
4	(1,5,6,8,9,10)	@23Wp40_1	Ошибка, начинается со специального символа

Написанные тест-кейсы помогут выявить недостатки работы системы и предотвратят сбои в ее работе.

5.1.3 Функциональное тестирование белым ящиком

Тестирование программного обеспечения белым ящиком – это метод тестирования, который предполагает, что структура кода и особенности системы заранее известны тестировщику. При создании тест-кейсов важно избегать любых зависимостей, потому что один из них может быть удален из-за ненужности, тогда сломается логика проверки зависимых от него тест-кейсов. Результаты функционального тестирования регистрации и авторизации приведены в таблице 4.

Функциональное тестирование показало корректность работы клиентской части мобильного приложения. Проверку прошел функционал регистрации и авторизации пользователя в системе. Это стандартизированный сценарий проверки, который помогает систематизировать процесс тестирования, обеспечить его воспроизводимость и объективную оценку соответствия системы требованиям. Данный механизм необходим для работы с системой, состоящей из большой языковой модели и AI-агента.

Таблица 4 – Функциональное тестирование

Вариант использования	Тест и его входные параметры	Результат
Зарегистрироваться в системе	Ввод верных данных: Имя пользователя: Щеваев Михаил Владимирович; Идентификатор: 4767; Пароль: As@45Tm80; Номер телефона: 89521928576; Адрес: Попова, Первомайский, 57 68, Пенза, Россия.	Тест пройден
	Ввод занятого идентификатора: Имя пользователя: Щеваев Михаил Владимирович; Идентификатор: 4790; Пароль: As@45Tm80; Номер телефона: 89521928576; Адрес: Попова, Первомайский, 57 68, Пенза, Россия.	Тест пройден
	Ввод неверного пароля: Имя пользователя: Щеваев Михаил Владимирович; Идентификатор: 4767; Пароль: As@45Tm80; Номер телефона: 89521928576; Адрес: Попова, Первомайский, 57 68, Пенза, Россия.	Тест пройден
	Ввод некорректного номера телефона: Имя пользователя: Щеваев Михаил Владимирович; Идентификатор: 4767; Пароль: As@45Tm80; Номер телефона: 89521928576; Адрес: Попова, Первомайский, 57 68, Пенза, Россия.	Тест пройден
	Ввод некорректного адреса проживания: Имя пользователя: Щеваев Михаил Владимирович; Идентификатор: 4767; Пароль: As@45Tm80; Номер телефона: 89521928576; Адрес: Попова, Первомайский, 57 68, Пенза, Россия.	Тест пройден
Авторизоваться в системе	Ввод верных данных: Идентификатор: 4767; Пароль: As@45Tm80.	Тест пройден
	Ввод неверного идентификатора: Идентификатор: 4767; Пароль: As@45Tm80.	Тест пройден
	Ввод неверного пароля: Идентификатор: 4767; Пароль: As@45Tm80.	Тест пройден

Функциональное тестирование показало корректность работы клиентской части мобильного приложения. Проверку прошел функционал регистрации и авторизации пользователя в системе.

5.2 Общий сценарий программных средств

Общий сценарий работы программных средств для обработки и анализа событий представлен в виде диаграммы деятельности на рисунке 26.



Рисунок 26 – Сценарий работы программных средств (диаграмма деятельности)

Алгоритм PrefixSpan работает, формируя возможные последовательности через анализ префиксных цепочек и проектирование соответствующих им постфиксных подпоследовательностей в базе данных.

Процесс начинается с загрузки данных и создания пустого списка последовательностей, в который постепенно включаются элементы из исходного набора. Затем алгоритм ищет все потенциальные последовательности, начинающиеся с каждого элемента списка. На следующем этапе генерируются различные последовательности: каждая новая цепочка продолжается добавлением элемента к уже существующим последовательностям. Далее анализируется поддержка каждой сформированной последовательности и сравнивается с установленным порогом. Те последовательности, которые соответствуют критериям и имеют общую начальную часть, сгруппировываются и добавляются в список возможных решений. Процесс генерации продолжается до тех пор, пока не будут идентифицированы все доступные вероятные последовательности.

5.2.1 Алгоритм SPAM

Алгоритм начинает свою работу с анализа входных данных и далее производит подсчет поддержки для определения часто встречающихся элементов. Затем осуществляется создание последовательностей, которые начинаются с текущего префикса. После вычисления поддержки для каждой из этих последовательностей производится проверка их соответствия минимально установленному порогу. В последующем этапе генерируются кандидаты, которые завершаются текущим префиксом, и также оцениваются их показатели поддержки в сравнении с минимальным требованием. Завершающим этапом является объединение различных последовательностей в более длинные цепочки. Работа алгоритма прекращается, когда больше невозможно выделить новые вероятные последовательности. Работа алгоритма SPAM представлена на рисунке 27. Кадры сокращаются в каждом проходе алгоритма.

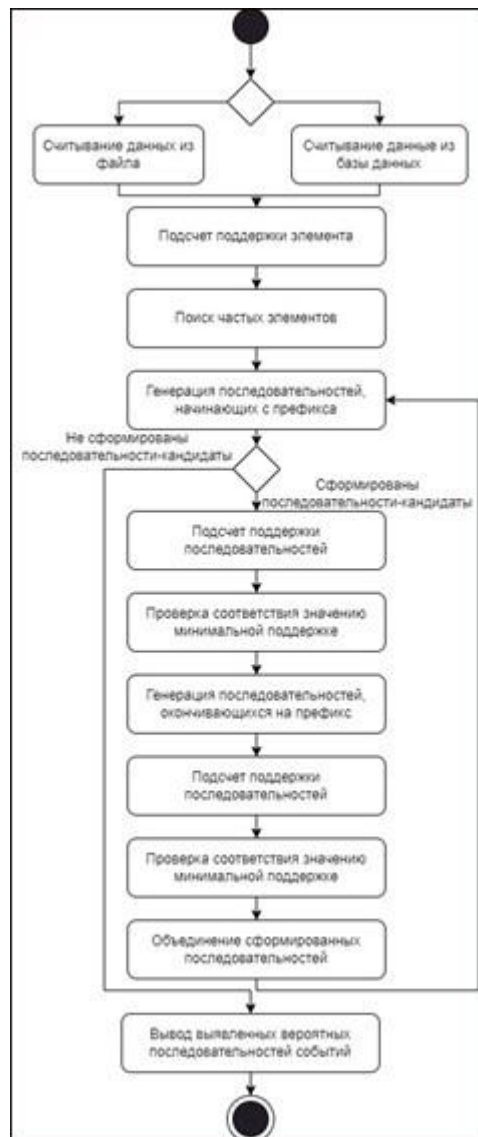


Рисунок 27 – Работа алгоритма SPAM

Идея алгоритма SPAM заключается в сокращении генерации кандидатов, благодаря этому он эффективно выявляет вероятные последовательности событий в больших наборах данных.

5.2.2 Альфа-алгоритм

Функция, запускающая альфа-алгоритм принимает на вход список транзакций и не имеет никаких параметров. Его работа заключается в поочередном переборе всех событий одного случая и определения типа связи всех его переходов. Описание работы алгоритма представлено в блок-схеме на рисунке 28.

При получении журнала событий он проходит предварительную обработку для выявления последовательностей событий. На основе этих

последовательностей определяются типы связей между различными событиями.

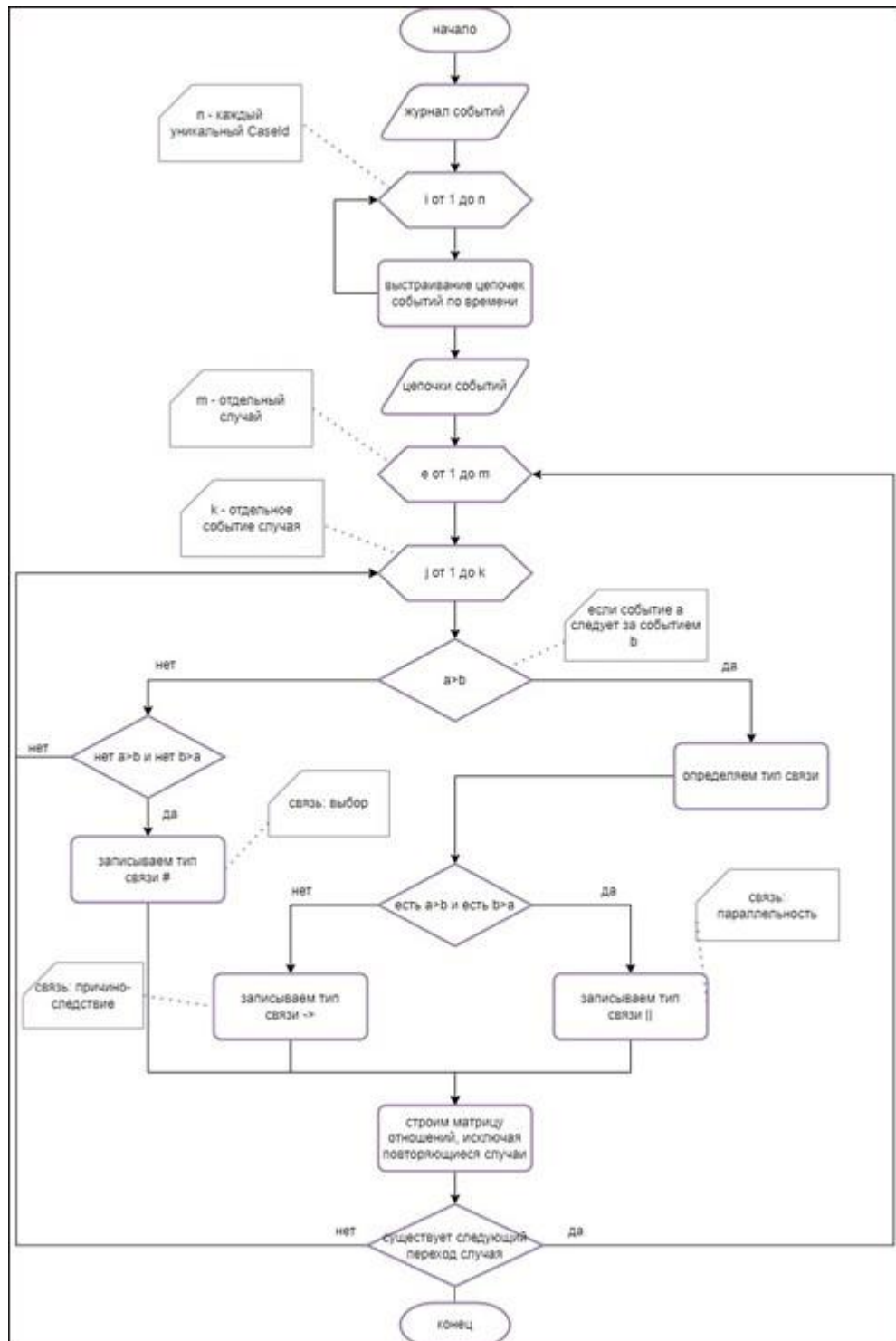


Рисунок 28 – Работа альфа-алгоритма

На основе этих последовательностей определяются типы связей между различными событиями. Последовательно анализируются пары событий (a и b), где проверяется наличие между ними логической связи. Если таковой не

обнаруживается, анализируется обратная связь (b и a), и при её наличии устанавливается тип связи как "выбор". В случае, если между событиями существует однонаправленная связь и отсутствует обратная, определяется тип связи как "причина-следствие".

Если обнаруживается двунаправленная связь между событиями (a и b, а также b и a), это указывает на наличие параллельной связи между ними.

На каждом этапе анализа формируется и корректируется матрица связей, при этом исключаются дублирующие переходы. По завершении обработки всех событий и связей получается окончательная матрица.

5.2.3 Эвристический алгоритм

Функция, которая запускает эвристический алгоритм, принимает на вход журнал событий и установленные параметры порога зависимости. Порог зависимости используется для отсеивания шумовых данных и его значение может колебаться в диапазоне от 0 до 1. Если значение порога приближается к 1, то в итоговой матрице будет меньше событий.

Эвристический алгоритм схож с методом альфа-правил для определения типов связей между событиями. В его структуре предусмотрены дополнительные внутренние циклы и проверки, направленные на оценку вероятности наличия связи между различными событиями.

Алгоритм описан с использованием блок-схемы, изображенной на рисунке 29. Первоочередная задача алгоритма заключается в формировании матрицы связей, одновременно вычисляя для каждого потенциального перехода уровень уверенности в его существовании. Переходы, значение уверенности для которых не превышает установленного порога зависимости, исключаются из рассмотрения и классифицируются как шумовые.

Отдельное условие появляется для неявных переходов. После построения матрицы, очищенной от шума, проверяются и подсчитываются вероятности для переходов с неявным значением.

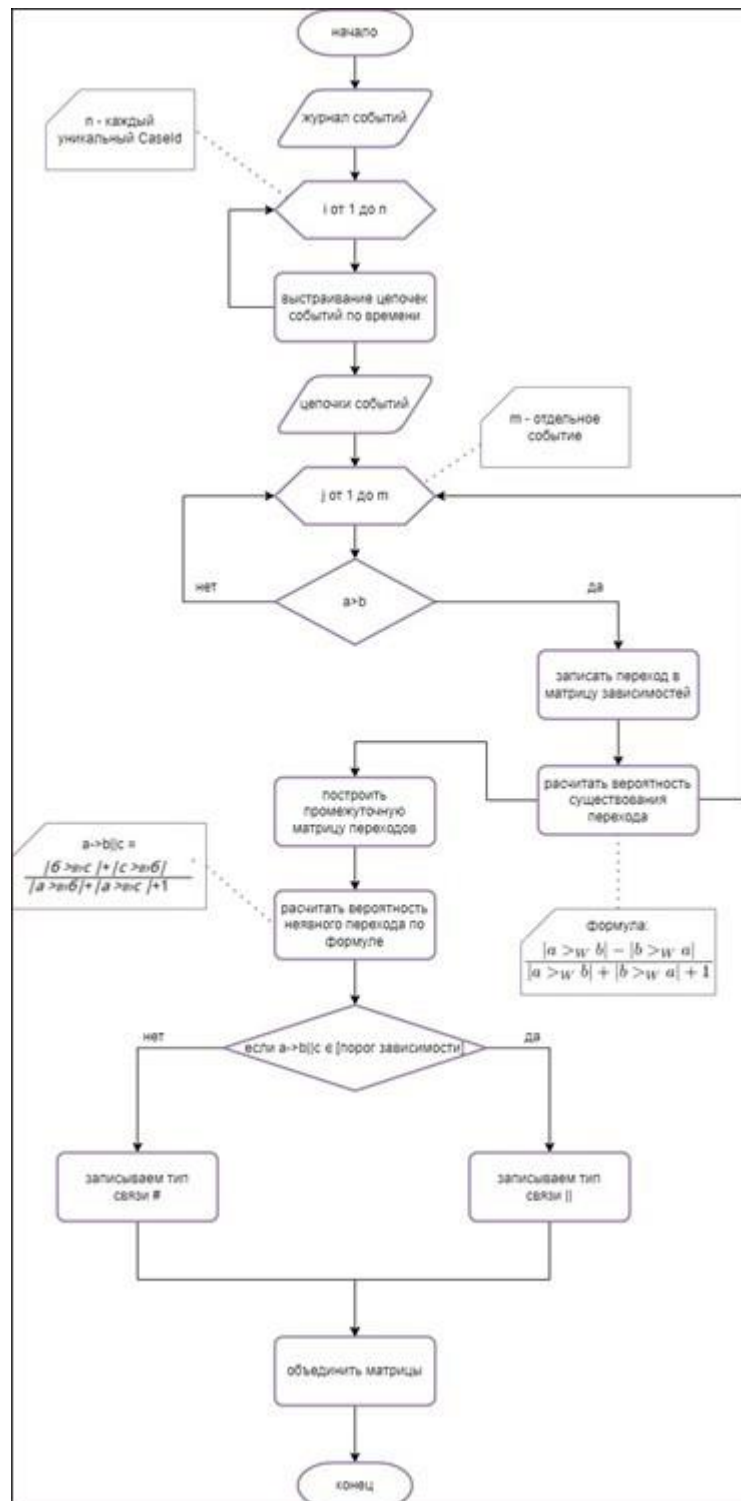


Рисунок 29 – Работа эвристического алгоритма

Если значение вероятности больше установленного порога, то переход считается неявным, в противном случае переход считается с типом исключительного выбора, после чего получается результирующая матрица.

5.2.4 Реализация эвристического алгоритма

Альфа-алгоритм принимает в качестве аргумента журнал событий и не требует наличия параметров. Его задача заключается в последовательном проходе по всем событиям в журнале и определении типа связи для всех переходов в каждом событии. Алгоритм представлен в виде блок-схемы на рисунке 30.

Перед началом анализа и выявления шаблонов, поступивший журнал событий проходит обработку, направленную на обнаружение цепочек событий. Затем, на основе этих цепочек определяются типы связей для всех переходов.

В процессе проверки, поочередно рассматриваются два события (a и b), чтобы определить наличие следования между ними. Если следования нет, то осуществляется проверка обратной ситуации (b и a). Если обнаруживается такой переход, ему присваивается тип связи "выбор". В случае, если следование было обнаружено, дополнительно проверяется, существует ли обратное следование от текущего. Если это условие не выполняется, то тип связи определяется как "причина-следствие".

Наличие следования между событиями как в одну сторону (a и b), так и в другую (b и a) указывает на наличие параллельной связи между событиями.

На каждом этапе перебора переходов строится и обновляется матрица переходов, а также удаляются повторяющиеся переходы. По окончании обработки всех событий и их переходов, получается результирующая матрица.

5.2.5 Реализация индуктивного алгоритма

Индуктивный алгоритм принимает на вход журнал событий и параметры порога зависимости. В данном алгоритме обрабатываются 4 типа связей: причина-следствие, параллельность, исключительный выбор и цикл

Цепочки обрабатываются в алгоритме следующим образом: сначала строится граф прямого следования. После построения графа, алгоритм выполняет разрезы на этом графе.

Для каждого разреза алгоритм ищет сначала разрезы исключительного выбора ($\#$), затем причина-следствие ($\rightarrow$), затем параллельные разрезы ($\parallel$) и, наконец, повторяющиеся циклы ($\cup$). Разрез с исключительным выбором делит граф на наборы, что исключает наличие дуг, ведущих из набора в набор. За последовательное нахождение элементов в каждом следующем друг за другом наборе отвечает разрез последовательности. Это говорит о том, что каждый текущий набор элементов является прямым продолжением элементов предшествующих наборов. Для достижения связи между действиями разных наборов применяется деление графа разрезом параллельности. Разрез повторяющегося цикла разделяет граф на наборы. Один из них отвечает за все выполняемые события, а другие оставшиеся за повторяющиеся события. Выполняемые события стартуют с цикла повторения и им же заканчиваются. Повторяющиеся события идут перед выполняемым действием первого набора, следуя за конечным действием.

Блок-схема индуктивного алгоритма показана на рисунке 30.

Выбирая вид разреза, алгоритм перебирает граф, занимаясь поиском нетривиального разреза с исключительным выбором. В случае нахождения такого разреза алгоритм делит граф на два или более наборов, что позволяет применить максимальное уменьшение исключительного выбора. Максимальный разрез ограничивает возможности деления графа другими разрезами.

Алгоритм приступает к поиску нетривиального разреза "причина-следствие", только когда нетривиальных разрезов с исключительным выбором больше нет. Граф сокращается на каждом этапе работы алгоритма до изолирования всех событий. Когда каждое событие изолировано, алгоритм создает дерево событий из удаленной информации, учитывая типы связей, которые соединяют эти события.

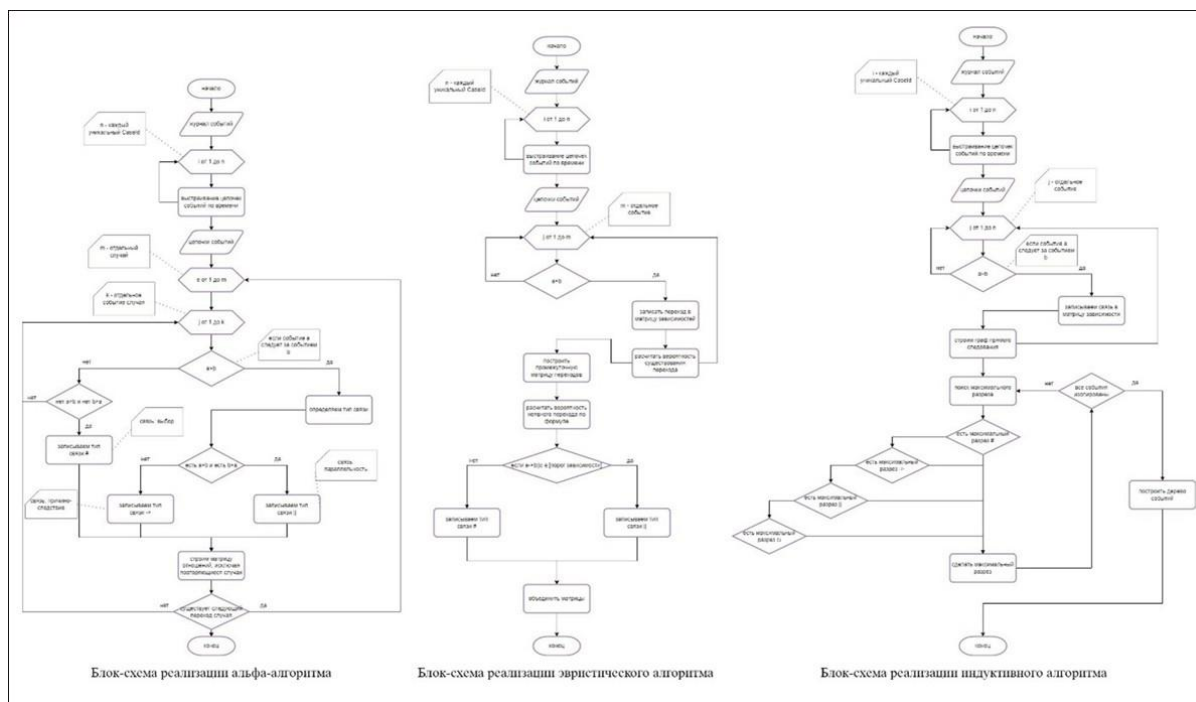


Рисунок 30 – Схемы реализации алгоритмов

5.2.6 Формирование ключевых особенностей построения алгоритма

Стандартные алгоритмы извлечения, как правило, сфокусированы на выявлении составных событий типа "И" и "ИЛИ". Однако, для выявления более сложных процессов необходимо расширить этот набор. Для этого требуется использовать маркеры типовых событий, что помогает ускорить обработку и сортировку процессов. Также одним из ключевых этапов в составлении собственного алгоритма обработки является борьба с шумами. Основываясь на обзоре основных алгоритмов выявления составных событий, можно сделать вывод, что залогом успешной работы алгоритма является предварительная обработка данных. В процессе предварительной обработки выполняются несколько шагов, обеспечивающих надежность и качество анализа.

Первым шагом становится удаление дубликатов. Путем проверки наличия повторяющихся записей в журнале событий и их удалением. Этот процесс помогает избежать искажений и дублирования информации.

Далее следует фильтрация выбросов путем определения аномальных значений, используя статистические методы.

Следующий шаг – сглаживание данных. Применение методов сглаживания, таких как скользящее среднее или экспоненциальное сглаживание, помогает уменьшить шум и колебания в данных.

После сглаживания выполняется их нормализация и масштабирование. Приведение данных к определенному диапазону или масштабу позволяет обеспечить сопоставимость между различными признаками или значениями, что улучшает сравнение и анализ данных.

Последней операцией является группировка событий. Группировка событий может осуществляться на основе различных факторов, таких как временные параметры, схожие признаки или логические связи. Например, события, произошедшие в определенном временном окне или имеющие схожие характеристики, могут быть объединены в одну группу для дальнейшего анализа.

Создавая свой алгоритм извлечения необходимо учитывать контекст, в котором происходят события. Принимать во внимание данные о времени, ролях участников, атрибутах событий и других контекстуальных параметрах. Алгоритм должен быть способен анализировать и связывать информацию о контексте с последовательностью событий для определения сложных шаблонов и зависимостей.

ЗАКЛЮЧЕНИЕ

В результате выполнения магистерской работы были проанализированы большие языковые модели, которые были использованы в качестве AI-ассистента для проведения собеседования, задача которых состояла анализировать навыки кандидата, генерировать вопросы по конкретной специальности и оценивать знания кандидата. Были взяты уже обученные модели от разных компаний для выявления наиболее подходящего продукта под поставленные цели. В итоге были выявлены модели, которые хорошо справились с задачей формирования отчетов по финансовым транзакциям. И было выявлено, что на данном этапе развития больших языковых моделей, существует очень много различных решений, являющимися бесплатными с возможностью коммерческого использования, которые можно интегрировать для реализации задач в собственных программных продуктах.

Исходя из вышесказанного, можно сделать вывод, что за время выполнения выпускной квалификационной работы были выполнены все поставленные задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. HH Статистика: сервис открытой аналитики рынка труда // hh.ru [САЙТ]. – URL: https://stats.hh.ru/?countrySalaryDynamicChartProfArea=information_technology&hhIndexProfArea=information_technology (дата обращения: 19.03.2026).
2. Qwen3-VL-8B // Baidu Baika [САЙТ]. – URL: <https://baike.baidu.com/ru/item/Qwen3-VL-8B/785925> (дата обращения: 07.05.2025).
3. Qwen3-8B // Hugging Face [САЙТ]. – URL: <https://huggingface.co/Qwen/Qwen3-8B> (дата обращения: 11.06.2025).
4. Ministral 3 8B: Specifications and GPU VRAM Requirements // APXML [САЙТ]. – URL: <https://apxml.com/models/ministral-3-8b> (дата обращения: 03.09.2025).
5. Mistral 7B // Mistral AI [САЙТ]. – URL: <https://mistral.ai/news/announcing-mistral-7b> (дата обращения: 17.11.2025).
6. Подробное руководство по созданию архитектуры мобильных приложений. [Электронный ресурс]. URL: <https://scand.com/ru/company/blog/the-ultimate-guide-to-mobile-application-architecture/> (дата обращения: 08.12.2023).
7. Создание приложений на базе технологии Parse. [Электронный ресурс]. URL: https://github.com/codepath/android_guides/wiki/Building-Data-driven-Apps-with-Parse (дата обращения: 05.02.2023).
8. Канер С. Тестирование программного обеспечения. – Санкт-Петербург.: Питер, 2018. – ISBN 979-5-4462-0773-7. – 129 с.
9. Скороход С. Управление проектами средствами Microsoft Project. Курс лекций. – Москва.: Интуит НОУ, 2016. – 319 с.
10. Attention Is All You Need // arXiv. – 2017. – URL: <https://arxiv.org/pdf/1706.03762> (дата обращения: 20.05.2025).
11. Ньюмен Сэм. Создание микросервисов. – Санкт-Петербург: Питер, 2023. – 23 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Разработка автоматизированной системы принятия решений для обработки
финансовых транзакций

Листинг программы

https://github.com/schewaev/Ai_agent.

Файл `assessmentApi.ts`

```
import { http } from '../../../shared/api/http'
import type {
  AssessmentAttemptDetails,
  AssessmentAttemptListItem,
  AssessmentListItem,
  CreateAssessmentRequest,
  CreateAssessmentResponse,
  PublicAssessment,
  StartPublicAssessmentRequest,
  StartPublicAssessmentResponse,
  SubmitAssessmentAnswerRequest,
  SubmitAssessmentAnswerResponse,
  SubmitAssessmentVoiceAnswerRequest,
} from '../../../types/assessment.types'

export async function createAssessment(
  payload: CreateAssessmentRequest,
): Promise<CreateAssessmentResponse> {
  const { data } = await
  http.post<CreateAssessmentResponse>('/api/v1/assessments',
  payload)
  return data
}

export async function getMyAssessments():
Promise<AssessmentListItem[]> {
  const { data } = await
  http.get<AssessmentListItem[]>('/api/v1/assessments/my')
  return data
}

export async function getAssessmentAttempts(
  assessmentId: string,
): Promise<AssessmentAttemptListItem[]> {
  const { data } = await
  http.get<AssessmentAttemptListItem[]>(
    `/api/v1/assessments/${assessmentId}/attempts`,
  )
  return data
}

export async function getAssessmentAttemptDetails(
  attemptId: string,
): Promise<AssessmentAttemptDetails> {
  const { data } = await http.get<AssessmentAttemptDetails>(
    `/api/v1/assessments/attempts/${attemptId}`,
  )
  return data
}
```

```

}

export async function getPublicAssessment(publicToken: string):
Promise<PublicAssessment> {
    const { data } = await http.get<PublicAssessment>(
        `/api/v1/public/assessments/${publicToken}`,
    )
    return data
}

export async function startPublicAssessment(
    publicToken: string,
    payload: StartPublicAssessmentRequest,
): Promise<StartPublicAssessmentResponse> {
    const { data } = await
http.post<StartPublicAssessmentResponse>(
    `/api/v1/public/assessments/${publicToken}/start`,
    payload,
)
    return data
}

export async function submitPublicAssessmentAnswer(
    attemptId: string,
    payload: SubmitAssessmentAnswerRequest,
): Promise<SubmitAssessmentAnswerResponse> {
    const { data } = await
http.post<SubmitAssessmentAnswerResponse>(
    `/api/v1/public/assessment-
attempts/${attemptId}/answer`,
    payload,
)
    return data
}

export async function submitPublicAssessmentVoiceAnswer(
    attemptId: string,
    payload: SubmitAssessmentVoiceAnswerRequest,
): Promise<SubmitAssessmentAnswerResponse> {
    const { data } = await
http.post<SubmitAssessmentAnswerResponse>(
    `/api/v1/public/assessment-
attempts/${attemptId}/answer/voice`,
    payload,
)
    return data
}

export async function getPublicAssessmentResult(token: string):
Promise<AssessmentAttemptDetails> {
    const { data } = await http.get<AssessmentAttemptDetails>(
        `/api/v1/public/assessment-results/${token}`,
    )
}

```

```

        return data
    }

```

Файл GatewaySecurityConfig.java

```

package com.example.gateway.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.config.annotation.method.configurat
ion.EnableReactiveMethodSecurity;
import
org.springframework.security.config.annotation.web.reactive.Enab
leWebFluxSecurity;
import
org.springframework.security.config.web.server.ServerHttpSecurit
y;
import
org.springframework.security.web.server.SecurityWebFilterChain;

@Configuration
@EnableWebFluxSecurity
@EnableReactiveMethodSecurity
public class GatewaySecurityConfig {

    @Bean
    public SecurityWebFilterChain
springSecurityFilterChain(ServerHttpSecurity http) {
        return http
            .csrf(ServerHttpSecurity.CsrfSpec::disable)
            .authorizeExchange(exchange -> exchange

.pathMatchers(HttpMethod.OPTIONS, "**").permitAll()

            .pathMatchers(HttpMethod.POST,
"/api/v1/auth/register").permitAll()
            .pathMatchers(HttpMethod.POST,
"/api/v1/auth/login").permitAll()
            .pathMatchers(HttpMethod.POST,
"/api/v1/auth/refresh").permitAll()

            .pathMatchers(HttpMethod.GET,
"/api/v1/public/assessments/**").permitAll()
            .pathMatchers(HttpMethod.POST,
"/api/v1/public/assessments/**").permitAll()
            .pathMatchers(HttpMethod.POST,
"/api/v1/public/assessment-attempts/**").permitAll()
            .pathMatchers(HttpMethod.GET,
"/api/v1/public/assessment-results/**").permitAll()

            .pathMatchers(HttpMethod.POST,
"/api/v1/interviews/start").permitAll()

```

```

        .pathMatchers(HttpMethod.POST,
"/api/v1/interviews/answer").permitAll()
        .pathMatchers(HttpMethod.POST,
"/api/v1/interviews/answer/voice").permitAll()
        .pathMatchers(HttpMethod.POST,
"/api/v1/interviews/final-evaluation").permitAll()
        .pathMatchers(HttpMethod.POST,
"/api/v1/interviews/stop").permitAll()
        .pathMatchers(HttpMethod.GET,
"/api/v1/interviews/**").permitAll()

.pathMatchers("/actuator/health").permitAll()
        .pathMatchers("/swagger-
ui.html", "/swagger-ui/**", "/v3/api-docs/**").permitAll()

.pathMatchers("/api/v1/reports/**").authenticated()

        .anyExchange().authenticated()
    )
    .oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt -
> {
        )))
    .build();
}
}

```

Файл Controller.java

```

package com.example..controller;

import com.example..dto.request.*;
import com.example..dto.response.FinalEvaluationResponse;
import com.example..dto.response.SessionStatusResponse;
import com.example..dto.response.StartInterviewResponse;
import com.example..dto.response.SubmitAnswerResponse;
import com.example..service.InterviewService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/v1/interviews")
@RequiredArgsConstructor
public class Controller {

    private final Service iService;

    @PostMapping("/start")
    public StartResponse startI(@Valid @RequestBody StartRequest

```

```

request) {
    return Service.startInterview(request);
}

@PostMapping("/answer")
public SubmitAnswerResponse submitAnswer(@Valid @RequestBody
SubmitAnswerRequest request) {
    return Service.submitAnswer(request);
}

@PostMapping("/answer/voice")
public SubmitAnswerResponse submitAnswer(@Valid @RequestBody
SubmitAnswerRequest request) {
    return Service.submitVoiceAnswer(request);
}

@PostMapping("/final-evaluation")
public FinalEvaluationResponse finalEvaluation(@Valid
@RequestBody FinalEvaluationRequest request) {
    return interviewService.finalEvaluation(request);
}

@PostMapping("/stop")
public SubmitAnswerResponse stopInterview(@Valid
@RequestBody StopInterviewRequest request) {
    return Service.stopInterview(request);
}

@GetMapping("/history")
public List<Map<String, Object>> history() {
    return Service.history();
}

@GetMapping("/{sessionId}")
public SessionStatusResponse getStatus(@PathVariable String
sessionId) {
    return Service.getStatus(sessionId);
}
}

```

Файл ReportService.java

```

package com.example.reportservice.service;

import com.example.reportservice.dto.CreateReportRequest;
import com.example.reportservice.dto.ReportResponse;
import com.example.reportservice.entity.ReportEntity;
import com.example.reportservice.repository.ReportRepository;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.server.ResponseStatusException;

```

```

import org.springframework.http.HttpStatus;

import java.security.SecureRandom;
import java.time.OffsetDateTime;
import java.util.Base64;
import java.util.List;
import java.util.Map;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class ReportService {
    private final ReportRepository repository;
    private final ObjectMapper objectMapper;
    private final SecureRandom secureRandom = new
SecureRandom();

    @Transactional
    public ReportResponse create(CreateReportRequest request) {
        ReportEntity entity = new ReportEntity();
        entity.setId(UUID.randomUUID());
        entity.setSourceType(request.getSourceType());
        entity.setSourceId(request.getSourceId());
        entity.setOwnerUserId(request.getOwnerUserId());
        entity.setCandidateName(request.getCandidateName());
        entity.setCandidateEmail(request.getCandidateEmail());
        entity.setOverallScore(request.getOverallScore());
        entity.setRecommendation(request.getRecommendation());
        entity.setSummary(request.getSummary());
        entity.setPublicToken(generateToken());
        entity.setCreatedAt(OffsetDateTime.now());
        try {
entity.setReportJson(objectMapper.writeValueAsString(request.get
ReportPayload()));
        } catch (Exception e) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Invalid report
payload");
        }
        return toResponse(repository.save(entity));
    }

    public ReportResponse get(UUID id) {
        return toResponse(repository.findById(id)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Report not
found")));
    }

    public ReportResponse getPublic(String token) {
        return toResponse(repository.findByPublicToken(token)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Report not
found")));
    }
}

```

```

        public List<ReportResponse> byOwner(UUID ownerId) {
            return
repository.findByOwnerIdOrderByCreatedAtDesc(ownerId).stream().map(this::toResponse).toList();
        }
        private ReportResponse toResponse(ReportEntity e) {
            ReportResponse r = new ReportResponse();
            r.setId(e.getId());
            r.setSourceType(e.getSourceType());
            r.setSourceId(e.getSourceId());
            r.setOwnerId(e.getOwnerId());
            r.setCandidateName(e.getCandidateName());
            r.setCandidateEmail(e.getCandidateEmail());
            r.setPublicToken(e.getPublicToken());
            r.setOverallScore(e.getOverallScore());
            r.setRecommendation(e.getRecommendation());
            r.setSummary(e.getSummary());
            r.setCreatedAt(e.getCreatedAt());
            try {

r.setReportPayload(objectMapper.readValue(e.getReportJson(), new
TypeReference<Map<String, Object>>() {}));
            } catch (Exception ex) {
                r.setReportPayload(Map.of());
            }
            return r;
        }

        private String generateToken() {
            byte[] bytes = new byte[32];
            secureRandom.nextBytes(bytes);
            return
Base64.getUrlEncoder().withoutPadding().encodeToString(bytes);
        }
    }
}

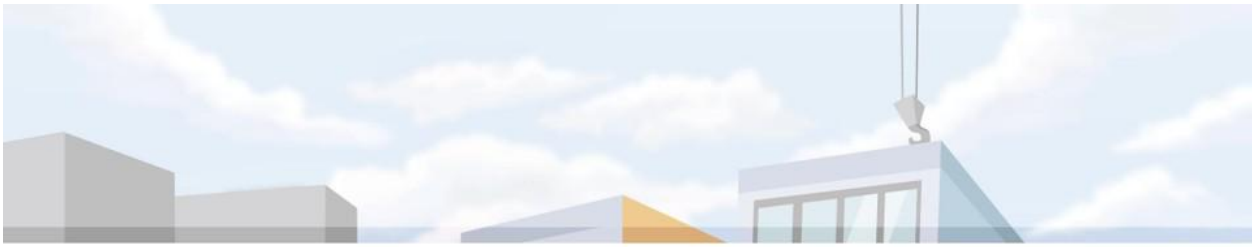
```

ПРИЛОЖЕНИЕ Б

(обязательное)

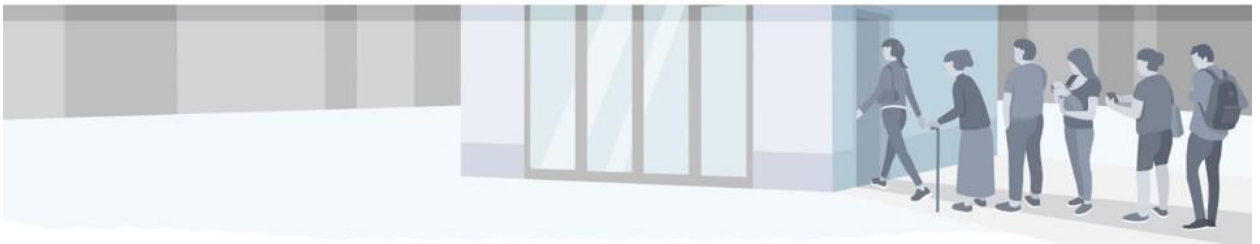
Разработка автоматизированной системы принятия решений для обработки
финансовых транзакций

Презентация



Доклад на тему: Разработка автоматизированной системы принятия решений для обработки финансовых транзакций

Автор работы:
студент группы 24ВВим1 Щеваев Михаил Владимирович
Научный руководитель:
к.т.н., доцент, Гурьянов Лев Вячеславович



Задачи и цель работы

- Исследование области
больших языковых моделей
- Внедрение модели в
приложение
- Тестирование найденных
решений
- Реализация приложения





Исследуемые LLM-модели

Ключевые требования:
локальное использование
русского языка,
коммерчески свободная
лицензия, решение
задачи в соответствии с
инструкциями

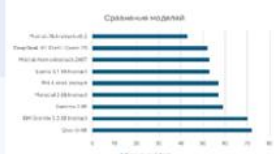


Тестирование LLM-моделей

Распределение мест при
ручном тестировании LLM:

1. Ministral 3 8B
2. Mistral-Nemo 12B
3. Qwen 3 8B
4. Granite 8B
5. Mistral 7B
6. Llama 3.1 8B

Результаты тестирования моделей в
бенчмарках, баллы выставлены с
помощью механизма взвешивания



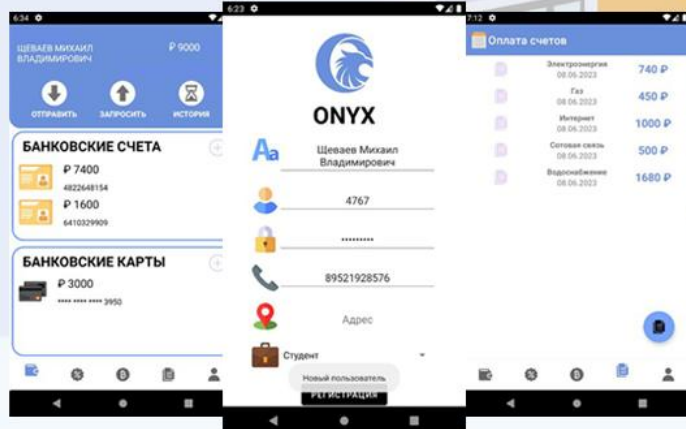
Распределение мест при ручном
тестировании LLM в приложении:

1. Ministral 3 8B
2. Mistral-Nemo 12B
3. Qwen 3 8B
4. Granite 8B
5. Mistral 7B
6. Llama 3.1 8B





Реализация мобильного приложения



Реализация мобильного приложения

Акт выполненных работ

16.05.2025

Исполнители: ИП Голобурдин Алексей Анатольевич

Заказчик: ПАО «МТС»

№	Наименование товара	Ед. изм.	Кол-во	Цена, Р
1	Обучение студента	шт	1	170 000

Общая стоимость выполненных работ, оказанных услуг семьдесят тысяч. Заказчик не имеет претензий по срокам услуг

ПАО «МТС»

ИНН: 7740000076

ОГРН: 1027700149124

Юридический адрес: Россия, Москва, ул.

Марксистская, д. 4, стр. 1

Банк: ПАО Сбербанк

БИК: 044525225

Расчетный счет: 40702810938000001510

Корр. счет: 30101810400000000225

```
32 name: str # наименование банка
33 BIC: str # BIC
34 current_account: str # расчетный счет
35 corporate_account: str # корреспондентский счет
36
37
38
39 class Customer:
40     """Заказчик"""
41     name: str # полное название юридического лица, например, ООО «Рога и копыта»
42     INN: str # INN
43     OGRN: str # ОГРН или ОГРНИП
44     address: str # юридический адрес
45     signatory: str # подписант
46     bank: Bank # банковские реквизиты заказчика
47
48
49 class Job:
50     """Задача"""
51     task: str # выполняемая задача
52     price: int # цена за задачу
53
54
55 @tool
56 def generate_pdf_act(customer: Customer, jobs: list[Job]) -> None:
57     """
58     Генератор PDF-акта, в котором заполнены данные
59     клиента, его банковские реквизиты, а также выполненные задачи
60     """
61     Args:
62         customer (Customer): данные клиента
63         jobs (list[Job]): список выполненных задач для внесения в акт
64     Returns:
65         None
66     """
67     print(f'generate_pdf_act({asdict(customer)}), ({list(map(lambda j: asdict(j), jobs))})')
68
69
70 class LLMAgent:
71     def __init__(self, model: LanguageModelLike, tools: Sequence[BaseTool]):
72         self.model = model
73         self.agent = create_react_agent(
74             model,
75             tools=tools,
76             checkpoint=InMemorySaver())
77
78
79 main.py 66.5 31%
```



Заключение доклада

- Исследованы большие языковые модели
- Проведено тестирование LLM-моделей
- Выбрана наиболее подходящая модель, которая была внедрена в приложение
- Разработана программа, позволяющая формировать финансовые отчеты по описанию

